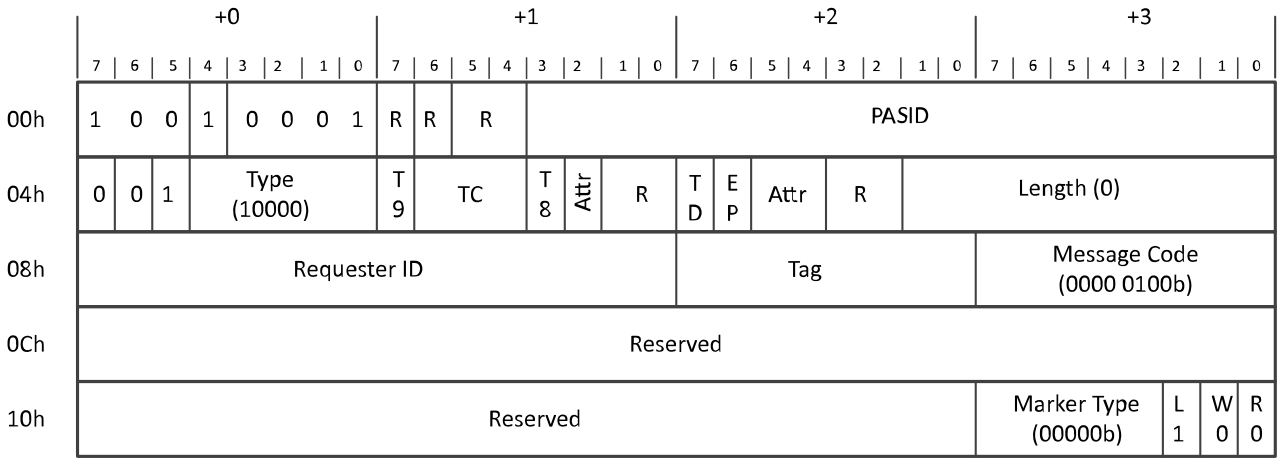


Figure 10-17 Stop Marker Message

A Stop Marker Message is encoded as a Page Request Message that contains a PASID TLP Prefix with the following exceptions:

- The L, W and R fields contain 1b, 0b and 0b respectively.
- The Untranslated Address field and upper bits of the PRG Index field are Reserved.
- The Marker Type field contains 00000b to indicate that this is a Stop Marker Message.

The Execute Requested and Privileged Mode Requested bits in the PASID TLP Prefix are Reserved.

- The Traffic Class must be 0.
- The Relaxed Ordering attribute bit must be Clear.
- The ID-Based Ordering attribute bit may be Set.

Behavior is undefined if a Stop Marker Message is received and any of the following are true:

- Marker Type not equal to 00000b.
- No PASID TLP Prefix is present.
- The PASID value does not match an outstanding stop request.
- An incomplete Page Request Message for the PASID is outstanding (i.e. for some PRG Index, the most recently received Page Request Message did not have the L bit Set).

10.4.2 Page Request Group Response Message

System hardware and/or software communicate with a Function's page request interface via PRG Response Messages. A PRG Response Message is used by a host to signal the completion of a PRG, or the catastrophic failure of the interface. A single PRG Response Message is issued in response to a PRG, independent of the number of page requests associated with the PRG. There is no mechanism for indicating a partial request completion or partial request failure. If any of the pages associated with a given PRG cannot be satisfied, then the request is considered to have failed and the reason for the failure is supplied in the PRG Response Message. The host has no obligation to partially satisfy a multi-page request. If one of the requested pages cannot be made resident, then the entire request can, but need not, be discarded. That is,

the residence of pages that share a PRG with a failed page request, but that are not associated with the failure, is indeterminate from the Function's perspective.

There are four possible Page Request failures:

1. The requested page is not a valid Untranslated Address.
2. PASID TLP Prefix support exists, the Page Request has a PASID TLP Prefix, and either PASID TLP Prefix usage is not enabled for this request, the PASID value is not valid, or the Execute Requested bit is Set when R is Clear.[171]
3. The requested page does not have the requested access attributes (including Execute permission and/or Privileged Mode access when the Page Request has a PASID TLP Prefix).
4. The system is, for an unspecified reason, unable to respond to the request. This response is terminal (the host may no longer respond to any page requests and may not supply any further replies to the Function until the Function's page request interface has been reset). For example, a request that violates a Function's assigned request limit or overflows the RC's buffering capability may cause this type of failure.

A Function's response to Page Request failure cases 1, 2, and 3 above is implementation dependent. The failure is not necessarily persistent, that is, a failed request may, in some instances succeed if re-issued. The range of possibilities precludes the precise specification of a generalized failure behavior, though on a per Function basis, the response to a failure will be an implementation dependent behavior.

All responses are sent to their associated Functions via PRG Response Messages. A Function must be capable of sinking multiple consecutive messages without losing any information. To avoid deadlock, a Function must able to process PRG Response Messages for all of the Function's outstanding Page Request Messages without depending on the Function sending or receiving any other TLP.[172] A PRG Response Message is an ID routed PCIe message. The only Page Request Interface specific fields in this message are the Response Code and PRG. All other fields are standard PCIe message fields. (Note: these messages are routed based on the ID in bytes 8 and 9; with bytes 4 and 5 containing the host's RID.)

Receipt of a PRG Response Message that contains a PRG Index that is not currently outstanding at a Function shall result in the UPRGI flag in the Page Request Extended Capability being Set and in the issuance of an Unexpected Response (UR) by the Function containing the Page Request Extended Capability. With the exception of setting the UPRGI flag, a Function treats receipt of an unexpected PRG Index in exactly the same manner that it treats receipt of a standard PCIe read completion for which there is no outstanding request.

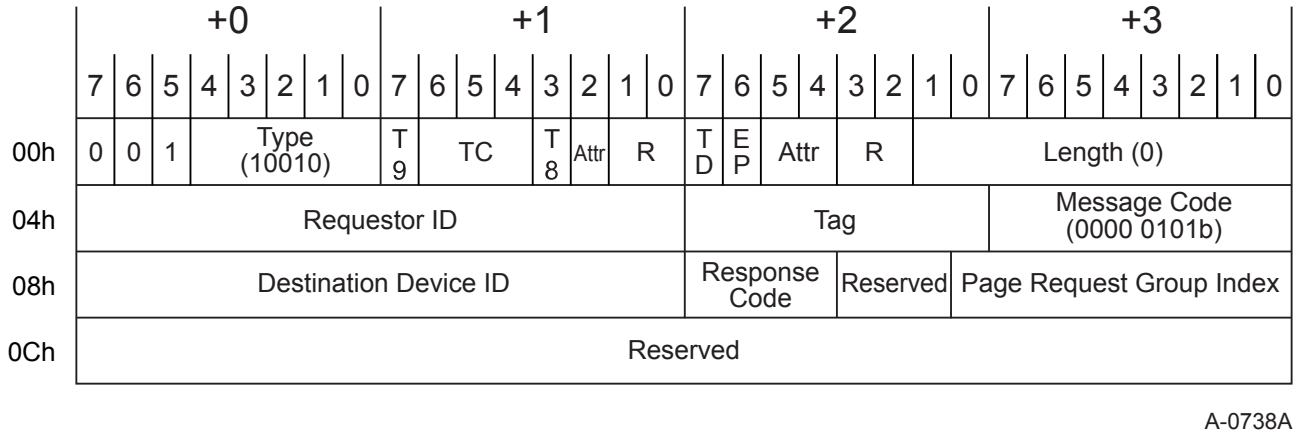


Figure 10-18 PRG Response Message

171. Behavior when PASID TLP Prefix support does not exist is defined in Section 6.20.1 .
172. For example, processing a PRG Response Message that causes the Function to send a TLP Upstream must not block processing of subsequent downstream TLPs even if the Upstream TLP is delayed by flow control.

Table 10-6 PRG Response Message Data Fields

Field	Meaning
Page Request Group Index	**Page Request Group Index** - This field contains a Function supplied index to which the RC is responding. A given PRG Index will receive exactly one response per instance of PRG (with the possible exception of a Response Failure).
Response Code	**Response Code** - This field contains the response type of the associated PRG. The encodings are presented in Section 10.4.2.1 .

10.4.2.1 Response Code Field

The values and meaning for the Response Code field are listed in Table 10-7 .

Table 10-7 Response Codes

Value	Status	Meaning
0000b	Success	All pages within the associated PRG were successfully made resident.
0001b	Invalid Request	One or more pages within the associated PRG do not exist or requests access privilege(s) that cannot be granted. Unless the page mapping associated with the Function is altered, re-issuance of the associated request will never result in success.
1110b:0010b	Unused	Unused Response Code values. A Function receiving such a message shall process it as if the message contained a Response Code of Response Failure.
1111b	Response Failure	One or more pages within the associated request group have encountered/caused a catastrophic error. This response disables the Page Request Interface at the Function. Any pending page requests for other PRGs will be satisfied at the convenience of the host. The Function shall ignore any subsequent PRG Response Messages, pending re-enablement of the Page Request Interface.

10.4.2.2 PASID TLP Prefix Usage on PRG Responses

If a Page Request has a PASID TLP Prefix, the corresponding PRG Response Message may optionally contain one as well.

If the PRG Response PASID Required bit is Clear, PRG Response Messages do not have a PASID TLP Prefix.

If the PRG Response PASID Required bit is Set, PRG Response Messages have a PASID TLP Prefix if the Page Request also had one. The Function is permitted to use the PASID value from the prefix in conjunction with the PRG Index to match requests and responses.

In PASID TLP Prefixes attached to PRG Response Messages, the Execute Requested and Privileged Mode Requested bits are Reserved and the PASID value is copied from the PASID value of the Page Request.

10.5 ATS Configuration

10.5.1 *ATS Extended Capability*

Each Function that supports ATS (capable of generating Translation Requests) must have the ATS Extended Capability structure in its extended configuration space. It is permitted to be implemented by Endpoints or Root Complex Integrated Endpoints.

Figure 10-19 details allocation of the register fields in the ATS Extended Capability structure.

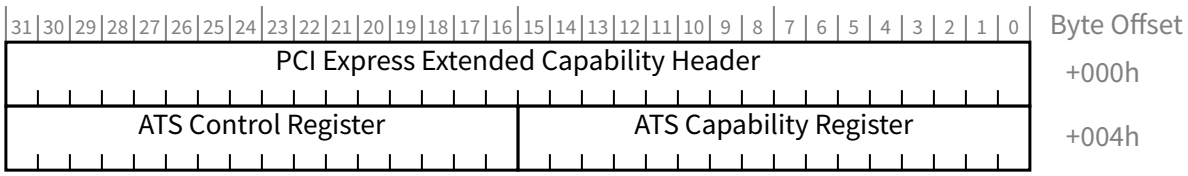


Figure 10-19 ATS Extended Capability Structure

10.5.1.1 ATS Extended Capability Header (Offset 00h)

Figure 10-20 details allocation of the register fields in the ATS Extended Capability Header; Table 10-8 provides the respective field definitions.

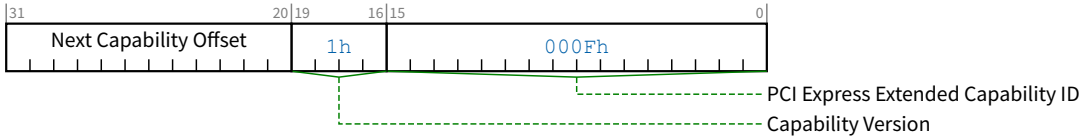


Figure 10-20 ATS Extended Capability Header

Table 10-8 ATS Extended Capability Header

Bit Location	Register Description	Attributes
15:0	***PCI Express Extended Capability ID*** - Indicates the ATS Extended Capability structure. This field must return a Capability ID of "000Fh" indicating that this is an ATS Extended Capability structure.	RO
19:16	***Capability Version*** - This field is a PCI-SIG defined version number that indicates the version of the Capability structure present. Must be "1h" for this version of the specification.	RO
31:20	***Next Capability Offset*** - The offset to the next PCI Extended Capability structure or 000h if no other items exist in the linked list of capabilities.	RO

10.5.1.2 ATS Capability Register (Offset 04h)

Figure 10-21 details the allocation of register fields of an ATS Capability Register; Table 10-9 provides the respective bit definitions.

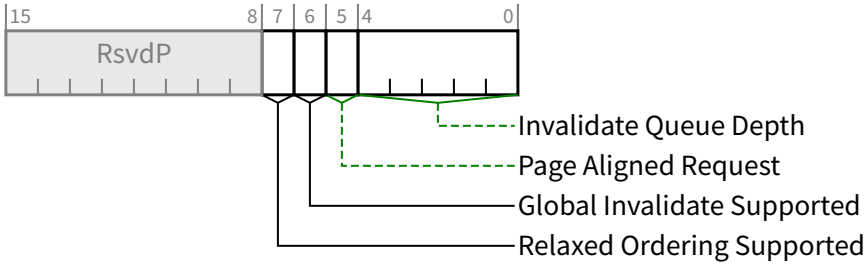


Figure 10-21 ATS Capability Register (Offset 04h)

Table 10-9 ATS Capability Register (Offset 04h)

Bit Location	Register Description	Attributes
4:0	***Invalidate Queue Depth*** - The number of Invalidate Requests that the Function can accept before putting backpressure on the Upstream connection. If 0 0000b, the Function can accept 32 Invalidate Requests.	RO
5	***Page Aligned Request*** - If Set, indicates the Untranslated Address is always aligned to a 4096 byte boundary. Setting this field is recommended. This field permits software to distinguish between implementations compatible with this specification and those compatible with an earlier version of this specification in which a Requester was permitted to supply anything in bits [11:2].	RO
6	***Global Invalidate Supported*** - If Set, the Function supports Invalidation Requests that have the Global Invalidate bit Set. If Clear, the Function ignores the Global Invalidate bit in all Invalidate Requests (see Section 10.3.8). This bit is 0b if the Function does not support the PASID TLP Prefix.	RO
7	***Relaxed Ordering Supported*** - If Set, indicates this Function is permitted to Set the RO bit in Translation Requests when Enable Relaxed Ordering bit is Set.	RO

10.5.1.3 ATS Control Register (Offset 06h)

Figure 10-22 details the allocation of register fields of an ATS Control Register; Table 10-10 provides the respective bit definitions.

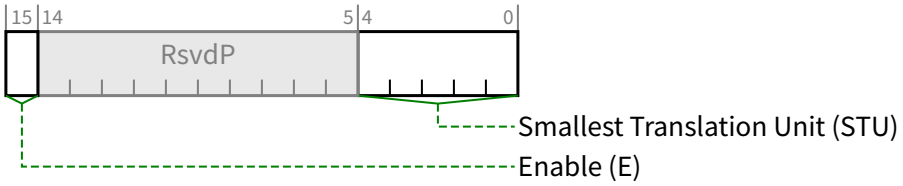


Figure 10-22 ATS Control Register

Table 10-10 ATS Control Register

Bit Location	Register Description	Attributes
4:0	***Smallest Translation Unit (STU)*** - This value indicates to the Function the minimum number of 4096-byte blocks that is indicated in a Translation Completions or Invalidate Requests. This is a power of 2 multiplier and the number of blocks is 2^{STU}. A value of 0 0000b indicates one block and a value of 1 1111b indicates 2^{31} blocks (or 8 TB total) Default value is 0 0000b.	RW
15	***Enable (E)*** - When Set, the Function is enabled to cache translations. Behavior is undefined if this bit is Set and the value of the PASID Enable, Execute Permission Enable, or Privileged Mode Enable bits are changed. Default value is 0b.	RW

10.5.2 *Page Request Extended Capability* Structure

A Page Request Extended Capability Structure is used to configure the Page Request Interface mechanism. A Multi-Function Endpoint or Root Complex Integrated Endpoint Device may implement a Page Request Interface and the associated capability on any Function within the Device. For SR-IOV, a single Page Request Interface is permitted for the PF and is shared between the PF and the associated VFs. The PF implements this capability and the VFs do not. Every Page Request Interface mechanism operates independently.

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

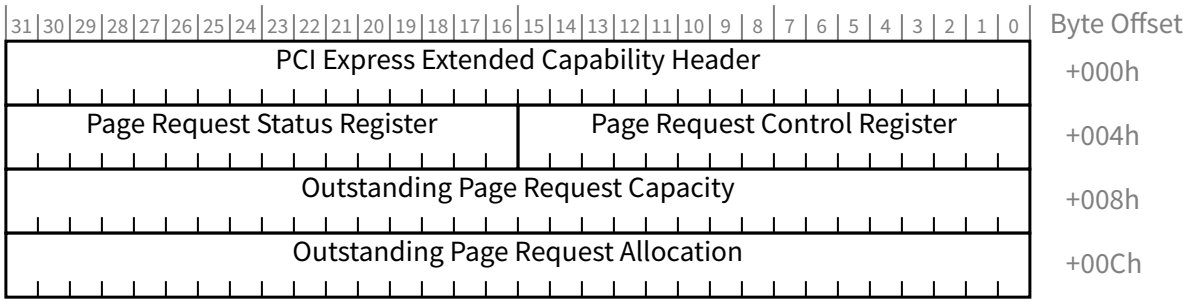


Figure 10-23 Page Request Extended Capability Structure

10.5.2.1 Page Request Extended Capability Header (Offset 00h)

Figure 10-24 details allocation of the register fields in the Page Request Extended Capability Header; Table 10-11 provides the respective field definitions.

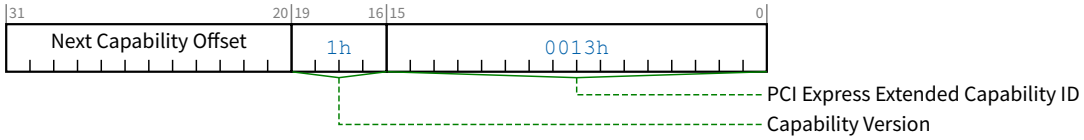


Figure 10-24 Page Request Extended Capability Header

Table 10-11 Page Request Extended Capability Header

Bit Location	Register Description	Attributes
15:0	***PCI Express Extended Capability ID*** - Indicates that the associated extended capability structure is a Page Request Extended Capability. This field must return a Capability ID of "0013h".	RO
19:16	***Capability Version*** - This field is a PCI-SIG defined version number that indicates the version of the Capability structure present. Must be "1h" for this version of the specification.	RO
31:20	***Next Capability Offset*** - The offset to the next PCI Extended Capability structure or 000h if no other items exist in the linked list of capabilities.	RO

10.5.2.2 Page Request Control Register (Offset 04h)

Figure 10-25 details allocation of the register fields in the Page Request Control Register; Table 10-12 provides the respective field definitions.

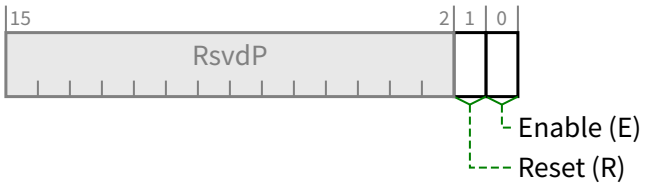


Figure 10-25 Page Request Control Register

Table 10-12 Page Request Control Register

Bit Location	Register Description	Attributes
0	***Enable (E)*** - This field, when set, indicates that the Page Request Interface is allowed to make page requests. If this field is Clear, the Page Request Interface is not allowed to issue page requests. If both this field and the Stopped field are Clear, then the Page Request Interface will not issue new page requests, but has outstanding page requests that have been transmitted or are queued for transmission. When the Page Request Interface is transitioned from not-Enabled to Enabled, its status flags (Stopped, Response Failure, and Unexpected Response flags) are cleared. Enabling a Page Request Interface that has not successfully Stopped has indeterminate results. Default value is 0b.	RW

Bit Location	Register Description	Attributes
1	***Reset (R)*** - When the Enable field is clear, or is being cleared in the same register update that sets this field, writing a 1b to this field, clears the associated implementation dependent page request credit counter and pending request state for the associated Page Request Interface. No action is initiated if this field is written to 0b or if this field is written with any value while the Enable field is Set. Reads of this field return 0b	RW

10.5.2.3 Page Request Status Register (Offset 06h)

Figure 10-26 details allocation of the register fields in the Page Request Error Register; Table 10-13 provides the respective field definitions.

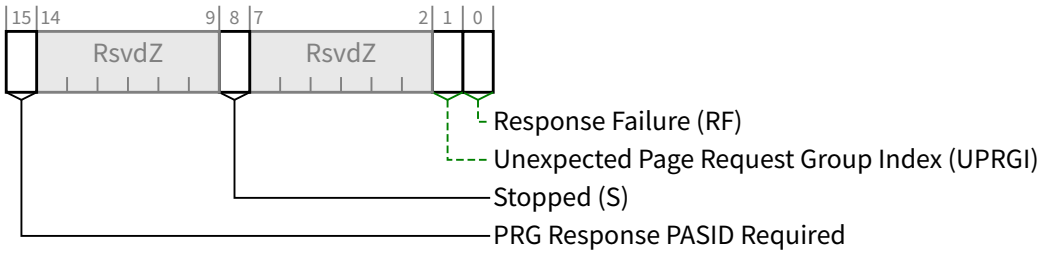


Figure 10-26 Page Request Status Register

Table 10-13 Page Request Status Register

Bit Location	Register Description	Attributes
0	***Response Failure (RF)*** - This field, when Set, indicates that the Function has received a PRG Response Message indicating a Response Failure. The Function expects no further responses from the host (any received are ignored). This field is Set by the Function and Cleared when a one is written to the field. For SR-IOV, this field is Set in the PF if any associated Function (PF or VF) receives a PRG Response Message indicating Response Failure. Default value is 0b.	RW1C
1	***Unexpected Page Request Group Index (UPRGI)*** - This field, when Set, indicates that the Function has received a PRG Response Message containing a PRG index that has no matching request. This field is Set by the Function and cleared when a one is written to the field. For SR-IOV, this field is Set in the PF if any associated Function (PF or VF) receives a PRG Response Message that does has no matching request. Default value is 0b.	RW1C
8	***Stopped (S)*** - When this field is Set, the associated page request interface has stopped issuing additional page requests and that all previously issued Page Requests have completed. When this field is Clear the associated page request interface either has not stopped or has stopped issuing new Page Requests but has outstanding Page Requests. This field is only meaningful if Enable is Clear. If Enable is Set, this field is undefined. When the Enable field is Cleared, after having been previously Set, the interface transitions to the stopping state and Clears this field. After all page requests currently outstanding at the Function(s) have	RO

Bit Location	Register Description	Attributes
	completed, this field is Set and the interface enters the disabled state. If there were no outstanding page requests, this field may be Set immediately when Enable is Cleared. Resetting the interface will cause an immediate transition to the disabled state. While in the stopping state, receipt of a Response Failure message will result in the immediate transition to the disabled state (Setting this field). For SR-IOV, this field is Set only when all associated Functions (PF and VFs) have stopped issuing page requests. Default value is 1b.	
15	***PRG Response PASID Required*** - If Set, the Function expects a PASID TLP Prefix on PRG Response Messages when the corresponding Page Requests had a PASID TLP Prefix. If Clear, the Function does not expect PASID TLP Prefixes on any PRG Response Message. Function behavior is undefined if this bit is Clear and the Function receives a PRG Response Message with a PASID TLP Prefix. Function behavior is undefined if this bit is Set and the Function receives a PRG Response Message with no PASID TLP Prefix when the corresponding Page Requests had a PASID TLP Prefix. This bit is RsvdZ if the Function does not support the PASID TLP Prefix.	RO

10.5.2.4 Outstanding Page Request Capacity (Offset 08h)

This register contains the number of outstanding page request messages the associated Page Request Interface physically supports. This is the upper limit on the number of pages that can be usefully allocated to the Page Request Interface.

This register is Read Only.

10.5.2.5 Outstanding Page Request Allocation (Offset 0Ch)

This register contains the number of outstanding page request messages the associated Page Request Interface is allowed to issue (have outstanding at any given instance).

The number of PRGs a Page Request Interface has outstanding is less than or equal to the number of request messages it has issued. For example, if system software allocates 1000 messages to a Page Request Interface then a single PRG could use all 1000 of the possible requests. Conversely, at one request per PRG the Page Request Interface would run out of PRG indices (of which there are only 512) before it consumes all its page request credits. A Page Request Interface must pre-allocate its request availability for any given PRG, that is, all the requests required by a given PRG must be available before any of the requests may be issued.

This register is Read/Write. Behavior is undefined if this register is changed while the Enable flag is set. Behavior is undefined if this register is written with a value larger than Outstanding Page Request Capacity. Default value is 0.

When PASID TLP Prefix is supported, the Request Allocation remains associated with the Function and is shared across the Function as well as all PASIDs of the Function.

Stopping a PASID does not affect any allocation used by that PASID. The system should continue to respond with PRG Response Messages in order to return Page Request and PRG Index resources to the Function (see Section 10.4.2.1).

Stop Marker Messages consume buffering but are not included in this allocation (see Section 10.4.1.2.1). Systems should provide additional buffering for Stop Marker Messages and should limit the number of outstanding Stop Marker Messages to avoid overrunning this additional buffering.

Isochronous Applications

A.1 Introduction

The design goal of isochronous mechanisms in PCI Express is to ensure that isochronous traffic receives its allocated bandwidth over a relevant time period while also preventing starvation of other non-isochronous traffic.

Furthermore, there may exist data traffic that requires a level of service falling in between what is required for bulk data traffic and isochronous data traffic. This type of traffic can be supported through the use of Port arbitration within Switches, the use of TC labels [1:7], and optional additional VC resources. Policies for assignment of TC labels and VC resources that are not isochronous-focused are outside the scope of the PCI Express specification.

Two paradigms of PCI Express communication are supported by the PCI Express isochronous mechanisms: Endpoint-to-Root-Complex communication model and peer-to-peer (Endpoint-to-Endpoint) communication model. In the Endpoint-to-Root-Complex communication model, the primary isochronous traffic is memory read and write requests to the Root Complex and read completions from the Root Complex. Figure A-1 shows an example of a simple system with both communication models. In the figure, devices A, B, called Requesters, are PCI Express Endpoints capable of issuing isochronous request transactions, while device C and Root Complex, called Completers, are capable of being the targets of isochronous request transactions. An Endpoint-to-Root-Complex communication is established between device A and the Root Complex, and a peer-to-peer communication is established between device B and device C. In the rest of this section, Requester and Completer will be used to make reference to PCI Express elements involved in transactions. The specific aspects of each communication model will be called out explicitly.

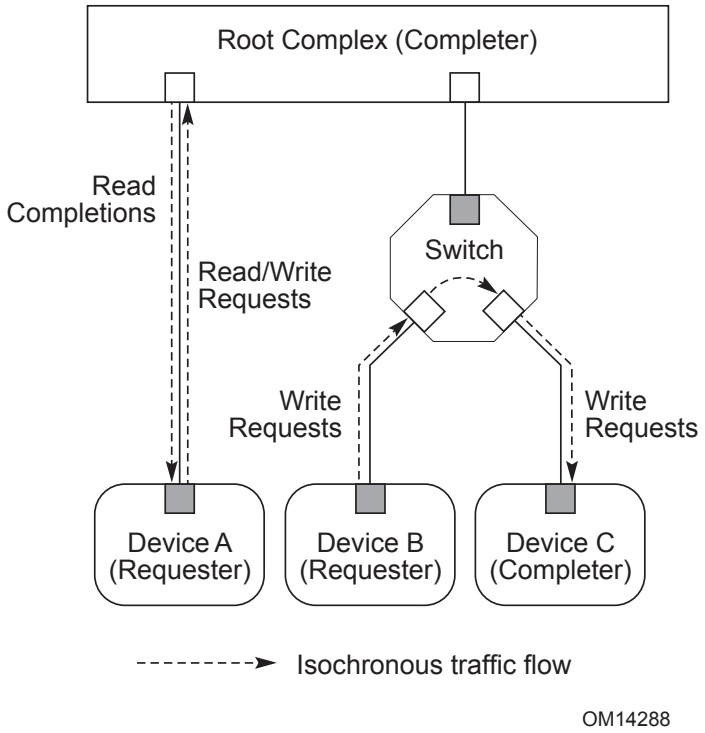


Figure A-1 An Example Showing Endpoint-to-Root-Complex and Peer-to-Peer Communication Models

Guaranteed bandwidth and deterministic latency require end-to-end configuration of fabric resources. If isochronous traffic is intermixed with non-isochronous traffic, it may not be possible to provide any guarantees/determinism as required by the application usage model. It is recommended that system software configure and assign fabric resources such that traffic intermix either does not occur or is such that the application usage model guarantees can be met. This can be accomplished by assigning dedicated VC resources and corresponding TC labels to the isochronous traffic flow(s) on a given path within the fabric.

Note that there may be one or more isochronous traffic flows per VC/TC label and it is up to system software to insure that the aggregation of these flows does not exceed the requisite bandwidth and latency requirements.

It is also possible for a fabric to support multiple isochronous traffic flows separated across multiple VC (a given flow cannot span multiple VC/TC labels).

In general, as long as the device can meet the isochronous bandwidth and latency requirements, there is nothing to preclude a single VC device from supporting isochronous traffic if multiple TC labels are supported to delineate such traffic from non-isochronous traffic within the fabric.

A.2 Isochronous Contract and Contract Parameters

In order to support isochronous data transfer with guaranteed bandwidth and deterministic latency, an isochronous contract must be established between a Requester/Completer pair and the PCI Express fabric. This contract must enforce both resource reservation and traffic regulation. Without such a contract, two basic problems, over-subscription and congestion, may occur as illustrated in Figure A-2 . When interconnect bandwidth resources are over-subscribed, the increased latency may cause failure of isochronous service and starvation of non-isochronous services. Traffic congestion occurs when flow control credits are not returned possibly due to a higher than expected/provisioned packet injection rate. This may cause excessive service latencies for both isochronous traffic and non-isochronous traffic.

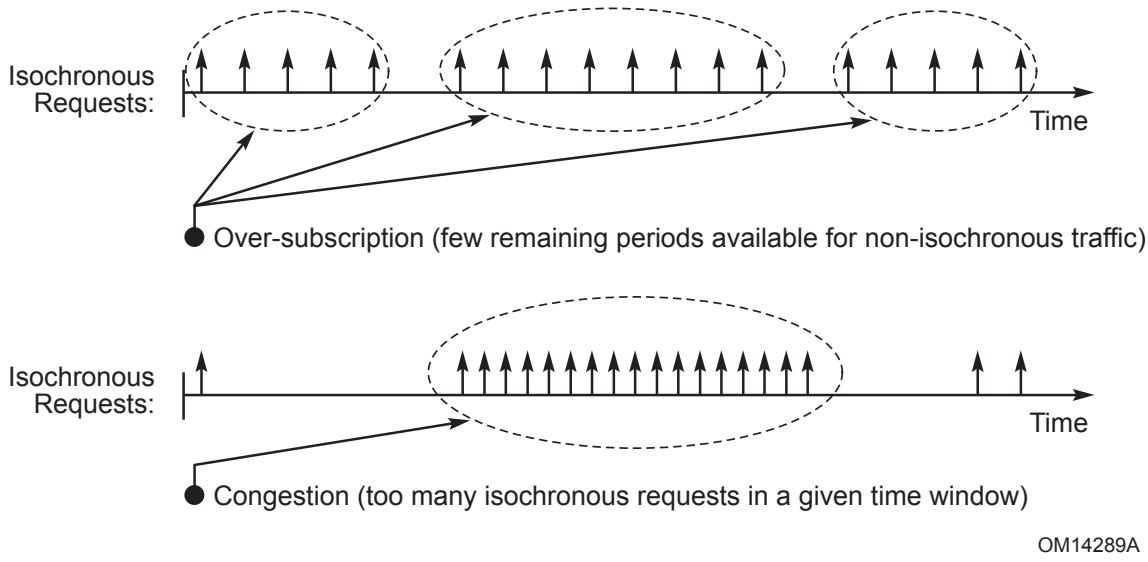


Figure A-2 Two Basic Bandwidth Resourcing Problems: Over-Subscription and Congestion

The isochronous transfer mechanism in this specification addresses these problems with traffic regulation including admission control and service discipline. Under a software managed admission control, a Requester must not issue isochronous transactions unless the required isochronous bandwidth and resource have been allocated. Specifically, the isochronous bandwidth is given by the following formula:

$$BW = \frac{N * Y}{T}$$

Equation A-1 Isochronous Bandwidth

The formula defines allocated bandwidth (BW) as a function of specified number (*N*) of transactions of a specified payload size (*Y*) within a specified time period (*T*). Another important parameter in the isochronous contract is latency. Based on the contract, isochronous transactions are completed within a specified latency (*L*). Once a Requester/ Completer pair is admitted for isochronous communication, the bandwidth and latency are guaranteed to the Requester (a PCI Express Endpoint) by the Completer (Root Complex for Endpoint-to-Root-Complex communication and another PCI Express Endpoint for peer-to-peer communication) and by the PCI Express fabric components (Switches).

Specific service disciplines must be implemented by isochronous-capable PCI Express components. The service disciplines are imposed to PCI Express Switches and Completers in such a manner that the service of isochronous requests is subject to a specific service interval (*t*). This mechanism is used to provide the method of controlling when an isochronous packet injected by a Requester is serviced. Consequently, isochronous traffic is policed in such manner that only packets that can be injected into the fabric in compliance with the isochronous contract are allowed to make immediate progress and start being serviced by the PCI Express fabric. A non-compliant Requester that tries to inject more isochronous transactions than what was being allowed by the contract is prevented from doing so by the flow-control mechanism thereby allowing compliant Requesters to correctly operate independent of non-compliant Requesters.

In the Endpoint-to-Root-Complex model, since the aggregated isochronous traffic is eventually limited by the host memory subsystem's bandwidth capabilities, isochronous read requests, and write requests (and Messages) are budgeted together. A Requester may divide the isochronous bandwidth between read requests and write requests as appropriate.

A.2.1 Isochronous Time Period and Isochronous Virtual Timeslot

The PCI Express isochronous time period (*T*) is uniformly divided into units of virtual timeslots (*t*). To provide precise isochronous bandwidth distribution only one isochronous request packet is allowed per virtual timeslot. The virtual timeslot supported by a PCI Express component is reported through the Reference Clock field in the Virtual Channel Capability structure or the Multi-Function Virtual Channel Capability structure. When Reference Clock = 00b, duration of a virtual timeslot *t* is 100 ns. Duration of isochronous time period *T* depends on the number of phases of the supported time-based WRR Port arbitration table size. When the time-based WRR Port Arbitration Table size equals to 128, there are 128 virtual timeslots (*t*) in an isochronous time period, i.e. T = 12.8 µs.

Note that isochronous period *T* as well as virtual timeslots *t* do not need to be aligned and synchronized among different PCI Express isochronous devices, i.e., the notion of {*T, t*} is local to each individual isochronous device.

A.2.2 Isochronous Payload Size

The payload size (*Y*) for isochronous transactions must not exceed Max_Payload_Size (see Section 7.8.4). After configuration, the Max_Payload_Size is set and fixed for each path that supports isochronous service with a value required to meet isochronous latency. The fixed Max_Payload_Size value is used for isochronous bandwidth budgeting regardless of the actual size of data payload associated with isochronous transactions. For isochronous bandwidth budgeting, we have

Y = Max_Payload_Size

Equation A-2 Isochronous Payload Size

A transaction with partial writes is treated as a normally accounted transaction. A Completer must account for partial writes as part of bandwidth assignment (for worst case servicing time).

A.2.3 Isochronous Bandwidth Allocation

Given T, t and Y, the maximum virtual timeslots within a time period is

$$N_{max} = \frac{T}{t}$$

Equation A-3 N_{max}

and the maximum specifiable isochronous bandwidth is

$$BW_{max} = \frac{Y}{t}$$

Equation A-4 BW_{max}

The granularity with which isochronous bandwidth can be allocated is defined as:

$$BW_{granularity} = \frac{Y}{T}$$

Equation A-5 $BW_{granularity}$

Given T and t at 12.8 µs and 100 ns, respectively, N_{max} is 128. As shown in Table A-1 , BW_{max} and $BW_{granularity}$ are functions of the isochronous payload size Y.

Table A-1 Isochronous Bandwidth Ranges and Granularities

Y (bytes)	128	256	512	1024
BW_{max} (MB/s)	1289	2560	5120	10240
$BW_{granularity}$ (MB/s)	10	20	40	80

Similar to bandwidth budgeting, isochronous service disciplines including arbitration schemes are based on counting requests (not the sizes of those requests). Therefore, assigning isochronous bandwidth BW_{link} to a PCI Express Link is equivalent to assigning N_{link} virtual timeslots per isochronous period, where N_{link} is given by

$$N_{link} = \frac{BW_{link}}{BW_{granularity}}$$

Equation A-6 N_{link}

A Switch Port serving as an Egress Port (or an RCRB serving as a "virtual" Egress Port) for an isochronous traffic, the N_{max} virtual timeslots within T are represented by the time-based WRR Port Arbitration Table in the PCI Express Virtual Channel Capability structure detailed in Section 7.9.1 . The table consists of N_{max} entries. An entry in the table represents one virtual timeslot in the isochronous time period. When a table entry is given a value of PN, it means that the timeslot is assigned to an Ingress Port (in respect to the isochronous traffic targeting the Egress Port) designated by a Port Number of PN. Therefore, N_{link} virtual timeslots are assigned to the Ingress Port when there are N_{link} entries in the table with value of PN. The Egress Port may admit one isochronous request transaction from the Ingress Port for further service only when the table entry reached by the Egress Port's isochronous time ticker (that increments by 1 every t time and wraps around when reaching T) is set to PN. Even if there are outstanding isochronous requests ready in the Ingress Port, they will not be served until next round of time-based WRR arbitration. In this manner, the time-based Port Arbitration Table serves for both isochronous bandwidth assignment and isochronous traffic regulation.

For an Endpoint serving as a Requester or a Completer, isochronous bandwidth allocation is accomplished through negotiation between system software and device driver, which is outside of the scope of this specification.

A.2.4 Isochronous Transaction Latency

Transaction latency is composed of the latency through the PCI Express fabric and the latency contributed by the Completer. Isochronous transaction latency is defined for each transaction and measured in units of virtual timeslot t.

- The *read latency* is defined as the round-trip latency. This is the delay from the time when the device submits a memory read request packet to its Transaction Layer (Transmit side) to the time when the corresponding read completion arrives at the device's Transaction Layer (Receive side).
- The *write latency* is defined as the delay from the time when the Requester posts a memory write request to its PCI Express Transaction Layer (Transmit side) to the time when the data write becomes globally visible within the memory subsystem of the Completer. A write to memory reaches the point of global visibility when all agents accessing that memory address get the updated data.

When the upper bound and the lower bound of isochronous transaction latency are provided, the size of isochronous data buffers in a Requester can be determined. For most of common platforms, the minimum isochronous transaction latency is much smaller than the maximum. As a conservative measure, the minimum isochronous transaction latency is assumed to be zero; only guidelines on measuring the maximum isochronous transaction latency are provided here.

For a Requester, the maximum isochronous (read or write) transaction latency (L) can be accounted as the following:

$$L = L_{Fabric} + L_{Completer}$$

Equation A-7 Max Isochronous Transaction Latency

where L_{Fabric} is the maximum latency of the PCI Express fabric and $L_{Completer}$ is the maximum latency of the Completer.

L_{Fabric} which applies to both read and write transactions, depends on the topology, latency across each PCI Express Link, and the arbitration point in the path between the Requester to the Completer. The latency on a PCI Express Link

depends on pipeline delays, width and operational frequency of the Link, transmission of electrical signals across the medium, wake up latency from low power states, and delays caused by Data Link Layer Retry.

A restriction on the PCI Express topology may be imposed for each targeted platform in order to provide a practically meaningful guideline for L_{Fabric}. The values of L_{Fabric} should be reasonable and serve as practical upper limits under normal operating conditions.

The value of $L_{Completer}$ depends on the memory technology, memory configuration, and the arbitration policies in the Completer that comprehend PCI Express isochronous traffic. The target value for $L_{Completer}$ should provide enough headroom to allow for implementation tradeoffs.

Definitions of read and write transaction latencies for a Completer are different:

- Read transaction latency for the Completer is defined as the delay from the time a memory read transaction is available at the Receiver end of a PCI Express Port in the Completer to the time the corresponding read completion transaction is posted to the transmission end of the PCI Express Port.
- Write transaction latency is defined as the delay from the time a memory write transaction is available at the Receiver end of a PCI Express Port in the Completer to the time that the transmitted data is globally visible.

All of the isochronous transaction latencies defined above are based on the assumption that the Requester injects isochronous transactions uniformly. According to an isochronous contract of {*N, T, t*}, the uniform traffic injection is defined such that up to *N* transactions are evenly distributed over the isochronous period *T* based on a ticker granularity of virtual timeslot *t*. For a Requester with non-uniform isochronous transaction injection, the Requester is responsible of accounting for any additional delay due to the deviation of its injection pattern from a uniform injection pattern.

A.2.5 An Example Illustrating Isochronous Parameters

Figure A-3 illustrates the key isochronous parameters using a simplified example with $T = 20t$ and $L = 22t$. A Requester has reserved isochronous bandwidth of four transactions per *T*. The device shares the allocated isochronous bandwidths for both read requests and write requests. As shown, during one isochronous time period, the Requester issues two read requests and two write requests. All requests are completed within the designated transaction latency *L*. Also shown in the figure, there is no time dependency between the service time of write requests and the arrival time of read completions.

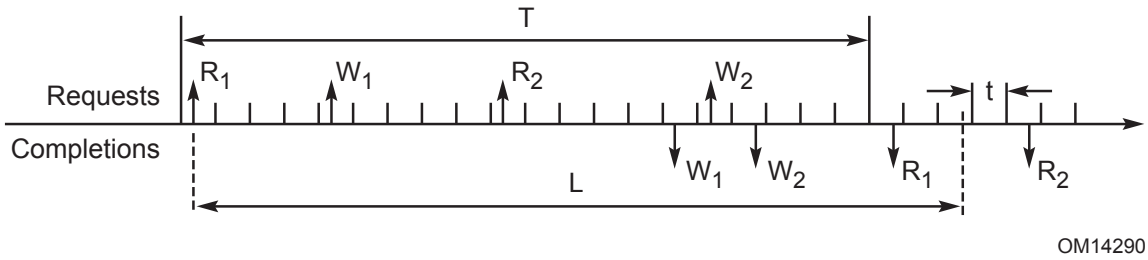


Figure A-3 A Simplified Example Illustrating PCI Express Isochronous Parameters

A.3 Isochronous Transaction Rules

Isochronous transactions follow the same rules as described in Chapter 2 . In order to assist the Completer to meet latency requirements, the following additional rules further illustrate and clarify the proper behavior of isochronous transactions:

- The value in the Length field of requests must never exceed Max_Payload_Size.

A.4 Transaction Ordering

In general, isochronous transactions follow the ordering rules described in Section 2.4 . The following ordering rule further illustrates and clarifies the proper behavior of isochronous transactions:

- There are no ordering guarantees between any isochronous and non-isochronous transactions because the traffic has been segregated into distinct VC resources.
- Isochronous write requests are serviced on any PCI Express Link in strictly the same order as isochronous write requests are posted.
- Switches must allow isochronous posted requests to pass isochronous read completions.

A.5 Isochronous Data Coherency

Cache coherency for isochronous transactions is an operating system software and Root Complex hardware issue. PCI Express provides the necessary mechanism to control Root Complex behavior in terms of enforcing hardware cache coherency on a per transaction basis.

For platforms where snoop latency in a Root Complex is either unbounded or can be excessively large, in order to meet tight maximum isochronous transaction latency $L_{Completer}$, or more precisely $L_{Root\_Complex}$, all isochronous transactions should have the No Snoop Attribute bit set.

A Root Complex must report the Root Complex's capability to the system software by setting the Reject Snoop Transactions field in the VC Resource Capability register (for any VC resource capable of supporting isochronous traffic) in its RCRB. Based on whether or not a Root Complex is capable of providing hardware enforced cache coherency for isochronous traffic while still meeting isochronous latency target, system software can then inform the device driver of Endpoints to set or unset the No Snoop Attribute bit for isochronous transactions.

Note that cache coherency considerations for isochronous traffic do not apply to peer-to-peer communication.

A.6 Flow Control

Completers and PCI Express fabric components should implement proper sizing of buffers such that under normal operating conditions, no backpressure due to flow control should be applied to isochronous traffic injected uniformly by a Requester. For Requesters that are compliant to the isochronous contract, but have bursty injection behavior, Switches and Completers may apply flow control backpressure as long as the admitted isochronous traffic is uniform and compliant to the isochronous contract. Under abnormal conditions when isochronous traffic jitter becomes significant or when isochronous traffic is oversubscribed due to excessive Data Link Layer Retry, flow control provides a natural mechanism to ensure functional correctness.

A.7 Considerations for Bandwidth Allocation

A.7.1 Isochronous Bandwidth of PCI Express Links

Isochronous bandwidth budgeting for PCI Express Links can be derived based on Link parameters such as isochronous payload size and the speed and width of the Link.

Isochronous bandwidth allocation for a PCI Express Link should be limited to certain percentage of the maximum effective Link bandwidth in order to leave sufficient bandwidth for non-isochronous traffic and to account for temporary Link bandwidth reduction due to retries. Link utilization is counted based on the actual cycles consumed on the physical PCI Express Link. The maximum number of virtual slots allowed per Link (N_{link}) depends on the isochronous packet payload size and the speed and width of the Link.

As isochronous bandwidth allocation on a PCI Express Link is based on number of requests N_{link} per isochronous period. There is no distinction between read requests and write requests in budgeting isochronous bandwidth on a PCI Express Link.

A.7.2 Isochronous Bandwidth of Endpoints

For peer-to-peer communication, the device driver is responsible for reporting to system software if the device is capable of being a Completer for isochronous transactions. In addition, the driver must report the device's isochronous bandwidth capability. The specifics of the report mechanism are outside the scope of this specification.

A.7.3 Isochronous Bandwidth of Switches

Allocation of isochronous bandwidth for a Switch must consider the capacity and utilization of PCI Express Links associated with the Ingress Port and the Egress Port of the Switch that connect the Requester and the Completer, respectively. The lowest common denominator of the two determines if a requested isochronous bandwidth can be supported.

A.7.4 Isochronous Bandwidth of Root Complex

Isochronous bandwidth of Root Complex is reported to the software through its RCRB structure. Specifically, the Maximum Time Slots field of the VC Resource Capability register in VC Capability structure indicates the total isochronous bandwidth shared by the Root Ports associated with the RCRB. Details of the platform budgeting for available isochronous bandwidth within a Root Complex are outside of the scope of this specification.

A.8 Considerations for PCI Express Components

A.8.1 An Endpoint as a Requester

Before an Endpoint as a Requester can start issuing isochronous request transactions, the following configuration steps must be performed by software:

- Configuration of at least one VC resource capable of supporting isochronous communication and assignment of at least one TC label.
- Enablement of this VC resource.

When the Requester uniformly injects isochronous requests, the Receive Port, either a Switch Port or a Root Port, should issue Flow Control credits back promptly such that no backpressure should be applied to the associated VC. This type of Requester may size its buffer based on the PCI Express fabric latency L_{Fabric} plus the Completer's latency $L_{Completer}$.

When isochronous transactions are injected non-uniformly, either some transactions experience longer PCI Express fabric delay or the Requester gets back-pressured on the associated VC. This type of Requester must size its buffer to account for the deviation of its injection pattern from uniformity.

A.8.2 An Endpoint as a Completer

An Endpoint may serve as a Completer for isochronous peer-to-peer communication. Before an Endpoint starts serving isochronous transactions, system software must identify/configure a VC resource capable of supporting isochronous traffic and assigned a corresponding TC label.

An Endpoint Completer must observe the maximum isochronous transaction latency ($L_{Completer}$). An Endpoint Completer does not have to regulate isochronous request traffic if attached to a Switch since Switches implement traffic regulation. However, an Endpoint Completer must size its internal buffer such that no backpressure should be applied to the corresponding VC.

A.8.3 Switches

A Switch may have multiple ports capable of supporting isochronous transactions. Before a Switch starts serving isochronous transactions for a Port, the software must perform the following configuration steps:

- Configuration/enablement of at least one VC resource capable of supporting isochronous communication.
- Configuration of the Port as an Ingress Port:
  - Configuration (or reconfiguration if the associated VC of the Egress Port is already enabled) of the time-based WRR Port Arbitration Table of the targeting Egress Port to include N_{link} entries set to the Ingress Port's Port Number. Here N_{link} is the isochronous allocation for the Ingress Port.
  - Enabling the targeting Egress Port to load newly programmed Port Arbitration Table.
- Configuration of the Port as an Egress Port:
  - Configuration of each VC's Port Arbitration Table with number of entries set according to the assigned isochronous bandwidth for all Ingress Ports.
  - Select proper VC Arbitration, e.g., as strict-priority based VC Arbitration.
  - If required, configuration of the Port's VC Arbitration Table with large weights assigned accordingly to each associated VC.

Each VC associated with isochronous traffic may be served as the highest priority in arbitrating for the shared PCI Express Link resource at an Egress Port. This is comprehended by a Switch's internal arbitration scheme.

In addition, a Switch Port may use “just in time” scheduling mechanism to reduce VC arbitration latency. Instead of pipelining non-isochronous Transport Layer packets to the Data Link Layer of the Egress Port in a manner that Data Link Layer transmit buffer becomes saturated, the Switch Port may hold off scheduling of a new non-isochronous packet to the Data Link Layer as long as it is possible without incurring unnecessary Link idle time.

When a VC configured to support isochronous traffic is enabled for a Switch Port (ingress) that is connected to a Requester, the Switch must enforce proper traffic regulation to ensure that isochronous traffic from the Port conforms to this specification. With such enforcement, normal isochronous transactions from compliant Requesters will not be impacted by ill behavior of any non-compliant Requester.

The above isochronous traffic regulation mechanism only applies to request transactions but not to completion transactions. When Endpoint-to-Root-Complex and peer-to-peer communications co-exist in a Switch, an Egress Port may mix isochronous write requests and read completions in the same direction. In the case of contention, the Egress Port must allow write requests to pass read completions to ensure the Switch meets latency requirement for isochronous requests.

A.8.4 Root Complex

A Root Complex may have multiple Root Ports capable of supporting isochronous transactions. Before a Root Complex starts serving isochronous transactions for a Root Port, the Port must be configured by software to enable VC to support isochronous traffic using the following configuration steps:

- Configuration of at least one VC resource capable of supporting isochronous communication and assignment of at least one TC label.
- Configuration of the Root Port as an Ingress Port:
  - Configuration (or reconfiguration if the associated VC in RCRB is already enabled) of the time-based WRR Port Arbitration Table of the targeting RCRB to include N_{link} entries set to the Ingress Port's Port Number. Here N_{link} is the isochronous allocation for the Port.
  - Enabling the targeting RCRB to load newly programmed Port Arbitration Table.
- Configuration of the Root Port as an Egress Port:
  - If supported, configuration of the Root Port's VC Arbitration Table with large weights assigned to the associated VC.
  - If the Root Complex supports peer-to-peer traffic between Root Ports, configuration of the Root Port's Port Arbitration Table number of entries is set according to the assigned isochronous bandwidth for all Ingress Ports.

A Root Complex must observe the maximum isochronous transaction latency ($L_{Completer}$ or more precisely $L_{Root\_Complex}$) that applies to all the Root Ports in the Root Complex. How a Root Complex schedules memory cycles for PCI Express isochronous transactions and other memory transactions is outside of the scope of this specification as long as $L_{Root\_Complex}$ is met for PCI Express isochronous transactions.

When a VC is enabled to support isochronous traffic for a Root Port, the Root Complex must enforce proper traffic regulation to ensure that isochronous traffic from the Root Port conforms to this specification. With such enforcement, normal isochronous transactions from compliant Requesters will not be impacted by ill behavior of any non-compliant Requesters. Isochronous traffic regulation is implemented using the time-based Port Arbitration Table in RCRB.

Root Complex may perform the following operations for invalid isochronous transactions:

- Return partial completions for read requests with the value in the Length field exceeding Max_Payload_Size.

Symbol Encoding

B.

Table B-1 shows the byte-to-Symbol encodings for data characters. Table B-2 shows the Symbol encodings for the Special Symbols used for TLP/DLLP Framing and for interface management. RD- and RD+ refer to the Running Disparity of the Symbol sequence on a per-Lane basis.

Table B-1 8b/10b Data Symbol Codes

Data Byte Name	Data Byte Value (hex)	Bits HGF EDCBA (binary)	Current RD- abcdei fghj (binary)	Current RD+ abcdei fghj (binary)
D0.0	00	000 00000	100111 0100	011000 1011
D1.0	01	000 00001	011101 0100	100010 1011
D2.0	02	000 00010	101101 0100	010010 1011
D3.0	03	000 00011	110001 1011	110001 0100
D4.0	04	000 00100	110101 0100	001010 1011
D5.0	05	000 00101	101001 1011	101001 0100
D6.0	06	000 00110	011001 1011	011001 0100
D7.0	07	000 00111	111000 1011	000111 0100
D8.0	08	000 01000	111001 0100	000110 1011
D9.0	09	000 01001	100101 1011	100101 0100
D10.0	0A	000 01010	010101 1011	010101 0100
D11.0	0B	000 01011	110100 1011	110100 0100
D12.0	0C	000 01100	001101 1011	001101 0100
D13.0	0D	000 01101	101100 1011	101100 0100
D14.0	0E	000 01110	011100 1011	011100 0100
D15.0	0F	000 01111	010111 0100	101000 1011
D16.0	10	000 10000	011011 0100	100100 1011
D17.0	11	000 10001	100011 1011	100011 0100
D18.0	12	000 10010	010011 1011	010011 0100
D19.0	13	000 10011	110010 1011	110010 0100
D20.0	14	000 10100	001011 1011	001011 0100
D21.0	15	000 10101	101010 1011	101010 0100
D22.0	16	000 10110	011010 1011	011010 0100
D23.0	17	000 10111	111010 0100	000101 1011

Data Byte Name	Data Byte Value (hex)	Bits HGF EDCBA (binary)	Current RD- abcdei fghj (binary)	Current RD+ abcdei fghj (binary)
D24.0	18	000 11000	110011 0100	001100 1011
D25.0	19	000 11001	100110 1011	100110 0100
D26.0	1A	000 11010	010110 1011	010110 0100
D27.0	1B	000 11011	110110 0100	001001 1011
D28.0	1C	000 11100	001110 1011	001110 0100
D29.0	1D	000 11101	101110 0100	010001 1011
D30.0	1E	000 11110	011110 0100	100001 1011
D31.0	1F	000 11111	101011 0100	010100 1011
D0.1	20	001 00000	100111 1001	011000 1001
D1.1	21	001 00001	011101 1001	100010 1001
D2.1	22	001 00010	101101 1001	010010 1001
D3.1	23	001 00011	110001 1001	110001 1001
D4.1	24	001 00100	110101 1001	001010 1001
D5.1	25	001 00101	101001 1001	101001 1001
D6.1	26	001 00110	011001 1001	011001 1001
D7.1	27	001 00111	111000 1001	000111 1001
D8.1	28	001 01000	111001 1001	000110 1001
D9.1	29	001 01001	100101 1001	100101 1001
D10.1	2A	001 01010	010101 1001	010101 1001
D11.1	2B	001 01011	110100 1001	110100 1001
D12.1	2C	001 01100	001101 1001	001101 1001
D13.1	2D	001 01101	101100 1001	101100 1001
D14.1	2E	001 01110	011100 1001	011100 1001
D15.1	2F	001 01111	010111 1001	101000 1001
D16.1	30	001 10000	011011 1001	100100 1001
D17.1	31	001 10001	100011 1001	100011 1001
D18.1	32	001 10010	010011 1001	010011 1001
D19.1	33	001 10011	110010 1001	110010 1001
D20.1	34	001 10100	001011 1001	001011 1001

Data Byte Name	Data Byte Value (hex)	Bits HGF EDCBA (binary)	Current RD- abcdei fghj (binary)	Current RD+ abcdei fghj (binary)
D21.1	35	001 10101	101010 1001	101010 1001
D22.1	36	001 10110	011010 1001	011010 1001
D23.1	37	001 10111	111010 1001	000101 1001
D24.1	38	001 11000	110011 1001	001100 1001
D25.1	39	001 11001	100110 1001	100110 1001
D26.1	3A	001 11010	010110 1001	010110 1001
D27.1	3B	001 11011	110110 1001	001001 1001
D28.1	3C	001 11100	001110 1001	001110 1001
D29.1	3D	001 11101	101110 1001	010001 1001
D30.1	3E	001 11110	011110 1001	100001 1001
D31.1	3F	001 11111	101011 1001	010100 1001
D0.2	40	010 00000	100111 0101	011000 0101
D1.2	41	010 00001	011101 0101	100010 0101
D2.2	42	010 00010	101101 0101	010010 0101
D3.2	43	010 00011	110001 0101	110001 0101
D4.2	44	010 00100	110101 0101	001010 0101
D5.2	45	010 00101	101001 0101	101001 0101
D6.2	46	010 00110	011001 0101	011001 0101
D7.2	47	010 00111	111000 0101	000111 0101
D8.2	48	010 01000	111001 0101	000110 0101
D9.2	49	010 01001	100101 0101	100101 0101
D10.2	4A	010 01010	010101 0101	010101 0101
D11.2	4B	010 01011	110100 0101	110100 0101
D12.2	4C	010 01100	001101 0101	001101 0101
D13.2	4D	010 01101	101100 0101	101100 0101
D14.2	4E	010 01110	011100 0101	011100 0101
D15.2	4F	010 01111	010111 0101	101000 0101
D16.2	50	010 10000	011011 0101	100100 0101
D17.2	51	010 10001	100011 0101	100011 0101

Data Byte Name	Data Byte Value (hex)	Bits HGF EDCBA (binary)	Current RD- abcdei fghj (binary)	Current RD+ abcdei fghj (binary)
D18.2	52	010 10010	010011 0101	010011 0101
D19.2	53	010 10011	110010 0101	110010 0101
D20.2	54	010 10100	001011 0101	001011 0101
D21.2	55	010 10101	101010 0101	101010 0101
D22.2	56	010 10110	011010 0101	011010 0101
D23.2	57	010 10111	111010 0101	000101 0101
D24.2	58	010 11000	110011 0101	001100 0101
D25.2	59	010 11001	100110 0101	100110 0101
D26.2	5A	010 11010	010110 0101	010110 0101
D27.2	5B	010 11011	110110 0101	001001 0101
D28.2	5C	010 11100	001110 0101	001110 0101
D29.2	5D	010 11101	101110 0101	010001 0101
D30.2	5E	010 11110	011110 0101	100001 0101
D31.2	5F	010 11111	101011 0101	010100 0101
D0.3	60	011 00000	100111 0011	011000 1100
D1.3	61	011 00001	011101 0011	100010 1100
D2.3	62	011 00010	101101 0011	010010 1100
D3.3	63	011 00011	110001 1100	110001 0011
D4.3	64	011 00100	110101 0011	001010 1100
D5.3	65	011 00101	101001 1100	101001 0011
D6.3	66	011 00110	011001 1100	011001 0011
D7.3	67	011 00111	111000 1100	000111 0011
D8.3	68	011 01000	111001 0011	000110 1100
D9.3	69	011 01001	100101 1100	100101 0011
D10.3	6A	011 01010	010101 1100	010101 0011
D11.3	6B	011 01011	110100 1100	110100 0011
D12.3	6C	011 01100	001101 1100	001101 0011
D13.3	6D	011 01101	101100 1100	101100 0011
D14.3	6E	011 01110	011100 1100	011100 0011

Data Byte Name	Data Byte Value (hex)	Bits HGF EDCBA (binary)	Current RD- abcdei fghj (binary)	Current RD+ abcdei fghj (binary)
D15.3	6F	011 01111	010111 0011	101000 1100
D16.3	70	011 10000	011011 0011	100100 1100
D17.3	71	011 10001	100011 1100	100011 0011
D18.3	72	011 10010	010011 1100	010011 0011
D19.3	73	011 10011	110010 1100	110010 0011
D20.3	74	011 10100	001011 1100	001011 0011
D21.3	75	011 10101	101010 1100	101010 0011
D22.3	76	011 10110	011010 1100	011010 0011
D23.3	77	011 10111	111010 0011	000101 1100
D24.3	78	011 11000	110011 0011	001100 1100
D25.3	79	011 11001	100110 1100	100110 0011
D26.3	7A	011 11010	010110 1100	010110 0011
D27.3	7B	011 11011	110110 0011	001001 1100
D28.3	7C	011 11100	001110 1100	001110 0011
D29.3	7D	011 11101	101110 0011	010001 1100
D30.3	7E	011 11110	011110 0011	100001 1100
D31.3	7F	011 11111	101011 0011	010100 1100
D0.4	80	100 00000	100111 0010	011000 1101
D1.4	81	100 00001	011101 0010	100010 1101
D2.4	82	100 00010	101101 0010	010010 1101
D3.4	83	100 00011	110001 1101	110001 0010
D4.4	84	100 00100	110101 0010	001010 1101
D5.4	85	100 00101	101001 1101	101001 0010
D6.4	86	100 00110	011001 1101	011001 0010
D7.4	87	100 00111	111000 1101	000111 0010
D8.4	88	100 01000	111001 0010	000110 1101
D9.4	89	100 01001	100101 1101	100101 0010
D10.4	8A	100 01010	010101 1101	010101 0010
D11.4	8B	100 01011	110100 1101	110100 0010

Data Byte Name	Data Byte Value (hex)	Bits HGF EDCBA (binary)	Current RD- abcdei fghj (binary)	Current RD+ abcdei fghj (binary)
D12.4	8C	100 01100	001101 1101	001101 0010
D13.4	8D	100 01101	101100 1101	101100 0010
D14.4	8E	100 01110	011100 1101	011100 0010
D15.4	8F	100 01111	010111 0010	101000 1101
D16.4	90	100 10000	011011 0010	100100 1101
D17.4	91	100 10001	100011 1101	100011 0010
D18.4	92	100 10010	010011 1101	010011 0010
D19.4	93	100 10011	110010 1101	110010 0010
D20.4	94	100 10100	001011 1101	001011 0010
D21.4	95	100 10101	101010 1101	101010 0010
D22.4	96	100 10110	011010 1101	011010 0010
D23.4	97	100 10111	111010 0010	000101 1101
D24.4	98	100 11000	110011 0010	001100 1101
D25.4	99	100 11001	100110 1101	100110 0010
D26.4	9A	100 11010	010110 1101	010110 0010
D27.4	9B	100 11011	110110 0010	001001 1101
D28.4	9C	100 11100	001110 1101	001110 0010
D29.4	9D	100 11101	101110 0010	010001 1101
D30.4	9E	100 11110	011110 0010	100001 1101
D31.4	9F	100 11111	101011 0010	010100 1101
D0.5	A0	101 00000	100111 1010	011000 1010
D1.5	A1	101 00001	011101 1010	100010 1010
D2.5	A2	101 00010	101101 1010	010010 1010
D3.5	A3	101 00011	110001 1010	110001 1010
D4.5	A4	101 00100	110101 1010	001010 1010
D5.5	A5	101 00101	101001 1010	101001 1010
D6.5	A6	101 00110	011001 1010	011001 1010
D7.5	A7	101 00111	111000 1010	000111 1010
D8.5	A8	101 01000	111001 1010	000110 1010

Data Byte Name	Data Byte Value (hex)	Bits HGF EDCBA (binary)	Current RD- abcdei fghj (binary)	Current RD+ abcdei fghj (binary)
D9.5	A9	101 01001	100101 1010	100101 1010
D10.5	AA	101 01010	010101 1010	010101 1010
D11.5	AB	101 01011	110100 1010	110100 1010
D12.5	AC	101 01100	001101 1010	001101 1010
D13.5	AD	101 01101	101100 1010	101100 1010
D14.5	AE	101 01110	011100 1010	011100 1010
D15.5	AF	101 01111	010111 1010	101000 1010
D16.5	B0	101 10000	011011 1010	100100 1010
D17.5	B1	101 10001	100011 1010	100011 1010
D18.5	B2	101 10010	010011 1010	010011 1010
D19.5	B3	101 10011	110010 1010	110010 1010
D20.5	B4	101 10100	001011 1010	001011 1010
D21.5	B5	101 10101	101010 1010	101010 1010
D22.5	B6	101 10110	011010 1010	011010 1010
D23.5	B7	101 10111	111010 1010	000101 1010
D24.5	B8	101 11000	110011 1010	001100 1010
D25.5	B9	101 11001	100110 1010	100110 1010
D26.5	BA	101 11010	010110 1010	010110 1010
D27.5	BB	101 11011	110110 1010	001001 1010
D28.5	BC	101 11100	001110 1010	001110 1010
D29.5	BD	101 11101	101110 1010	010001 1010
D30.5	BE	101 11110	011110 1010	100001 1010
D31.5	BF	101 11111	101011 1010	010100 1010
D0.6	C0	110 00000	100111 0110	011000 0110
D1.6	C1	110 00001	011101 0110	100010 0110
D2.6	C2	110 00010	101101 0110	010010 0110
D3.6	C3	110 00011	110001 0110	110001 0110
D4.6	C4	110 00100	110101 0110	001010 0110
D5.6	C5	110 00101	101001 0110	101001 0110

Data Byte Name	Data Byte Value (hex)	Bits HGF EDCBA (binary)	Current RD- abcdei fghj (binary)	Current RD+ abcdei fghj (binary)
D6.6	C6	110 00110	011001 0110	011001 0110
D7.6	C7	110 00111	111000 0110	000111 0110
D8.6	C8	110 01000	111001 0110	000110 0110
D9.6	C9	110 01001	100101 0110	100101 0110
D10.6	CA	110 01010	010101 0110	010101 0110
D11.6	CB	110 01011	110100 0110	110100 0110
D12.6	CC	110 01100	001101 0110	001101 0110
D13.6	CD	110 01101	101100 0110	101100 0110
D14.6	CE	110 01110	011100 0110	011100 0110
D15.6	CF	110 01111	010111 0110	101000 0110
D16.6	D0	110 10000	011011 0110	100100 0110
D17.6	D1	110 10001	100011 0110	100011 0110
D18.6	D2	110 10010	010011 0110	010011 0110
D19.6	D3	110 10011	110010 0110	110010 0110
D20.6	D4	110 10100	001011 0110	001011 0110
D21.6	D5	110 10101	101010 0110	101010 0110
D22.6	D6	110 10110	011010 0110	011010 0110
D23.6	D7	110 10111	111010 0110	000101 0110
D24.6	D8	110 11000	110011 0110	001100 0110
D25.6	D9	110 11001	100110 0110	100110 0110
D26.6	DA	110 11010	010110 0110	010110 0110
D27.6	DB	110 11011	110110 0110	001001 0110
D28.6	DC	110 11100	001110 0110	001110 0110
D29.6	DD	110 11101	101110 0110	010001 0110
D30.6	DE	110 11110	011110 0110	100001 0110
D31.6	DF	110 11111	101011 0110	010100 0110
D0.7	E0	111 00000	100111 0001	011000 1110
D1.7	E1	111 00001	011101 0001	100010 1110
D2.7	E2	111 00010	101101 0001	010010 1110

Data Byte Name	Data Byte Value (hex)	Bits HGF EDCBA (binary)	Current RD- abcdei fghj (binary)	Current RD+ abcdei fghj (binary)
D3.7	E3	111 00011	110001 1110	110001 0001
D4.7	E4	111 00100	110101 0001	001010 1110
D5.7	E5	111 00101	101001 1110	101001 0001
D6.7	E6	111 00110	011001 1110	011001 0001
D7.7	E7	111 00111	111000 1110	000111 0001
D8.7	E8	111 01000	111001 0001	000110 1110
D9.7	E9	111 01001	100101 1110	100101 0001
D10.7	EA	111 01010	010101 1110	010101 0001
D11.7	EB	111 01011	110100 1110	110100 1000
D12.7	EC	111 01100	001101 1110	001101 0001
D13.7	ED	111 01101	101100 1110	101100 1000
D14.7	EE	111 01110	011100 1110	011100 1000
D15.7	EF	111 01111	010111 0001	101000 1110
D16.7	F0	111 10000	011011 0001	100100 1110
D17.7	F1	111 10001	100011 0111	100011 0001
D18.7	F2	111 10010	010011 0111	010011 0001
D19.7	F3	111 10011	110010 1110	110010 0001
D20.7	F4	111 10100	001011 0111	001011 0001
D21.7	F5	111 10101	101010 1110	101010 0001
D22.7	F6	111 10110	011010 1110	011010 0001
D23.7	F7	111 10111	111010 0001	000101 1110
D24.7	F8	111 11000	110011 0001	001100 1110
D25.7	F9	111 11001	100110 1110	100110 0001
D26.7	FA	111 11010	010110 1110	010110 0001
D27.7	FB	111 11011	110110 0001	001001 1110
D28.7	FC	111 11100	001110 1110	001110 0001
D29.7	FD	111 11101	101110 0001	010001 1110
D30.7	FE	111 11110	011110 0001	100001 1110
D31.7	FF	111 11111	101011 0001	010100 1110

Table B-2 8b/10b Special Character Symbol Codes

Data Byte Name	Data Byte Value (hex)	Bits HGF EDCBA (binary)	Current RD- abcdei fghj (binary)	Current RD+ abcdei fghj (binary)
K28.0	1C	000 11100	001111 0100	110000 1011
K28.1	3C	001 11100	001111 1001	110000 0110
K28.2	5C	010 11100	001111 0101	110000 1010
K28.3	7C	011 11100	001111 0011	110000 1100
K28.4	9C	100 11100	001111 0010	110000 1101
K28.5	BC	101 11100	001111 1010	110000 0101
K28.6	DC	110 11100	001111 0110	110000 1001
K28.7	FC	111 11100	001111 1000	110000 0111
K23.7	F7	111 10111	111010 1000	000101 0111
K27.7	FB	111 11011	110110 1000	001001 0111
K29.7	FD	111 11101	101110 1000	010001 0111
K30.7	FE	111 11110	011110 1000	100001 0111

Physical Layer Appendix

C.

C.1 8b/10b Data Scrambling Example

The following subroutines encode and decode an 8-bit value contained in “inbyte” with the LFSR. This is presented as one example only; there are many ways to obtain the proper output. This example demonstrates how to advance the LFSR eight times in one operation and how to XOR the data in one operation. Many other implementations are possible but they must all produce the same output as that shown here.

The following algorithm uses the “C” programming language conventions, where “<<” and “>>” represent the shift left and shift right operators, “>” is the compare greater than operator, and “^” is the exclusive or operator, and “&” is the logical “AND” operator.

```
/*
   this routine implements the serial descrambling algorithm in parallel form
   for the LSFR polynomial: x^16+x^5+x^4+x^3+1
   this advances the LSFR 8 bits every time it is called
   this requires fewer than 25 xor gates to implement (with a static register)
   The XOR required to advance 8 bits/clock is:
bit 0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
    8   9  10  11  12  13  14  15   0   1   2   3   4   5   6   7
        8   9  10  11  12  13  14  15
            8   9  10  11  12  13  14  15
                8   9  10  11  12  13  14  15
   The serial data is just the reverse of the upper byte:
   bit 0   1   2   3   4   5   6   7
      15  14  13  12  11  10   9   8
*/
```

```
int scramble_byte(int inbyte)
{
    static int scrambit[16];
    static int bit[16];
    static int bit_out[16];
    static unsigned short lfsr = 0xffff; // 16 bit short for polynomial
    int i, outbyte;

    if (inbyte == COMMA)        // if this is a comma
    {
        lfsr = 0xffff;          // reset the LFSR
        return (COMMA);         // and return the same data
    }

    if (inbyte == SKIP)         // don't advance or encode on skip
        return (SKIP);

    for (i=0; i<16;i++)         // convert LFSR to bit array for legibility
        bit[i] = (lfsr >> i) & 1;

        for (i=0; i<8; i++)     // convert byte to be scrambled for legibility
        scrambit[i] = (inbyte >> i) & 1;

        // apply the xor to the data
    if (! (inbyte & 0x100) &&           // if not a KCODE, scramble the data
        ! (TrainingSequence == TRUE))   // and if not in the middle of
    {                                   // a training sequence
        scrambit[0] ^= bit[15];
        scrambit[1] ^= bit[14];
        scrambit[2] ^= bit[13];
        scrambit[3] ^= bit[12];
        scrambit[4] ^= bit[11];
        scrambit[5] ^= bit[10];
        scrambit[6] ^= bit[9];
        scrambit[7] ^= bit[8];
    }

    // Now advance the LFSR 8 serial clocks
    bit_out[ 0] = bit[ 8];
    bit_out[ 1] = bit[ 9];
    bit_out[ 2] = bit[10];
    bit_out[ 3] = bit[11] ^ bit[ 8];
    bit_out[ 4] = bit[12] ^ bit[ 9] ^ bit[ 8];
    bit_out[ 5] = bit[13] ^ bit[10] ^ bit[ 9] ^ bit[ 8];
    bit_out[ 6] = bit[14] ^ bit[11] ^ bit[10] ^ bit[ 9];
    bit_out[ 7] = bit[15] ^ bit[12] ^ bit[11] ^ bit[10];
    bit_out[ 8] = bit[ 0] ^ bit[13] ^ bit[12] ^ bit[11];
    bit_out[ 9] = bit[ 1] ^ bit[14] ^ bit[13] ^ bit[12];
    bit_out[10] = bit[ 2] ^ bit[15] ^ bit[14] ^ bit[13];
    bit_out[11] = bit[ 3] ^ bit[15] ^ bit[14];
    bit_out[12] = bit[ 4] ^ bit[15];
    bit_out[13] = bit[ 5];
    bit_out[14] = bit[ 6];
    bit_out[15] = bit[ 7];

    lfsr = 0;
    for (i=0; i <16; i++)    // convert the LFSR back to an integer
        lfsr += (bit_out[i] << i);
```

```
    outbyte = 0;
    for (i= 0; i<8; i++)     // convert data back to an integer
        outbyte += (scrambit[i] << i);

    return outbyte;
}
```

```
/*  NOTE THAT THE DESCRAMBLE ROUTINE IS IDENTICAL TO THE SCRAMBLE ROUTINE
    this routine implements the serial descrambling algorithm in parallel form
    this advances the lfsr 8 bits every time it is called
    this uses fewer than 25 xor gates to implement (with a static register)
    The XOR tree is the same as the scrambling routine
*/

int unscramble_byte(int inbyte)
{
    static int descrambit[8];
    static int bit[16];
    static int bit_out[16];
    static unsigned short lfsr = 0xffff; // 16 bit short for polynomial
    int outbyte, i;

    if (inbyte == COMMA)    // if this is a comma
    {
        lfsr = 0xffff;      // reset the LFSR
        return (COMMA);     // and return the same data
    }
    if (inbyte == SKIP)     // don't advance or encode on skip
        return (SKIP);

    for (i=0; i<16;i++)    // convert the LFSR to bit array for legibility
        bit[i] = (lfsr >> i) & 1;

    for (i=0; i<8; i++)    // convert byte to be de-scrambled for legibility
        descrambit[i] = (inbyte >> i) & 1;

    // apply the xor to the data
    if (! (inbyte & 0x100) &&         // if not a KCODE, scramble the data
        ! (TrainingSequence == TRUE)) // and if not in the middle of
    {                                 // a training sequence
        descrambit[0] ^= bit[15];
        descrambit[1] ^= bit[14];
        descrambit[2] ^= bit[13];
        descrambit[3] ^= bit[12];
        descrambit[4] ^= bit[11];
        descrambit[5] ^= bit[10];
        descrambit[6] ^= bit[9];
        descrambit[7] ^= bit[8];
    }

    // Now advance the LFSR 8 serial clocks
    bit_out[ 0] = bit[ 8];
    bit_out[ 1] = bit[ 9];
    bit_out[ 2] = bit[10];
    bit_out[ 3] = bit[11] ^ bit[ 8];
    bit_out[ 4] = bit[12] ^ bit[ 9] ^ bit[ 8];
    bit_out[ 5] = bit[13] ^ bit[10] ^ bit[ 9] ^ bit[ 8];
    bit_out[ 6] = bit[14] ^ bit[11] ^ bit[10] ^ bit[ 9];
    bit_out[ 7] = bit[15] ^ bit[12] ^ bit[11] ^ bit[10];
    bit_out[ 8] = bit[ 0] ^ bit[13] ^ bit[12] ^ bit[11];
    bit_out[ 9] = bit[ 1] ^ bit[14] ^ bit[13] ^ bit[12];
    bit_out[10] = bit[ 2] ^ bit[15] ^ bit[14] ^ bit[13];
    bit_out[11] = bit[ 3] ^ bit[15] ^ bit[14];
    bit_out[12] = bit[ 4] ^ bit[15];
    bit_out[13] = bit[ 5];
    bit_out[14] = bit[ 6];
    bit_out[15] = bit[ 7];

    lfsr = 0;
    for (i=0; i <16; i++)      // convert the LFSR back to an integer
        lfsr += (bit_out[i] << i);

    outbyte = 0;

    for (i= 0; i<8; i++)       // convert data back to an integer
        outbyte += (descrambit[i] << i);

    return outbyte;
}
```

The initial 16-bit values of the LFSR for the first 128 LFSR advances following a reset are listed below:

	0, 8	1, 9	2, A	3, B	4, C	5, D	6, E	7, F
00	FFFF	E817	0328	284B	4DE8	E755	404F	4140
08	4E79	761E	1466	6574	7DBD	B6E5	FDA6	B165
10	7D09	02E5	E572	673D	34CF	CB54	4743	4DEF
18	E055	40E0	EE40	54BE	B334	2C7B	7D0C	07E5
20	E5AF	BA3D	248A	8DC4	D995	85A1	BD5D	4425
28	2BA4	A2A3	B8D2	CBF8	EB43	5763	6E7F	773E
30	345F	5B54	5853	5F18	14B7	B474	6CD4	DC4C
38	5C7C	70FC	F6F0	E6E6	F376	603B	3260	64C2
40	CB84	9743	5CBF	B3FC	E47B	6E04	0C3E	3F2C
48	29D7	D1D1	C069	7BC0	CB73	6043	4A60	6FFA
50	F207	1102	01A9	A939	2351	566B	6646	4FF6
58	F927	3081	85B0	AC5D	478C	82EF	F3F2	E43B
60	2E04	027E	7E72	79AE	A501	1A7D	7F2A	2197
68	9019	0610	1096	9590	8FCD	D0E7	F650	46E6
70	E8D6	C228	3AB2	B70A	129F	9CE2	FC3C	2B5C
78	5AA3	AF6A	70C7	CDF0	E3D5	C0AB	B9C0	D9C1

An 8-bit value of 0 repeatedly encoded with the LFSR after reset produces the following consecutive 8-bit values:

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	FF	17	C0	14	B2	E7	02	82	72	6E	28	A6	BE	6D	BF	8D
10	BE	40	A7	E6	2C	D3	E2	B2	07	02	77	2A	CD	34	BE	E0
20	A7	5D	24	B1	9B	A1	BD	22	D4	45	1D	D3	D7	EA	76	EE
30	2C	DA	1A	FA	28	2D	36	3B	3A	0E	6F	67	CF	06	4C	26
40	D3	E9	3A	CD	27	76	30	FC	94	8B	03	DE	D3	06	52	F6
50	4F	88	80	95	C4	6A	66	F2	9F	0C	A1	35	E2	41	CF	27
60	74	40	7E	9E	A5	58	FE	84	09	60	08	A9	F1	0B	6F	62
70	17	43	5C	ED	48	39	3F	D4	5A	F5	0E	B3	C7	03	9D	9B
80	8B	0D	8E	5C	33	98	77	AE	2D	AC	0B	3E	DA	0B	42	7A
90	7C	D1	CF	A8	1C	12	EE	41	C2	3F	38	7A	0D	69	F4	01
A0	DA	31	72	C5	A0	D7	93	0E	DC	AF	A4	55	E7	F0	72	16
B0	68	D5	38	84	DD	00	CD	18	9E	CA	30	59	4C	75	1B	77

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
C0	31	C5	ED	CF	91	64	6E	3D	FE	E8	29	04	CF	6C	FC	C4
D0	0B	5E	DA	62	BA	5B	AB	DF	59	B7	7D	37	5E	E3	1A	C6
E0	88	14	F5	4F	8B	C8	56	CB	D3	10	42	63	04	8A	B4	F7
F0	84	01	A0	01	83	49	67	EE	3E	2A	8B	A4	76	AF	14	D5
100	4F	AC	60	B6	79	D6	62	B7	43	E7	E5	2A	40	2C	6E	7A
110	56	61	63	20	6A	97	4A	38	05	E5	DD	68	0D	78	4C	53
120	8B	D6	86	57	B2	AA	1A	80	18	DC	BA	FC	03	A3	4B	30

At 2.5 GT/s, scrambling produces the power spectrum (in the 10-bit domain) shown in Figure C-1 .

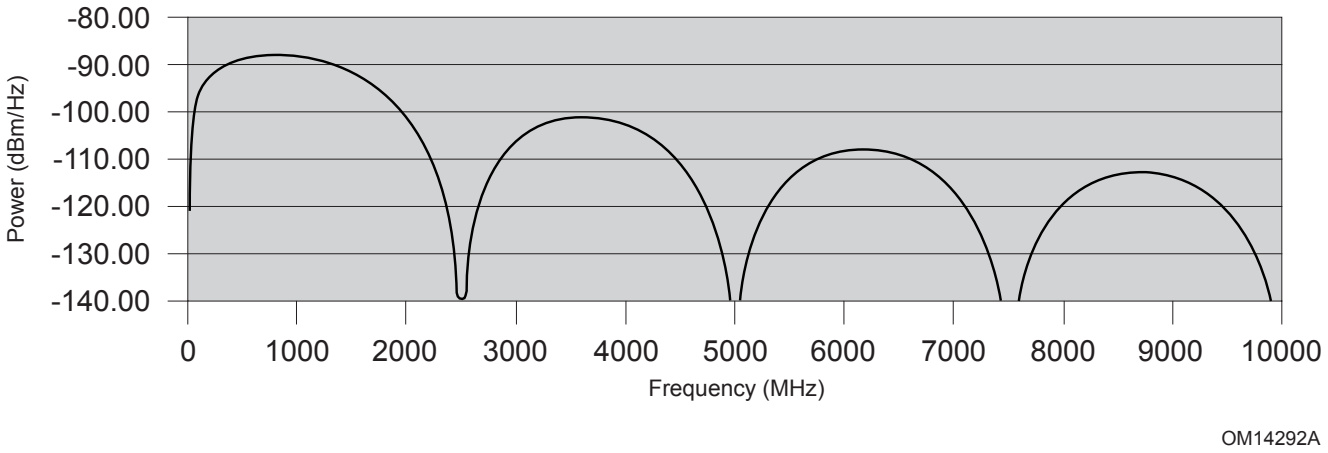


Figure C-1 Scrambling Spectrum at 2.5 GT/s for Data Value of 0

C.2 128b/130b Data Scrambling Example

The following subroutines illustrate how calculate the next value of the 128b/130b scrambling LFSR and how to scramble (or descramble) an 8-bit value, both given the current value of the LFSR.

This is presented as one example only; there are many ways to obtain the proper output. This example demonstrates how to advance the LFSR eight times in one operation and how to XOR the data in one operation. Many other implementations are possible but they must all produce the same output as that shown here.

The following algorithm uses the “C” programming language conventions, where “<<” and “>>” represent the shift left and shift right operators, “^” is the exclusive or operator, and “|=” is the assignment by bitwise or operator.

```
#include <stdio.h>

// "Set Bit" - Sets bit number "bit" of "var" to the value "val". Bit "bit" of "var" must start cleared.
#define SB(var,bit,val) var |= (val & 1) << bit

// "Get Bit" - Returns the value of bit number "bit" of "var".
#define GB(var,bit) ((var >> bit) & 1)

// Function to advance the LFSR value by 8 bits, given the current LFSR value
unsigned long int calc_next_lfsr(unsigned long int lfsr) {
  unsigned long int next_lfsr = 0;
  SB(next_lfsr,22, GB(lfsr,14) ^ GB(lfsr,16) ^ GB(lfsr,18) ^ GB(lfsr,20) ^ GB(lfsr,21) ^ GB(lfsr,22));
  SB(next_lfsr,21, GB(lfsr,13) ^ GB(lfsr,15) ^ GB(lfsr,17) ^ GB(lfsr,19) ^ GB(lfsr,20) ^ GB(lfsr,21));
  SB(next_lfsr,20, GB(lfsr,12) ^ GB(lfsr,19) ^ GB(lfsr,21));
  SB(next_lfsr,19, GB(lfsr,11) ^ GB(lfsr,18) ^ GB(lfsr,20) ^ GB(lfsr,22));
  SB(next_lfsr,18, GB(lfsr,10) ^ GB(lfsr,17) ^ GB(lfsr,19) ^ GB(lfsr,21));
  SB(next_lfsr,17, GB(lfsr, 9) ^ GB(lfsr,16) ^ GB(lfsr,18) ^ GB(lfsr,20) ^ GB(lfsr,22));
  SB(next_lfsr,16, GB(lfsr, 8) ^ GB(lfsr,15) ^ GB(lfsr,17) ^ GB(lfsr,19) ^ GB(lfsr,21) ^ GB(lfsr,22));
  SB(next_lfsr,15, GB(lfsr, 7) ^ GB(lfsr,22));
  SB(next_lfsr,14, GB(lfsr, 6) ^ GB(lfsr,21));
  SB(next_lfsr,13, GB(lfsr, 5) ^ GB(lfsr,20) ^ GB(lfsr,22));
  SB(next_lfsr,12, GB(lfsr, 4) ^ GB(lfsr,19) ^ GB(lfsr,21) ^ GB(lfsr,22));
  SB(next_lfsr,11, GB(lfsr, 3) ^ GB(lfsr,18) ^ GB(lfsr,20) ^ GB(lfsr,21) ^ GB(lfsr,22));
  SB(next_lfsr,10, GB(lfsr, 2) ^ GB(lfsr,17) ^ GB(lfsr,19) ^ GB(lfsr,20) ^ GB(lfsr,21) ^ GB(lfsr,22));
  SB(next_lfsr, 9, GB(lfsr, 1) ^ GB(lfsr,16) ^ GB(lfsr,18) ^ GB(lfsr,19) ^ GB(lfsr,20) ^ GB(lfsr,21));
  SB(next_lfsr, 8, GB(lfsr, 0) ^ GB(lfsr,15) ^ GB(lfsr,17) ^ GB(lfsr,18) ^ GB(lfsr,19) ^ GB(lfsr,20));
  SB(next_lfsr, 7, GB(lfsr,17) ^ GB(lfsr,19) ^ GB(lfsr,20) ^ GB(lfsr,21));
  SB(next_lfsr, 6, GB(lfsr,16) ^ GB(lfsr,18) ^ GB(lfsr,19) ^ GB(lfsr,20) ^ GB(lfsr,22));
  SB(next_lfsr, 5, GB(lfsr,15) ^ GB(lfsr,17) ^ GB(lfsr,18) ^ GB(lfsr,19) ^ GB(lfsr,21) ^ GB(lfsr,22));
  SB(next_lfsr, 4, GB(lfsr,17));
  SB(next_lfsr, 3, GB(lfsr,16));
  SB(next_lfsr, 2, GB(lfsr,15) ^ GB(lfsr,22));
  SB(next_lfsr, 1, GB(lfsr,16) ^ GB(lfsr,18) ^ GB(lfsr,20) ^ GB(lfsr,22));
  SB(next_lfsr, 0, GB(lfsr,15) ^ GB(lfsr,17) ^ GB(lfsr,19) ^ GB(lfsr,21) ^ GB(lfsr,22));
  return(next_lfsr);
}


// Function to scramble a byte, given the current LFSR value
unsigned char scramble_data(unsigned long lfsr, unsigned char data_in) {
  unsigned char data_out = 0;
  SB(data_out, 7, GB(data_in,7) ^ GB(lfsr,15) ^ GB(lfsr,17) ^ GB(lfsr,19) ^ GB(lfsr,21) ^ GB(lfsr,22));
  SB(data_out, 6, GB(data_in,6) ^ GB(lfsr,16) ^ GB(lfsr,18) ^ GB(lfsr,20) ^ GB(lfsr,22));
  SB(data_out, 5, GB(data_in,5) ^ GB(lfsr,17) ^ GB(lfsr,19) ^ GB(lfsr,21));
  SB(data_out, 4, GB(data_in,4) ^ GB(lfsr,18) ^ GB(lfsr,20) ^ GB(lfsr,22));
  SB(data_out, 3, GB(data_in,3) ^ GB(lfsr,19) ^ GB(lfsr,21));
  SB(data_out, 2, GB(data_in,2) ^ GB(lfsr,20) ^ GB(lfsr,22));
  SB(data_out, 1, GB(data_in,1) ^ GB(lfsr,21));
  SB(data_out, 0, GB(data_in,0) ^ GB(lfsr,22));
  return(data_out);
}


// Example of LFSR and scrambled data “0” sequence for Lane 0
main() {
  unsigned long lfsr = 0x1DBFBC; // Lane 0 reset LFSR value
  unsigned char unscrambled_data = 0x00;
  int i;
  printf("Iteration LFSR Next LFSR Scrambled Data\n");
  printf("- - - -\n");
  for (i = 0; i < 128; i++) {
    unsigned char scrambled_data = scramble_data(lfsr,unscrambled_data);
    unsigned long next_lfsr = calc_next_lfsr(lfsr);
    printf("%3d %06X %06X %02X\n", i, lfsr, next_lfsr, scrambled_data);
    lfsr = next_lfsr;
  }
}
```

The initial 23-bit values of the LFSR for the first 128 LFSR advances for an 8-bit quantity following a reset for Lane 0 are listed below (ordered left to right, top to bottom):

	0, 8	1, 9	2, A	3, B	4, C	5, D	6, E	7, F
00	1DBFBC	498C2E	1186E9	0FC5AD	7CB75D	3D8DA2	0ECC8F	379717
08	046BDF	21D462	423FCE	5177D6	1447EF	67CBA0	794F7A	6CA2AF
10	425060	3ED9D6	3DBF74	3C1A8F	7AE2E0	073E71	137D99	70B139
18	44F521	559720	1FBBA8	689D9F	376B02	597FFA	297C0E	7E450A
20	4BDE36	669B58	6BB530	78C3F9	0322C0	45C7FB	254F6A	323D89
28	07FDD2	71DFBC	68726B	799E27	1CFE8A	4AB864	42CB12	04AAF3
30	41F947	51F808	3A98CA	36A816	796895	4B4DAF	54037D	68E5C7
38	4F3302	60A51F	3AFCE3	528116	248131	1F65E6	17D2BA	34987E
40	6C0524	44DA45	7AF320	16FE71	5A5134	60B5F5	2A16E3	73AFF1
48	3D3ADA	18B5AA	4B9306	2BAB58	2D179E	5FDE68	46E1F8	644B91
50	3F78A4	7FCE1B	23CC59	7F017F	4DA97C	7EDF8B	705E13	0ADE04
58	6F1775	318CBE	70CC0C	39C021	0944ED	33F8AB	21DCBD	4AE0CE
60	1A6112	1B2F92	17ADD8	4BFA7E	42D358	1CE0F3	54C164	0BFDE2
68	0EF33F	082717	1200E1	4FCB73	39D53A	1C5FED	4ADE41	24EE12
70	7046E6	122B04	642E73	5A9AA4	0A24D0	34C250	362B24	5B5BB0
78	2833BF	73F640	648BDA	5E3281	490B97	373ECC	0CB1FA	6FE7A6

An 8-bit value of 0x00 repeatedly encoded with the LFSR after reset for Lane 0 produces the following consecutive 8-bit values (ordered left to right, top to bottom):

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	6C	BD	94	98	53	C6	D8	CE	50	6A	75	C1	04	4F	C3	07
10	75	26	C6	06	A3	B0	B4	AB	05	11	CC	57	4E	69	42	73
20	1D	0F	B7	03	E0	45	BA	5E	30	EB	D7	43	2C	5D	F5	D0
30	15	41	76	8E	C3	9D	D1	57	CD	FF	76	A1	7A	4C	64	2E
40	87	05	A3	24	89	FF	A2	4B	46	7C	1D	62	12	19	A5	2F
50	E6	B3	CA	33	ED	F3	2B	88	67	3E	AB	96	E8	9E	6A	5D
60	5C	1C	64	1D	F5	2C	51	C8	D8	A8	F4	4D	96	AC	5D	7A
70	2B	F4	2F	09	08	2E	0E	C9	02	4B	AF	D9	3D	4E	78	E7

Request Dependencies

This specification does not specify the rules governing the creation of resource dependencies between TLPs using different Traffic Classes. Dependencies between packets in different Traffic Classes can create potential deadlock if devices make different assumptions about what is allowed and what is not. Dependencies can be created when a packet is forwarded (transmitted verbatim) or translated (transmitted with modification) from an input Port to an output Port.

Resource dependencies are created when received packets are forwarded/translated on the same or a different Link. Due to the fact that the forwarding/translating device has finite buffer resources this behavior creates a dependency between the ability to receive a packet and the ability to transmit a packet (in potentially a different VC or sub-channel).

The following notation is used to create a framework to enumerate the possibilities:

X(m) → Y(n)

This means:

- a request in sub-channel *X*(*TC* = m) is forwarded/translated in sub-channel *Y*(*TC* = n).
- n and m are between 0-7.
- *X* and *Y* are either P (Posted Request), N (Non-Posted Request), or C (Completion).

The list of possible dependencies is:

P(m) → P(n)
P(m) → N(n)
P(m) → C(n)
N(m) → P(n)
N(m) → N(n)
N(m) → C(n)
C(m) → P(n)
C(m) → N(n)
C(m) → C(n)

For a given system, each of these dependencies needs to be classified as legal or illegal for each of the following cases:

- Root Port forwarding to own Link.
- Root Port forwarding to different Root Port's Link.
- Endpoint or Bridge forwarding to own Link.

A Switch is not allowed to modify the TLPs that flow through it, but must ensure complete independence of resources assigned to separate VCs. System software must comprehend the system dependency rules when configuring TC/VC mapping throughout the system.

One possible legal mapping is:

	RC (Same Port)	RC (Different Port)	Endpoint
P(m) → P(n)	m ≤ n	m ≤ n	m < n
P(m) → N(n)	m < n	m < n	m < n

	RC (Same Port)	RC (Different Port)	Endpoint
P(m) → C(n)	illegal	illegal	illegal
N(m) → P(n)	m < n	m < n	m < n
N(m) → N(n)	m ≤ n	m ≤ n	m < n
N(m) → C(n)	m = n	m = n	m = n
C(m) → P(n)	illegal	illegal	illegal
C(m) → N(n)	illegal	illegal	illegal
C(m) → C(n)	m ≥ n	m ≥ n	m > n

Note that this discussion only deals with avoiding the deadlock caused by the creation of resource dependencies. It does not deal with the additional livelock issues (or lack of forward progress) caused by the system's Virtual Channel arbitration policies.

Some of these potential dependencies are illegal or unreachable:

- P(m) → P(n), N(m) → N(n)
  - m = n - This case is illegal and will lead to deadlock, except when a Request is being forwarded from one Port to another of a Switch or Root Complex.
  - m ≠ n - See discussion below.
- P(m) → N(n)
  - m = n - This case is illegal and will lead to deadlock.
  - m ≠ n - See discussion below.
- N(m) → P(n) - See discussion below.
- P(m) → C(n) - This case is illegal and will lead to deadlock.
- N(m) → C(n)
  - m = n - This case occurs during the normal servicing of a non-posted request by either a root complex or an endpoint.
  - m ≠ n - This case is unreachable and should never happen. Completions always use the same TC as the corresponding request.
- C(m) → P(n), C(m) → N(n) - These cases are unreachable and should never happen due to the fact that completion buffers must be preallocated.
- C(m) → C(n)
  - m = n - This case is illegal and will lead to deadlock, except when a Completion is being forwarded from one Port to another of a Switch or Root Complex.
  - m ≠ n - This case will occur if N(n)→N(m) dependencies are allowed.

Other potential dependencies may be legal when comprehended as part of a specific usage model. For example, these cases:

P(m) → P(n), N(m) → N(n), P(m) → N(n), N(m) → P(n) where m ≠ n

might exist where a device services incoming requests by issuing modified versions of those requests using a different Traffic Class (for example, remapping the first requests address and generating the new request with the

resulting address). In these cases, suitable rules must be applied to prevent circular dependencies that would lead to deadlock or livelock.

Examples of devices that may find the above mappings useful:

- Bridges to complex protocols that require state to be save/restored to/from host memory, i.e., PCI Express to Infiniband bridges.
- Messaging engines that must do address translation based upon page tables stored in host memory.
- UMA graphics devices that store their frame buffer in host memory.

ID-Based Ordering Usage

E.1 Introduction

ID-Based Ordering (IDO) is a mechanism that permits certain ordering restrictions to be relaxed as a means to improve performance. IDO permits certain TLPs to pass other TLPs in cases where otherwise such passing would be forbidden. The passing permitted by IDO is not required for proper operation (e.g., deadlock avoidance); it is only a means of improving performance.

For discussing IDO, it's useful to introduce the concept of a "TLP stream", which is a set of TLPs that all have the same originator.[173] For several important cases where TLP passing is normally forbidden, IDO permits such passing to occur if the TLPs belong to different TLP streams.

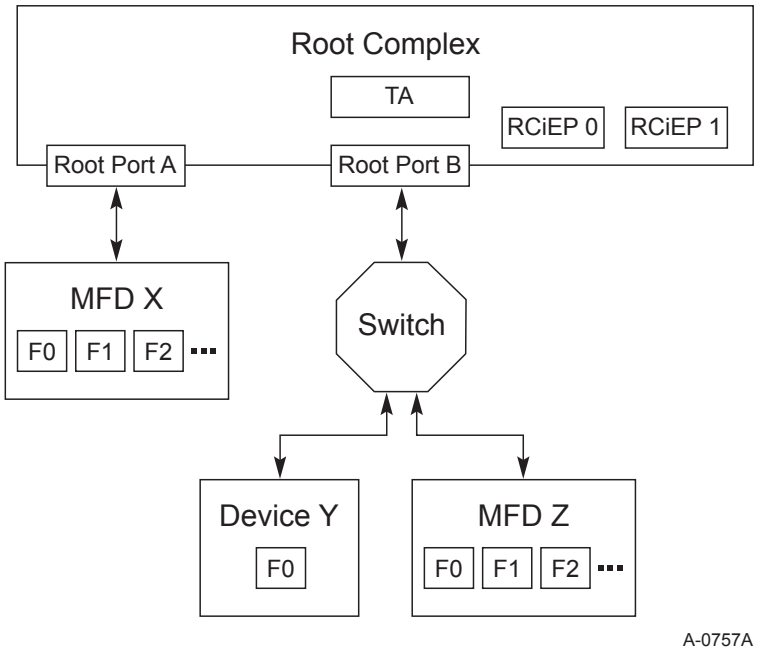


Figure E-1 Reference Topology for IDO Use

Figure E-1 shows a reference topology. The reference topology is not intended to discourage the use of IDO with other topologies, but rather to provide specific examples for discussion.

Devices X and Z are Multi-Function Devices (MFDs); device Y is a single-Function device. The RCiEPs are Root Complex Integrated Endpoint Functions, and might or might not be part of the same Device. We will assume that one or more Functions are using the Translation Agent (TA) in the Root Complex (RC).

Referring to the ordering table and descriptions in Section 2.4.1 , having the IDO bit set in a Posted Request, Non-Posted Request, or Completion TLP permits that TLP to pass a Posted Request TLP if the two TLPs belong to different TLP streams. In the following examples, DMAR and DMAW stand for Direct Memory Access Read and Write; PIOR and PIOW stand for Programmed I/O Read and Write.

173. That is, the Requester IDs of Requests and the Completer IDs of Completions are all the same.

E.2 Potential Benefits with IDO Use

Here are some example potential benefits that are envisioned with IDO use. Generally, IDO provides the most benefit when multiple TLP streams share a common Link and that Link becomes congested, either due to high utilization or due to temporary lack of Flow Control (FC) credit.

E.2.1 Benefits for MFD/RP Direct Connect

Here are some examples in the context of traffic between MFD X and the RC in Figure E-1 .

- Posted Request passing another Posted Request: when a DMAW from F0 is stalled due to an TA miss, if IDO is set for a DMAW from F1, it is permitted within the RC for this DMAW to pass the stalled DMAW from F0.
- Non-Posted Request passing a Posted Request: when a DMAW from F0 is stalled due to an TA miss, if IDO is set for a DMAR Request from F1, it is permitted within the RC for this DMAR Request to pass the stalled DMAW from F0.
- Completion passing a Posted Request: when a DMAW from F0 is stalled due to an TA miss, if IDO is set for a PIOR Completion from F1, it is permitted within the RC for this PIOR Completion to pass the stalled DMAW from F0.

E.2.2 Benefits for Switched Environments

Here are some examples in the context of traffic within the Switch in Figure E-1 .

- Non-Posted Request passing a Posted Request: when a DMAW from Device Y is stalled within the Switch due to a lack of FC credit from Root Port B, if IDO is set for a DMAR Request from MFD Z, it is permitted within the Switch for this DMAR Request to pass the stalled DMAW from Device Y. The same also holds for a DMAR Request from one Function in MFD Z passing a stalled DMAW from a different Function in MFD Z.
- Completion passing a Posted Request: when a DMAW from Device Y is stalled within the Switch due to a lack of FC credit from Root Port B, if IDO is set for a PIOR Completion from MFD Z, it is permitted within the Switch for this PIOR Completion to pass the stalled DMAW from Device Y. The same also holds for a PIOR Completion from one Function in MFD Z passing a stalled DMAW from a different Function in MFD Z.
- Posted Request passing another Posted Request: within a Switch, there is little or no envisioned benefit from having a DMAW from one TLP stream passing a DMAW from a different TLP stream. However, it is not prohibited for Switches to implement such passing as permitted by IDO.

E.2.3 Benefits for Integrated Endpoints

Here are some examples for the Root Complex Integrated Endpoints (RCiEPs) in Figure E-1 . The benefits are basically the same as for the MFD/RP Direct Connect case.

- Posted Request passing another Posted Request: when a DMAW from RCiEP 0 is stalled due to an TA miss, if IDO is set for a DMAW from RCiEP 1, it is permitted for this DMAW to pass the stalled DMAW from RCiEP 0.
- Non-Posted Request passing a Posted Request: when a DMAW from RCiEP 0 is stalled due to an TA miss, if IDO is set for a DMAR Request from RCiEP 1, it is permitted for this DMAR Request to pass the stalled DMAW from RCiEP 0.

- Completion passing a Posted Request: when a DMAW from RCiEP 0 is stalled due to an TA miss, if IDO is set for a PIOR Completion from RCiEP 1, it is permitted for this PIOR Completion to pass the stalled DMAW from RCiEP 0.

E.2.4 IDO Use in Conjunction with RO

IDO and RO[174] are orthogonal. Certain instances of passing; for example, a Posted Request passing another Posted Request, might be permitted by IDO, RO, or both at the same time. While IDO and RO have significant overlap for some cases, it is highly recommended that both be used whenever safely possible. RO permits certain TLP passing within the same TLP stream, which is never permitted by IDO. For traffic in different TLP streams, IDO permits control traffic to pass any other traffic, and generally it is not safe to Set RO with control traffic.

E.3 When to Use IDO

With Endpoint Functions[175] , it is safe to Set IDO in all applicable TLPs originated by the Endpoint when the Endpoint is directly communicating with only one other entity, most commonly the RC. For the RC case, “directly communicating” specifically includes DMA traffic, PIO traffic, and interrupt traffic; communicating with RCiEPs or communicating using P2P Root Port traffic constitutes communicating with multiple entities.

With a Root Port, there are no envisioned high-benefit use models where it is safe to Set IDO in all applicable TLPs that it originates. Use models where a Root Port Sets IDO in a subset of the applicable TLPs it originates are outside the scope of this specification.

E.4 When Not to Use IDO

E.4.1 When Not to Use IDO with Endpoints

With Endpoint Functions, it is not always safe to Set IDO in applicable TLPs it originates if the Endpoint directly communicates with multiple entities. It may be safe to Set IDO in some TLPs and not others, but such use models are outside the scope of this specification.

For example, in Figure E-1 if Device Y and MFD Z are communicating with P2P traffic and also communicating via host memory, it is not always safe for them to Set IDO in the TLPs they originate. As an example failure case, let’s assume that Device Y does a DMAW (to host memory) followed by a P2P Write to MFD Z. Upon observing the P2P Write, let’s assume that MFD Z then does a DMAW to the same location earlier targeted by the DMAW from Device Y. Normal ordering rules would guarantee that the DMAW from Device Y would be observed by host memory before the DMAW from MFD Z. However, if IDO is set in the DMAW from MFD Z, the RC would be permitted to have the second DMAW pass the first, causing a different end result in host memory contents.

Synchronization techniques like performing zero-length Reads might be used to avoid such communication failures when IDO is used, but specific use models are outside the scope of this specification.

174. In this Appendix, “RO” is an abbreviation for the Relaxed Ordering Attribute field.
175. Endpoint Functions include PCI Express Endpoints, Legacy PCI Express Endpoints, and Root Complex Integrated Endpoints.

E.4.2 When Not to Use IDO with Root Ports

With Root Ports, it is not always safe to Set IDO in applicable TLPs it originates if Endpoint Functions in the hierarchy do any P2P traffic. It may be safe to Set IDO in some TLPs and not others, but such use models are outside the scope of this specification.

As an example, in Figure E-1 if Device Y and MFD Z are communicating with P2P traffic and also communicating with host software, it is not always safe for Root Port B to Set IDO in the TLPs it originates. For example, let's assume that Device Y does a P2P Write to MFD Z followed by a DMAW (to host memory). Upon observing the DMAW, let's assume that the host does a PIOW to MFD Z. Normal ordering rules would guarantee that the P2P Write from Device Y would be observed by MFD Z before the PIOW from the host. However, if IDO is set in the PIOW from the host, the Switch would be permitted to have the PIOW pass the P2P Write, ultimately having the two Writes arrive at MFD Z out of order.

IMPLEMENTATION NOTE

Requester and Completer IDs for RC-Originated TLPs

With RC implementations where the Requester ID in a PIO Request does not match the Completer ID in a DMAR Completion, this enables another potential communication failure case if IDO is Set in the Completion. For this case, if a PIOW is followed by a DMAR Completion with IDO Set, a Switch below the Root Port could permit the DMAR Completion to pass the PIOW, violating the normal ordering rule that a non-RO Read Completion must not pass Posted Requests. The PIOW and DMAR Completion would appear to belong to different TLP streams, though logically they belong to the same TLP stream. Special caution is advised in setting IDO with TLPs originating from such RCs.

E.5 Software Control of IDO Use

E.5.1 Software Control of Endpoint IDO Use

By default, Endpoints are not enabled to Set IDO in any TLPs they originate.

IMPLEMENTATION NOTE

The "Simple" Policy for IDO Use

It is envisioned that Endpoints designed primarily to communicate directly with only one other entity (e.g., the RC) may find a "simple" policy for setting IDO to be adequate. Here's the envisioned "simple" policy. If the IDO Request Enable bit is Set, the Endpoint Sets IDO in all applicable Request TLPs that it originates. If the IDO Completion Enable bit is Set, the Endpoint Sets IDO in all Completion TLPs that it originates.

It is envisioned that a software driver associated with each Endpoint will determine when it is safe for the Endpoint to set IDO in applicable TLPs it originates. A driver should be able to determine if the Endpoint is communicating with multiple other entities, and should know the Endpoint's capabilities as far as setting IDO with all applicable TLPs when enabled, versus setting IDO selectively. If a driver determines that it is safe to enable the setting of IDO, the driver can set the IDO Request Enable and/or IDO Completion Enable bits either indirectly via OS services or directly, subject to OS policy.

If an Endpoint is designed for communication models where it is not safe to utilize the “simple” policy for IDO use, the Endpoint can implement more complex policies for determining when the Endpoint sets the IDO bit. Such implementations might utilize device-specific controls that are managed by the device driver. Such policies and device-specific control mechanisms are outside the scope of this specification.

E.5.2 Software Control of Root Port IDO Use

Since there are no envisioned high-benefit “simple” use models for Root Ports setting the IDO bit with TLPs they originate, and there are known communication failure cases if Root Ports set the IDO bit with all applicable TLPs they originate, it is anticipated that Root Ports will rarely be enabled to set IDO in TLPs they originate. Such use models and policies for Root Ports setting IDO are outside the scope of this specification.

Message Code Usage

Table F-1 contains a list of currently defined PCI Express Message Codes. Message codes are defined in this specification and in other specifications. This table will be updated as Messages are defined in other specifications but due to document release schedules, this table might not contain recently defined Messages.

Table F-1 Message Code Usage

Message Code	Routing r[2:0]	Type	Description
0000 0000	011	Msg	Unlock, see Section 2.2.8.4
0000 0001	010	MsgD	Invalidate Request Message, see Section 10.3.1
0000 0010	010	Msg	Invalidate Completion Message, see Section 10.3.2
0000 0100	000	Msg	Page Request Message, see Section 10.4.1
0000 0101	010	Msg	PRG Response Message, see Section 10.4.2
0001 0000	100	Msg	Latency Tolerance Reporting (LTR) Message, see Section 2.2.8.8
0001 0010	100	Msg	Optimized Buffer Flush/Fill (OBFF) Message, see Section 2.2.8.9
0001 0100	100	Msg	PM_Active_State_Nak, see Section 2.2.8.2
0001 1000	000	Msg	PM_PME, see Section 2.2.8.2
0001 1001	011	Msg	PME_Turn_Off, see Section 2.2.8.2
0001 1011	101	Msg	PME_TO_Ack, see Section 2.2.8.2
0010 0000	100	Msg	Assert_INTA, see Section 2.2.8.1
0010 0001	100	Msg	Assert_INTB, see Section 2.2.8.1
0010 0010	100	Msg	Assert_INTC, see Section 2.2.8.1
0010 0011	100	Msg	Assert_INTD, see Section 2.2.8.1
0010 0100	100	Msg	Deassert_INTA, see Section 2.2.8.1
0010 0101	100	Msg	Deassert_INTB, see Section 2.2.8.1
0010 0110	100	Msg	Deassert_INTC, see Section 2.2.8.1
0010 0111	100	Msg	Deassert_INTD, see Section 2.2.8.1
0011 0000	000	Msg	ERR_COR, see Section 2.2.8.3
0011 0001	000	Msg	ERR_NONFATAL, see Section 2.2.8.3
0011 0011	000	Msg	ERR_FATAL, see Section 2.2.8.3
0100 0000	100	Msg	Ignored Message, see Section 2.2.8.7

Message Code	Routing r[2:0]	Type	Description
0100 0001	100	Msg	Ignored Message, see Section 2.2.8.7
0100 0011	100	Msg	Ignored Message, see Section 2.2.8.7
0100 0100	100	Msg	Ignored Message, see Section 2.2.8.7
0100 0101	100	Msg	Ignored Message, see Section 2.2.8.7
0100 0111	100	Msg	Ignored Message, see Section 2.2.8.7
0100 1000	100	Msg	Ignored Message, see Section 2.2.8.7
0101 0000	100	MsgD	Set_Slot_Power_Limit, see Section 2.2.8.5
0101 0010	100	Msg	PTM Request, see Section 2.2.8.10
0101 0011	100	Msg/MsgD	PTM Response/PTM ResponseD, see Section 2.2.8.10
0111 1110	000, 010, 011, or 100	Msg/MsgD	Vendor_Defined Type 0, see Section 2.2.8.6
0111 1111	000, 010, 011, or 100	Msg/MsgD	Vendor_Defined Type 1, see Section 2.2.8.6

Table F-2 contains a list of currently defined Subtype codes for PCI-SIG-Defined VDMs (see Section 2.2.8.6.1). Subtype codes are defined in this specification and in other specifications. This table will be updated as Subtype codes are defined in other specifications but due to document release schedules, this table might not contain recently defined Subtypes.

Table F-2 PCI-SIG-Defined VDM Subtype Usage

Subtype	Routing r[2:0]	Type	Description
0000 0000	010 or 011	MsgD	LN Message, see Section 2.2.8.6.2
0000 0001	011	MsgD	Hierarchy ID Message, See Section 2.2.8.6.5 and Section 6.26
0000 1000	100	Msg	Device Readiness Status, see Section 2.2.8.6.3
0000 1001	000	Msg	Function Readiness Status, see Section 2.2.8.6.4

Protocol Multiplexing

G.

The Protocol Multiplexing mechanism provides a standard mechanism to transport non-PCI Express protocols across a PCI Express Link. The mechanism supports the multiplexing of PMUX Packets and TLPs onto a single PCI Express Link.

An example system topology using Protocol Multiplexing is shown in Figure G-1 . In this example, the Link may operate in two modes:

- PCI Express Link. Protocol Multiplexing is disabled.
- PMUX Link. Protocol Multiplexing is enabled. Both TLPs and PMUX Packets are used in a coordinated fashion. PMUX Packets may be used to support additional protocols efficiently.

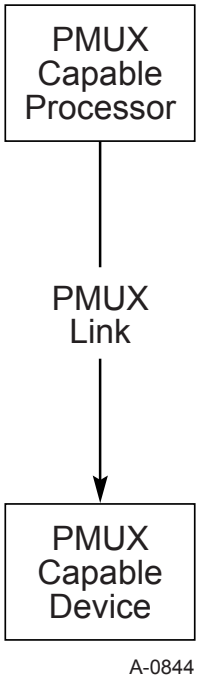


Figure G-1 Device and Processor Connected Using a PMUX Link

A PMUX Link is shown in Figure G-2 . Arbitration and encapsulation occurs between the transmit queues and the Link. Demultiplexing and decapsulation occurs between the Link and the various receive queues. Packets are sent from transmit queues to the corresponding receive queues. Packets are identified as either PMUX Packets or TLPs.

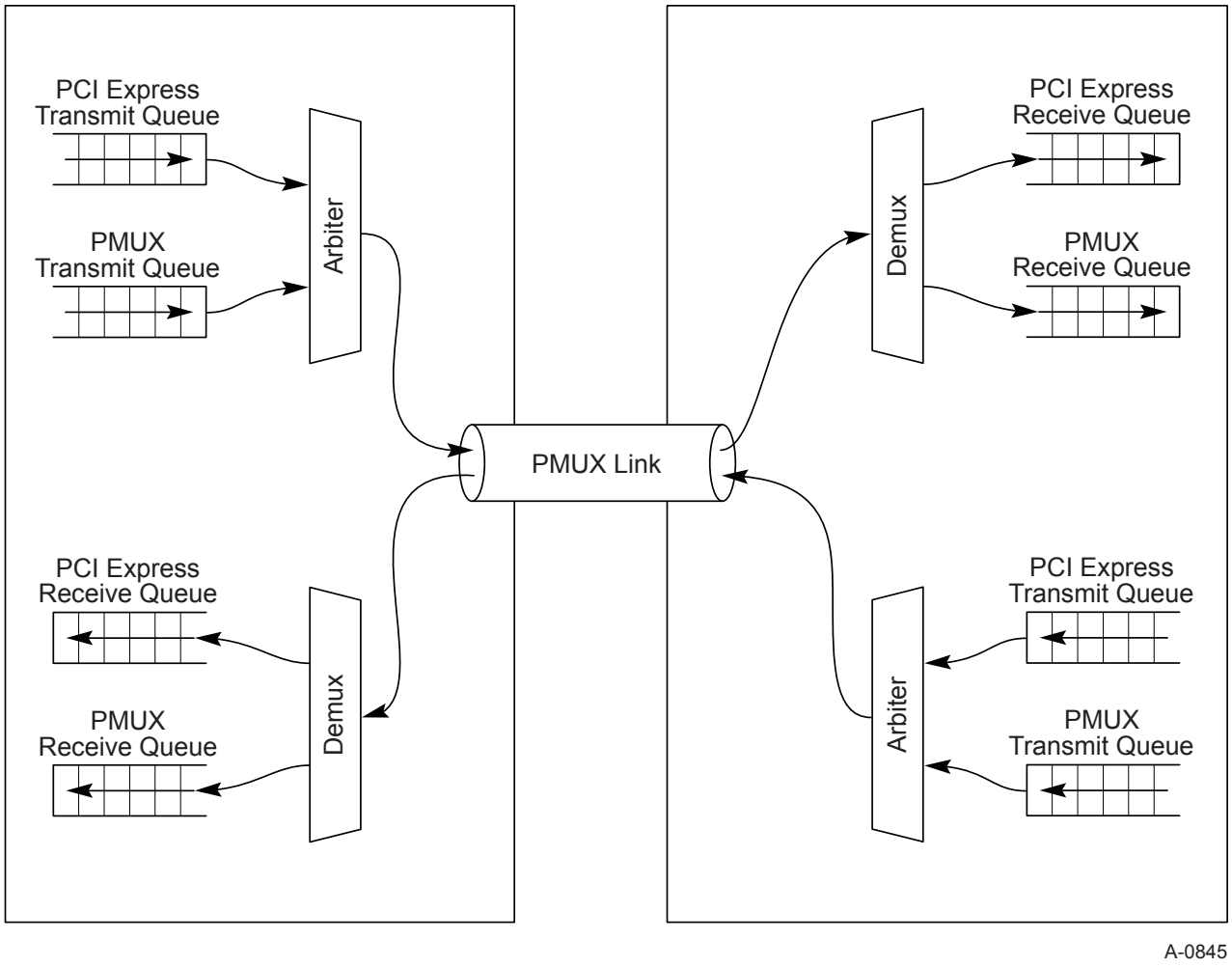


Figure G-2 PMUX Link

Important attributes of the Protocol Multiplexing mechanism are:

- Protocol Multiplexing support is optional normative.
- Protocol Multiplexing has no impact on PCI Express components that do not support it.
- Protocol Multiplexing has no impact on PCI Express TLPs and DLLPs, even when it is enabled.
- A Link may be used for both TLPs and PMUX Packets at the same time.
- Protocol Multiplexing does not consume or interfere with PCI Express resources (sequence numbers, credits, etc.). PMUX Packets use distinct resources associated with the specific multiplexed protocol.
- Protocol Multiplexing is disabled by default and is enabled by software. PMUX Packets must not be sent unless enabled by software. PMUX Packets received at Ports that support Protocol Multiplexing are ignored until Protocol Multiplexing is enabled by software.
- Protocol Multiplexing is selectable on a per-Link basis. Protocol Multiplexing may be used on any collection of Links in a system.
- A PMUX Link may support up to four simultaneously active PMUX Channels. Software configures the protocol used on each PMUX Channel.

- PMUX Packets contain an LCRC. This is used to provide data resiliency in a similar fashion as PCI Express TLPs.
- PMUX Packets do not use the ACK/NAK mechanism of PCI Express. Multiplexed protocol specific acknowledgement mechanisms can be used to provide reliable delivery when needed.
- PMUX Packets do not contain a TLP Sequence Number. Instead, they contain a 12 bit PMUX Packet Metadata field that is available for multiplexed protocol specific use.
- PMUX Packet transmitters must contain some arbitration/QoS mechanism for scheduling sending of PMUX Packets, TLPs and DLLPs; however, the mechanism used is outside the scope of this specification.
- The Protocol Multiplexing mechanism does not define any addressing or routing mechanism for PMUX Packets.

PMUX Packets are similar to PCI Express TLPs. The PMUX Packet Flow Through is shown in Figure G-3 . The PCI Express Packet flow through the layers is shown in Figure 1-5 . Changes from PCI Express Packet Flow Through are:

- PMUX Packets use a protocol specific PMUX Protocol Layer instead of the PCI Express Transaction Layer.
- PMUX Packets use a simplified Data Link Layer. The packet integrity portion of the Data Link Layer is mostly unchanged (LCRC computation uses a different seed value). The reliability and flow control aspects of the Data Link Layer are removed (the TLP Sequence Number field is repurposed as PMUX Packet Metadata).
- The Physical Layer is slightly modified to provide a mechanism to identify PMUX Packets.

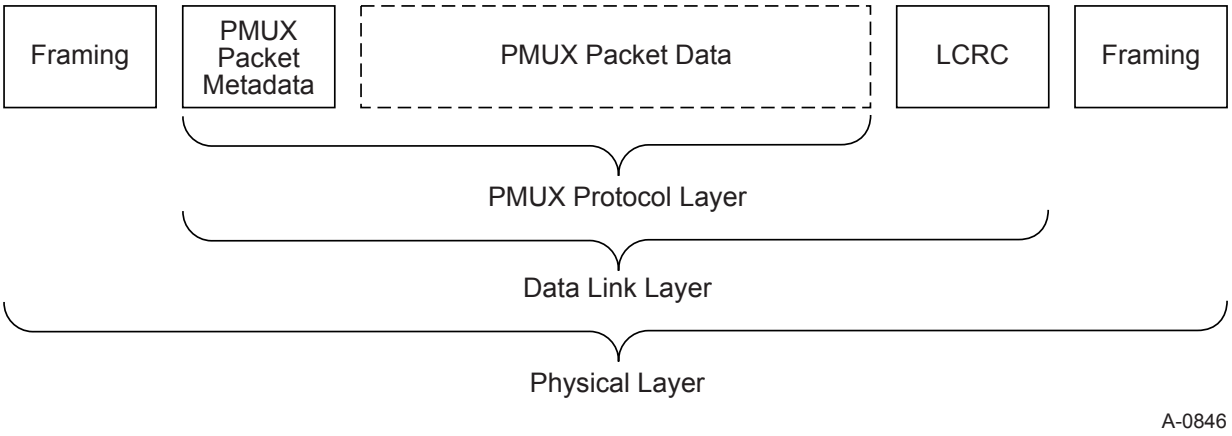


Figure G-3 PMUX Packet Flow Through the Layers

G.1 Protocol Multiplexing Interactions with PCI Express

Table G-1 and Table G-2 describe interactions between Protocol Multiplexing and PCI Express. Table G-1 describes how PCI Express attributes affect Protocol Multiplexing. Table G-2 describes how Protocol Multiplexing features affect PCI Express attributes.

Table G-1 PCI Express Attribute Impact on Protocol Multiplexing

PCI Express Attribute	Impact on Protocol Multiplexing
Link Speed	All PMUX Channels are disabled when the Current Link Speed corresponds to a speed that is not supported by Protocol Multiplexing (see Section G.5.4). A PMUX Channel may be disabled when the Current Link Speed corresponds to a speed that is not supported by the associated protocol.[176]

176. The mechanism software uses to determine what Link Speeds are supported by a protocol is outside the scope of this specification.

PCI Express Attribute	Impact on Protocol Multiplexing
	Link speed can change either explicitly due to a change in the Target Link Speed field or automatically due to an autonomous Link speed change (see Section 6.11). The PMUX Protocol Layer is permitted to influence the mechanism used by a component to determine when it requests an autonomous Link speed change. In addition, setting Hardware Autonomous Speed Disable at each end of the Link will prevent certain autonomous Link speed changes (see Section 7.5.3.18). The PMUX Protocol Layer may be notified of the change.
Link Width	A PMUX Channel may be disabled when the Link Width corresponds to a width that is not supported by the associated protocol.[177] The PMUX Protocol Layer is permitted to influence the mechanism used by a component to determine when it requests a Link Width change. The PMUX Protocol Layer may be notified of the change.
FLR initiated	All PMUX Channels on a Link are disabled if an FLR is directed to the Upstream Port's Function 0. No PMUX Channels are affected if an FLR is directed to any other Function.
DL_Down	All PMUX Channels on a Link are disabled.
Hot Reset	All PMUX Channels on a Link are disabled.
PERST#	All PMUX Channels on a Link are disabled.
TLP Replay	No effect on Protocol Multiplexing.
DLLP Lost	No effect on Protocol Multiplexing.
TLP Prefix	No effect on Protocol Multiplexing.
Locked Transactions	No effect on Protocol Multiplexing (including no effect on any protocol specific forwarding of PMUX Packets).
AtomicOp Transactions	No effect on Protocol Multiplexing.
Multicast Transactions	No effect on Protocol Multiplexing.
Access Control Services (ACS)	No effect on Protocol Multiplexing.
Alternative Routing ID-Interpretation (ARI)	No effect on Protocol Multiplexing.
TPH Requester	No effect on Protocol Multiplexing.
Virtual Channels	No effect on Protocol Multiplexing. PCI Express Links remain capable of supporting a full complement of VCs.
Internal Error	Corrected or Uncorrectable Internal Errors in the PMUX Protocol Layer may be reported as PCI Express Internal Errors.
L0s Link Power State	Protocol Multiplexing tracks the Link state. The PMUX Protocol Layer may request the Link transition back to L0.

177. The mechanism software uses to determine what Link Widths are supported by a protocol is outside the scope of this specification.

PCI Express Attribute	Impact on Protocol Multiplexing
L1 Link Power State	Protocol Multiplexing tracks the Link state. The PMUX Protocol Layer may request the Link transition back to L0.
Disabled LTSSM State	Disabling a Link also disables all PMUX Channels on the Link.
Loopback LTSSM State	Entering Loopback state disables all PMUX Channels on the Link.
Recovery LTSSM State	No effect on Protocol Multiplexing. The PMUX Protocol Layer may be notified.
Receiver or Framing Error	The error is reported to the PMUX Protocol Layer to indicate that data might have been lost. This can be used to initiate protocol specific error recovery mechanisms. The Error is reported to software using PCI Express Mechanisms.
Lane Reversal	No effect on Protocol Multiplexing. Support for Lane Reversal remains optional.
Polarity Inversion	No effect on Protocol Multiplexing.
Crosslink	No effect on Protocol Multiplexing. Support for Crosslink remains optional. If supported, the PMUX Protocol Layer may be notified of the outcome of the Crosslink Upstream / Downstream negotiation.
Lane assignment rules	Placement and frequency rules for STP Symbols and STP Tokens are not changed (see Section 4.2.1.2 and Section 4.2.2.3.2). These rules apply identically to PCI Express TLPs and PMUX Packets.
PCI Power Management Power State	All PMUX Channels on a Link are disabled if the Upstream Port's Function 0 is sent to non-D0 state.
Dynamic Power Allocation (DPA)	No effect on Protocol Multiplexing. The PMUX Protocol Layer is notified of the change and may participate in the power reduction. PCI Express power management includes any power used by the PMUX Protocol Layer.
PCI Power Management Power Consumed / Power Dissipated / Aux_Current	Power required by the PMUX Protocol Layer is included in the PCI structures.
Power Budgeting	Power required by the PMUX Protocol Layer is included in the PCI Express structures.
Slot Power Limit	Slot Power Limit includes power available to PMUX Protocol Layer.
ASPM L0s Entry Condition	The definition of Idle is extended to include: • No pending PMUX Packets to transmit over the Link. • For PMUX Channels that use protocol specific Flow Control, no credits are available to send PMUX Packets in that PMUX Channel.
ASPM L1 Entry Condition	A Link may not enter L1 if PMUX Packets are pending or scheduled to be transmitted.
ASPM L0s/L1 Exit Conditions	A Link may be directed to exit L0s or L1 if a component needs to transmit a PMUX Packet. Routing of PMUX Packets through routing elements is outside the scope of this specification; the associated L0s /L1 exit rules are also unspecified.
Bus Renumbering	No effect on Protocol Multiplexing.
Hot Plug	No direct effect on Protocol Multiplexing. Note Hot Plug events indirectly affect Data Link State which, in turn affects Protocol Multiplexing.

PCI Express Attribute	Impact on Protocol Multiplexing
TLP Sequence Number	No effect on Protocol Multiplexing. PMUX Packets do not consume TLP Sequence Numbers.
PCI Express Flow Control	No effect on Protocol Multiplexing. PMUX Packets do not consume PCI Express Flow Control credits. Flow Control Update DLLPs must be sent as required by PCI Express.
Error Reporting	No direct effect on Protocol Multiplexing. The PMUX Protocol Layer may be notified when an error is signaled or when an error message is received.
LCRC Errors in TLPs	No effect on Protocol Multiplexing.
Nullified TLPs	No effect on Protocol Multiplexing.
VC Arbitration	No effect on Protocol Multiplexing. Arbitration within PCI Express is unaffected by Protocol Multiplexing.
Port Arbitration	No effect on Protocol Multiplexing. Arbitration within PCI Express is unaffected by Protocol Multiplexing.
Electrical Idle Inference	PMUX Packets count as TLPs for the purpose of inferring Electrical Idle.
MR-IOV	Protocol Multiplexing may co-exist with MR-IOV. PMUX Packets are not part of any MR-IOV Virtual Hierarchy. Protocol Multiplexing is controlled using configuration space in the Management VH(s).

Table G-2 PMUX Attribute Impact on PCI Express

PMUX Attribute	Impact on PCI Express
PMUX Protocol Error	No effect on PCI Express.
LCRC Errors in PMUX Packets	No effect on PCI Express. PMUX Packets with LCRC errors are discarded without triggering PCI Express replay. This error is reported to the PMUX Protocol Layer and can be used to initiate protocol specific error recovery and/or error reporting mechanisms.
Link Unreliability	The PMUX Protocol Layer is permitted to influence the mechanism used by a component to determine if it requests an autonomous link speed change.
Nullified PMUX Packets	No effect on PCI Express. It is protocol specific whether PMUX Packets within a specific PMUX Channel may be nullified. If supported, PMUX Packets are nullified in the same manner as TLPs (e.g., inverting the LCRC and signaling nullification at the Physical Layer). Receiving a nullified PMUX Packet may be reported to the PMUX Protocol Layer.
Electrical Idle Inference	PMUX Packets count as TLPs for the purpose of inferring Electrical Idle.
PMUX Protocol Layer directs LTSSM to enter Recovery	Both PCI Express and the PMUX Protocol Layer are permitted to direct a transition from L0 to Recovery.
PMUX Channel Enabled / Disabled	No effect on PCI Express
PMUX Packet Receiver Buffer Overflow	No effect on PCI Express. This is a protocol problem within the PMUX Channel. The PMUX Transport Layer must continue to accept such packets dispose of them using protocol specific mechanisms.
Received PMUX Packet larger or smaller than supported by the associated protocol	No effect on PCI Express. These are protocol problems within the PMUX Channel. The PMUX Transport Layer must accept such packets and dispose of them using protocol specific mechanisms.
Received PMUX Packet that contains more than	No effect on PCI Express. This is an invalid PMUX Packet. The PMUX Transport Layer must accept such a packet and dispose of it. A protocol specific mechanism may be used to report the error.

PMUX Attribute	Impact on PCI Express
125 DWORDs of PMUX Packet Data	Note: This situation only exists for a packet encoded using 8b/10b. The TLP Length field of a packet encoded using 128b/130b cannot contain values that cause this situation.
PMUX Packet Received on disabled PMUX Channel	No effect on PCI Express. No effect on any other PMUX Channel. Receivers must silently ignore such packets regardless of packet length and regardless of whether or not the packet is nullified. PMUX Packets arriving on a disabled PMUX Channel may occur normally when software is in the process of initializing Protocol Multiplexing.
PMUX Packet Received at component that does not support Protocol Multiplexing	Software should not enable PMUX Packets unless both ends of a Link support Protocol Multiplexing. In the 128b/130b encoding, receiving a PMUX Packet by a component that does not support Protocol Multiplexing is a Framing Error (see Section 4.2.2.3.1). In the 8b/10b encoding, the PMUX Packet LCRC is computed differently that the TLP LCRC. Receivers that do not support Protocol Multiplexing will interpret PMUX Packets as TLPs with LCRC errors and will not process them.
Large PMUX Packets when PCI Express Max_Payload_Size is small	Under certain conditions, it is possible for a large PMUX Packet to trigger a premature PCI Express replay. For example, this can occur when the time needed to transmit a PMUX Packet is larger than the REPLAY_TIMER (see Section 3.6.2.1). To avoid this issue, implementations are permitted to not advance (hold) their REPLAY_TIMER during the reception of PMUX Packets. Note: The PCI Express REPLAY_TIMER mechanism has adequate headroom for most cases. This issue exists when (1) Max_Payload_Size is 000b, (2) PMUX Packets are larger than about 80 DWORDs, and (3) the REPLAY_TIMER is at the low end of the -0%/+100% tolerance.

G.2 PMUX Packets

A PMUX Packet contains the information shown in Figure G-4 .

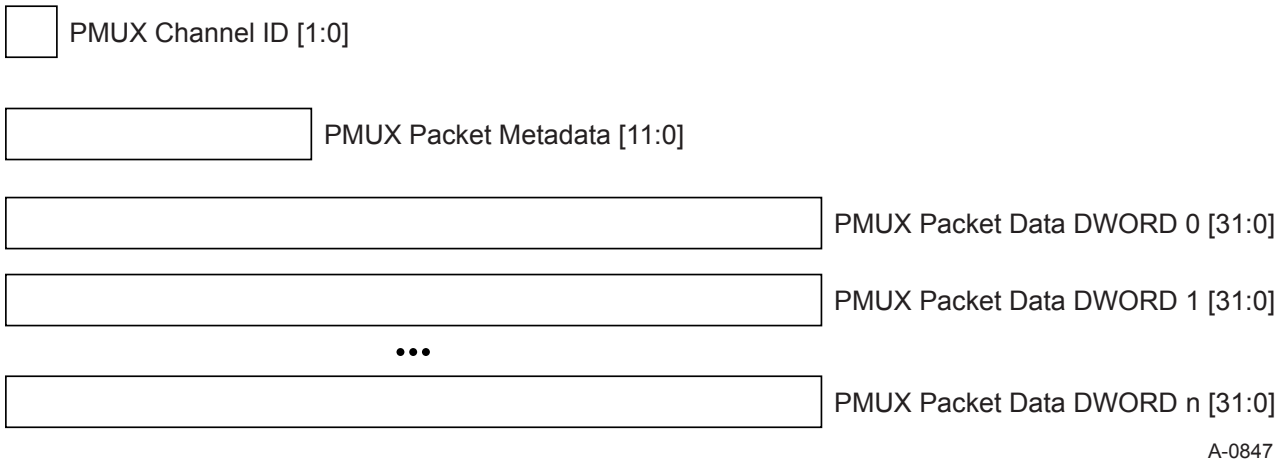


Figure G-4 PMUX Packet

PMUX Channel ID is a 2 bit field that identifies which protocol is associated with a PMUX Packet. PMUX Channel ID values are between 0 and 3 (inclusive).

PMUX Packet Metadata is a 12 bit field that provides information about the PMUX Packet. Definition of this field is protocol specific and is outside the scope of this specification.

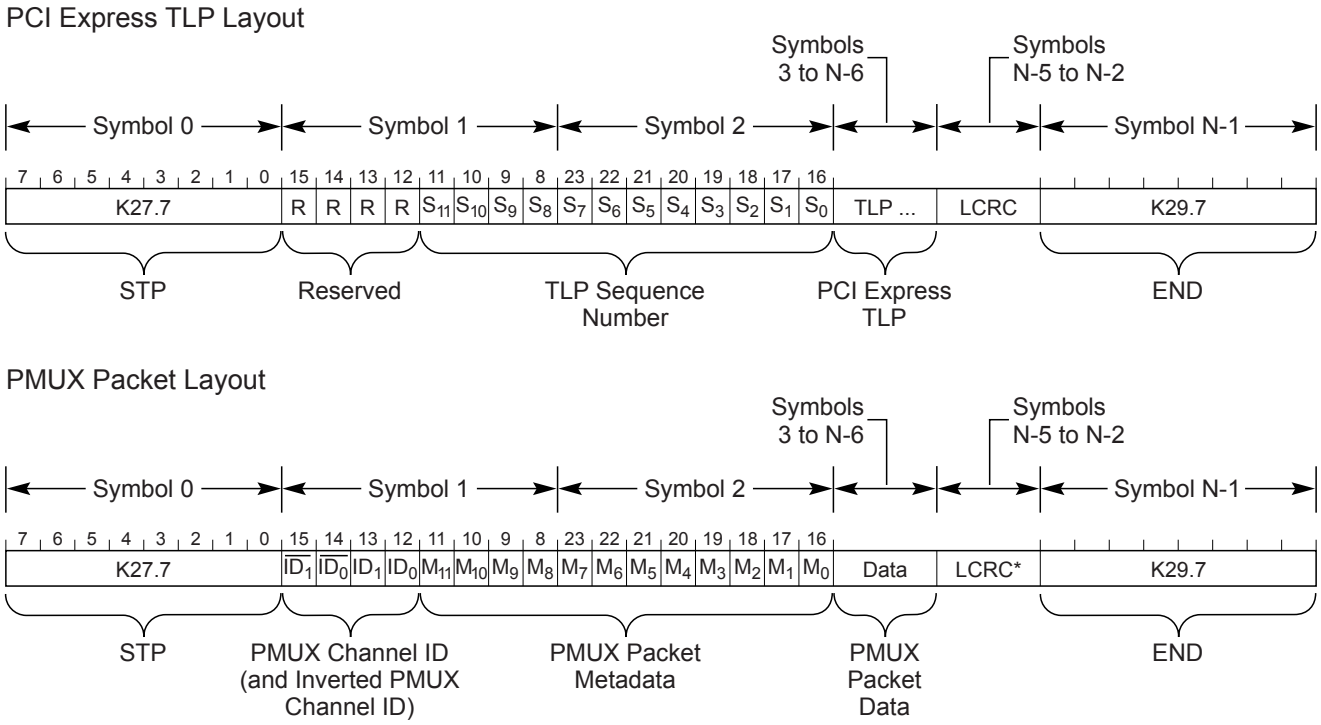
A PMUX Packet consists of between 0 and 125 DWORDs of PMUX Packet Data. Layout and usage of these DWORDs is protocol specific and is outside the scope of this specification. A PMUX Packet need not have any PMUX Packet Data and may consist only of PMUX Channel ID and PMUX Packet Metadata.

G.3 PMUX Packet Layout

There are two layouts defined for PMUX Packets. One layout is used for 2.5 and 5.0 GT/s data rates and another layout is used for 8.0 GT/s and higher data rates. These layouts are discussed in the following sections.

G.3.1 PMUX Packet Layout for 8b/10b Encoding

Figure G-5 and Table G-3 show the layout of PMUX Packets when using 8b/10b encoding. For reference, the 8b/10b encoding of a TLP is also shown (see Section 4.2.1.2 for the official definition).



* LCRC for PMUX Packets uses a different starting seed than that used for TLPs. This avoids aliasing problems and potential errors if software misconfigures a link so that PMUX Packets are seen by existing silicon.

A-0848

Figure G-5 TLP and PMUX Packet Framing (8b/10b Encoding)

Table G-3 PMUX Packet Layout (8b/10b Encoding)

Symbol	Field	Bit Position(s)	PMUX Packet Usage	TLP Usage
0	Start TLP Indicator	7:0	K27.7	
1	Inverted PMUX Channel ID[1:0]	7:6	Inverted (1s complement) of Symbol 1 bits 6:4	Reserved
	PMUX Channel ID[1:0]	5:4	PMUX Channel ID	Reserved
	PMUX Packet Metadata[11:8]	3:0	PMUX Packet Metadata[11:8]	TLP Sequence Number[11:8]
2	PMUX Packet Metadata[7:0]	7:0	PMUX Packet Metadata[7:0]	TLP Sequence Number[7:0]
3 to N-6	Packet	7:0	PMUX Packet	TLP
N-5 to N-2	LCRC	7:0	PMUX LCRC	PCI Express LCRC
N-1	END	7:0	K29.7	

For PMUX Packets, symbols 1 and 2 contain PMUX Packet Metadata in the same bit positions that TLPs use for TLP Sequence Number.

The PMUX LCRC algorithm is identical to the TLP LCRC algorithm as described in Section 3.6.2 with the following modifications:

- The seed value is FB3E E248h (TLP LCRC uses FFFF FFFFh).
- The PMUX Channel ID field in Symbol 1 bits 7:4 is included in the PMUX LCRC in the same manner as the 4 reserved bits in the TLP LCRC.
- The PMUX Packet Metadata field is included in the PMUX LCRC in the same manner as the TLP Sequence Number field is included in the TLP LCRC.

IMPLEMENTATION NOTE

PMUX Packets at Receivers that do not Support Protocol Multiplexing

The bits used for PMUX Channel ID are reserved unless Protocol Multiplexing is supported. As such, Receivers that do not support Protocol Multiplexing must ignore the PMUX Channel ID bits. If software misconfigures Protocol Multiplexing, a component that does not support Protocol Multiplexing could receive a PMUX Packet. To prevent that component from misinterpreting such a PMUX Packet as a valid TLP, the LCRC computation is changed for PMUX Packets. The result is that a valid PMUX Packet will never be misinterpreted as a valid TLP. These LCRC "errors" may trigger PCI Express replay and may result in REPLAY_NUM Rollover correctable errors being reported.

IMPLEMENTATION NOTE

PMUX Packet LCRC

The PMUX Channel ID field is covered by the LCRC. As such, when using 8b/10b encoding, receivers must wait until the LCRC is checked to make firm decisions based on the PMUX Channel ID value. The Inverted PMUX Channel ID can be compared against the PMUX Channel ID to make tentative decisions.

Note: The value of the LCRC associated with a given PMUX Packet is independent of the encoding used to transmit the packet.

G.3.2 PMUX Packet Layout at 128b/130b Encoding

Figure G-6 and Table G-4 show the layout of PMUX Packets when using 128b/130b encoding. For reference, the 128b/130b encoding of a TLP is also shown (see Section 4.2.2.2 for the official definition).

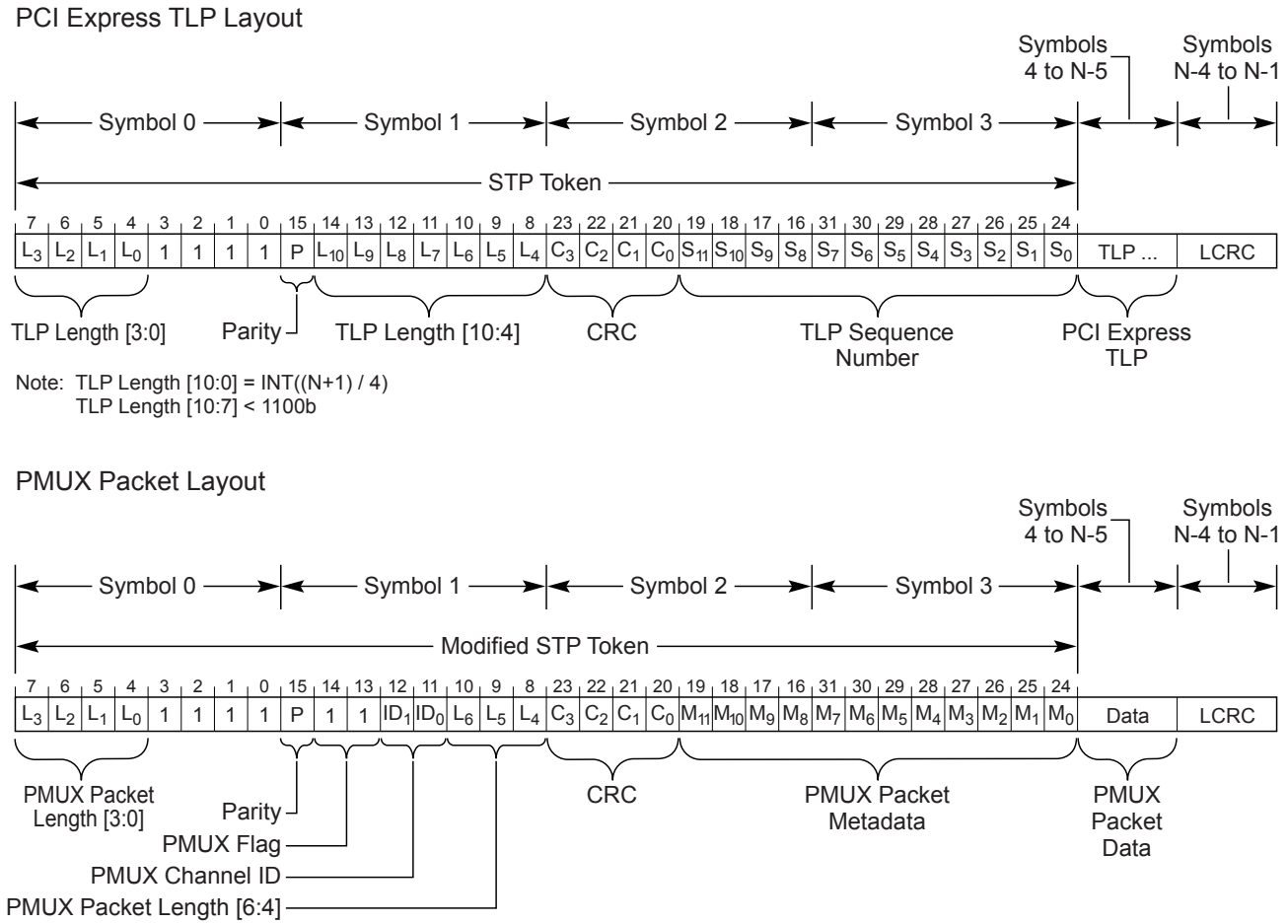


Figure G-6 TLP and PMUX Packet Framing (128b/130b Encoding)

Table G-4 PMUX Packet Layout (128b/130b Encoding)

Symbol	Field	Bit Position(s)	PMUX Packet Usage	TLP Usage
0	Start TLP Indicator	3:0	Value of 1111b	
	PMUX Packet Length[3:0]	7:4	Bits [3:0] of the PMUX Packet Length. Bit 0 is the least significant PMUX Packet Length bit.	Bits [3:0] of the TLP Length field. Bit 0 is the least significant TLP Length bit.
1	Frame Parity (P)	7	Even parity of Symbol 0 bits [7:4], Symbol 1 bits [6:0] and Symbol 2 bits [7:4]	

Symbol	Field	Bit Position(s)	PMUX Packet Usage	TLP Usage
	PMUX Packet Indicator	6:5	Value of 11b	Bits [10:4] of the TLP Length field. Bit 10 is the most significant TLP Length bit.
	PMUX Channel ID[1:0]	4:3	PMUX Channel ID	
	PMUX Packet Length[6:4]	2:0	Bits [6:4] of PMUX Packet Length. Bit 6 is the most significant PMUX Packet Length bit.	
2	PMUX Packet Metadata[11:8]	3:0	PMUX Packet Metadata[11:8]	TLP Sequence Number[11:8]
	Frame CRC (C[3:0])	7:4	CRC of Symbol 0, bits [7:4] and Symbol 1 bits [6:0]	
3	PMUX Packet Metadata[7:0]	7:0	PMUX Packet Metadata[7:0]	TLP Sequence Number[7:0]
4 to N-5	Packet	7:0	PMUX Packet	TLP
N-4 to N-1	LCRC	7:0	PMUX LCRC	PCI Express LCRC

Table G-5 describes the encodings of Symbol 1 bits [6:3] in more detail. If these bits contain a value less than 1001b, the packet is a TLP and is processed as described in Section 4.2.2 .[178] If these bits contain 1001b, 1010b, or 1011b, the encoding is reserved for future standardization and is processed as described in Section 4.2.2.3.3 . If these bits contain a value greater than or equal to 1100b, the packet is a PMUX Packet is defined as specified in this appendix.[179]

Table G-5 Symbol 1 Bits [6:3]

Symbol 1 bits [6:3]	Meaning
0xxxb or 1000b	Packet is a TLP. Bits [6:3] are TLP Length [10:7].
1001b, 1010b, or 1011b	Encoding reserved for future standardization. Receivers detecting these encodings shall process them as described in Section 4.2.2.3.3 .
1100b	Packet is a PMUX Packet. PMUX Channel ID is 0.
1101b	Packet is a PMUX Packet. PMUX Channel ID is 1.
1110b	Packet is a PMUX Packet. PMUX Channel ID is 2.
1111b	Packet is a PMUX Packet. PMUX Channel ID is 3.

For PMUX Packets, the packet length in DWORDs is contained in PMUX Packet Length [6:0]. Other than being a smaller field, PMUX Packet Length is interpreted in the same manner as TLP Length. Specifically, PMUX Packet Length also includes the framing and PMUX LCRC DWORDs (see Section 4.2.2.2).

For PMUX Packets, symbols 2 and 3 contain PMUX Packet Metadata in the same bit positions that TLPs use for TLP Sequence Number.

178. The value 1001b supports a maximum TLP Length [10:0] value of 1151 DWORDs (decimal).This will accommodate a TLP consisting of 4096 bytes of payload, 16 bytes of TLP Header, 4 bytes of TLP digest, and 480 bytes of TLP Prefix.
179. The value 1100b was chosen to simplify distinguishing PMUX Packets from TLPs and from the reserved encodings.

The PMUX LCRC algorithm is identical to the TLP LCRC algorithm as described in Section 3.6.2 with the following modifications:

- The seed value is FB3E E248h (TLP LCRC uses FFFF FFFFh).
- The PMUX Channel ID field in Symbol 1 bits 4:3 is used to compute a 4 bit value that is included in the PMUX LCRC in the same manner as the 4 reserved bits in the TLP LCRC. This 4 bit value contains the value that would be used, by the 8b/10b encoding, for Symbol 1 bits 7:4. Specifically, the lower 2 bits of this 4 bit value contain the PMUX Channel ID and the upper 2 bits contain the inverse (1s complement) of the PMUX Channel ID.
- The PMUX Packet Metadata field is included in the PMUX LCRC in the same manner as the TLP Sequence Number field is included in the TLP LCRC.

The Frame CRC and Frame Parity fields are computed as shown below. This is the same algorithm computed over the same bit positions as defined in Section 4.2.2.2 .

```
C[0] = 1b ^       PMUX_Channel_ID[0]
                ^ L[6] ^ L[4] ^ L[2] ^ L[1] ^ L[0]
C[1] = 1b ^ 1b ^ PMUX_Channel_ID[0]
                ^ L[5] ^ L[4] ^ L[3] ^ L[2]
C[2] = 1b ^       PMUX_Channel_ID[1]
                ^ L[6] ^ L[4] ^ L[3] ^ L[2] ^ L[1]
C[3] =            PMUX_Channel_ID[1]
                ^ PMUX_Channel_ID[0]
                ^ L[5] ^ L[3] ^ L[2] ^ L[1] ^ L[0]
P    = 1b ^ 1b ^ PMUX_Channel_ID[1]
                ^ PMUX_Channel_ID[0]
                ^ L[6] ^ L[5] ^ L[4] ^ L[3] ^ L[2] ^ L[1] ^ L[0]
                ^ C[3] ^ C[2] ^ C[1] ^ C[0]
```

IMPLEMENTATION NOTE

PMUX Channel ID and Frame CRC

When using 128b/130b encoding, the PMUX Channel ID field is covered by the Frame CRC and Frame Parity fields. As such, receivers may make decisions based on the PMUX Channel ID value as soon as the Frame CRC and Frame Parity is checked and need not wait until the PMUX LCRC is checked.

Note: The PMUX Channel ID is also covered by the LCRC. The value of the LCRC associated with a given PMUX Packet is independent of the encoding used to transmit the packet.

G.4 PMUX Control

Protocol Multiplexing is disabled by default. Each PMUX Channel must be explicitly enabled by software at each end of the associated Link. Protocol Multiplexing is disabled whenever the link drops (Data Link Layer indicates DL_Down).

A component that supports Protocol Multiplexing indicates such by the presence of the PMUX Extended Capability.

The following rules apply to components that support Protocol Multiplexing:

- PMUX Packets received in a PMUX Channel that is not enabled are silently ignored.

- PMUX Packets may not be transmitted unless the associated PMUX Channel is enabled. A PMUX Channel may also require additional, protocol specific, initialization mechanisms before PMUX Packets may be transmitted.

G.5 PMUX Extended Capability

Figure G-7 shows the PMUX Extended Capability structure. The presence of this capability indicates that the Port supports the optional Protocol Multiplexing mechanism. This capability is optional and may be present in any Downstream Port and in Function 0 of any Upstream Port. It must not be present in non-zero Functions of Upstream Ports or in RCRBs.

The length of the PMUX Extended Capability is determined by the PMUX Protocol Array Size field (see Section G.5.2).

This capability contains a list of the protocols supported by the Link (the PMUX Protocol Array). It also contains the mechanism software uses to enable and configure PMUX Channels. This capability must be present in both the Upstream and Downstream Ports of a Link in order for Protocol Multiplexing to be successfully enabled.

Software may enable the Upstream and Downstream Ports of a Link in either order. Software may enable multiple PMUX Channels using a single write to the PMUX Control Register.

Behavior is undefined if software enables Protocol Multiplexing in one Port and the other Port of the Link does not support Protocol Multiplexing. Behavior is also undefined if software configures a PMUX Channel inconsistently (the same PMUX Channel in the Ports on each end of a Link configured with incompatible protocols).

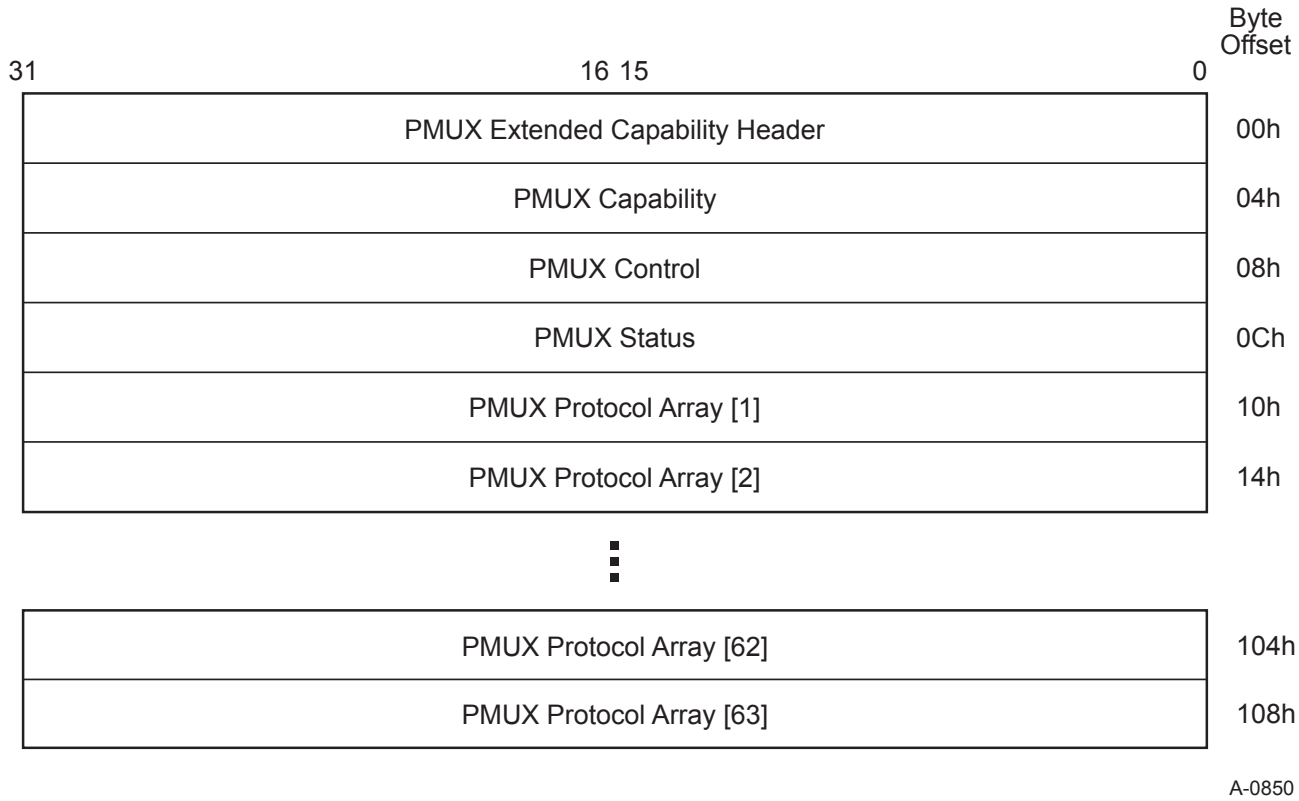


Figure G-7 PMUX Extended Capability

G.5.1 *PMUX Extended Capability Header* (Offset 00h)

Figure G-8 details the allocation of fields in the PMUX Extended Capability header; Table G-6 provides the respective bit definitions.

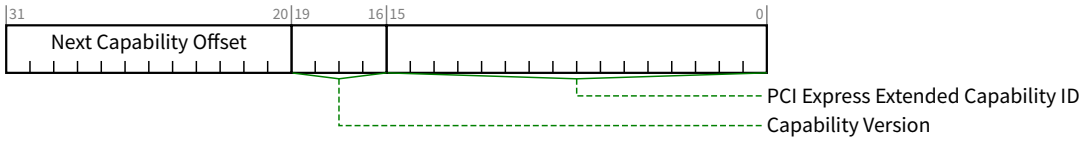


Figure G-8 PMUX Extended Capability Header

Table G-6 PMUX Extended Capability Header

Bit Location	Register Description	Attributes
15:0	***PCI Express Extended Capability ID*** - This field is a PCI-SIG defined ID number that indicates the nature and format of the Extended Capability. The Extended Capability ID for the PMUX Extended Capability is 001Ah.	RO
19:16	***Capability Version*** - This field is a PCI-SIG defined version number that indicates the version of the Capability structure present. Must be 1h for this version of the specification.	RO
31:20	***Next Capability Offset*** - This field contains the offset to the next PCI Express Capability structure or 000h if no other items exist in the linked list of capabilities. This offset is relative to the beginning of PCI compatible Configuration Space and thus must always be either 000h (for terminating the list of Capabilities) or greater than 0FFh.	RO

G.5.2 *PMUX Capability Register* (Offset 04h)

Figure G-9 details the allocation of fields in the PMUX Capability Register. Table G-7 provides the respective bit definitions.

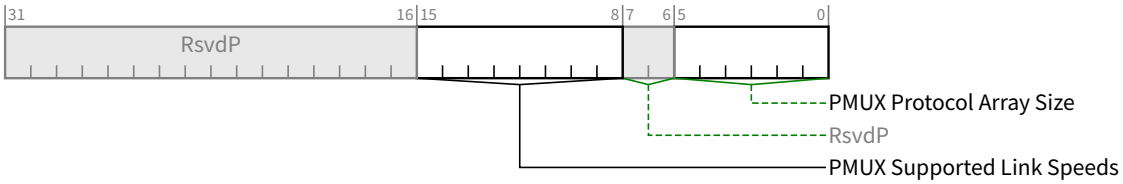


Figure G-9 PMUX Capability Register

Table G-7 PMUX Capability Register

Bit Location	Register Description	Attributes
5:0	***PMUX Protocol Array Size*** - Indicates the size of this Function's PMUX Protocol Array. This field may be 0 to indicate that even though no protocols are supported, the Port will ignore all received PMUX Packets.	RO
15:8	***PMUX Supported Link Speeds*** - This field indicates the Link speed(s) where Protocol Multiplexing is supported. Each bit corresponds to a Link speed. If a bit is Set, Protocol Multiplexing is supported at that Link speed. If a bit is Clear, Protocol Multiplexing is not supported at that Link speed. Bit definitions are: **Bit 8** 2.5 GT/s **Bit 9** 5.0 GT/s **Bit 10** 8.0 GT/s **Bit 11** 16.0 GT/s **Bit 12** 32.0 GT/s **Bits 15:13** RsvdP At least one Link speed must be supported (i.e., the field must be non-zero). A Port may support any combination of Link speeds. For example, this field could contain the value 0000 0100b indicating that Protocol Multiplexing is only supported at 8.0 GT/s. This field must not indicate support for Link speeds that are not supported by the Link (see Section 7.5.3.18). Note that this field indicates the Link speeds supported by Protocol Multiplexing for the Link. The Link speeds that a particular protocol supports and the mechanism used to report that information are protocol specific.	RO / RsvdP

G.5.3 *PMUX Control Register* (Offset 08h)

Figure G-10 details the allocation of fields in the PMUX Control Register. Table G-8 provides the respective bit definitions.

Channel *n* is enabled and available for use by the PMUX Protocol Layer when all of the following are true:

- The Channel *n* Assignment field is non-zero.
- The Channel *n* Assignment field is less than or equal to PMUX Protocol Array Size.
- The Channel *n* Assignment field indicates an implemented entry in the PMUX Protocol Array (see Section G.5.5).
- All of the PMUX Channel *n* Disabled bits are Clear (see Section G.5.4).

Otherwise, Channel *n* is disabled.

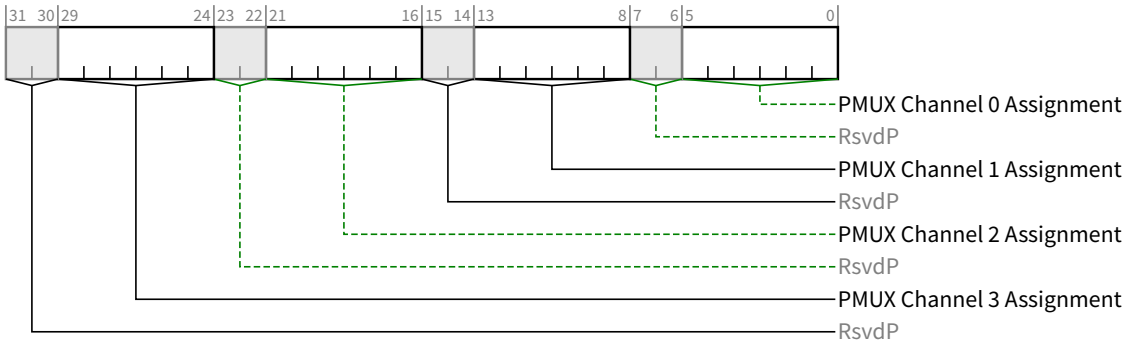


Figure G-10 PMUX Control Register

Table G-8 PMUX Control Register

Bit Location	Register Description	Attributes
5:0	***PMUX Channel 0 Assignment*** - This field indicates the protocol assigned to PMUX Channel 0. If the field is 0h, no protocol is assigned. If the field is non-zero, it is the index in the PMUX Protocol Array of the protocol assigned to PMUX Channel 0. If PMUX Protocol Array Size is less than 63 (see Section G.5.2), unused upper bits of this field may be hardwired to 0b. If PMUX Protocol Array Size is 0, this entire field may be hardwired to 0. This field defaults to 0h.	RW
13:8	***PMUX Channel 1 Assignment*** - This field indicates the protocol assigned to PMUX Channel 1. If the field is 0h, no protocol is assigned. If the field is non-zero, it is the index in the PMUX Protocol Array of the protocol assigned to PMUX Channel 1. If PMUX Protocol Array Size is less than 63 (see Section G.5.2), unused upper bits of this field may be hardwired to 0b. If PMUX Protocol Array Size is 0, this entire field may be hardwired to 0. This field defaults to 0h.	RW
21:16	***PMUX Channel 2 Assignment*** - This field indicates the protocol assigned to PMUX Channel 2. If the field is 0h, no protocol is assigned. If the field is non-zero, it is the index in the PMUX Protocol Array of the protocol assigned to PMUX Channel 2. If PMUX Protocol Array Size is less than 63 (see Section G.5.2), unused upper bits of this field may be hardwired to 0b. If PMUX Protocol Array Size is 0, this entire field may be hardwired to 0. This field defaults to 0h.	RW
29:24	***PMUX Channel 3 Assignment*** - This field indicates the protocol assigned to PMUX Channel 3. If the field is 0h, no protocol is assigned. If the field is non-zero, it is the index in the PMUX Protocol Array of the protocol assigned to PMUX Channel 3. If PMUX Protocol Array Size is less than 63 (see Section G.5.2), unused upper bits of this field may be hardwired to 0b. If PMUX Protocol Array Size is 0, this entire field may be hardwired to 0. This field defaults to 0h.	RW

G.5.4 *PMUX Status Register* (Offset 0Ch)

Figure G-11 details the allocation of fields in the PMUX Status Register. Table G-9 provides the respective bit definitions.

Each channel has a set of Disabled bits. When Channel *n* Assignment field is non-zero, the Channel *n* Disabled bits reflect the error status of the channel. The following Disabled bits are defined:

- PMUX Channel *n* Disabled: Link Speed
- PMUX Channel *n* Disabled: Link Width
- PMUX Channel *n* Disabled: Protocol Specific

When there are multiple reasons for disabling a channel, an implementation may choose which reason(s) to report. For example, if a protocol needs bandwidth equivalent to x1 8.0 GT/s, when there is inadequate bandwidth (e.g., the Link is operating at x1 5.0 GT/s, x1 2.5 GT/s, or x2 2.5 GT/s), it could disable the PMUX Channel by indicating any or all of Disabled: Link Width, Disabled: Link Speed, or Disabled: Protocol Specific.

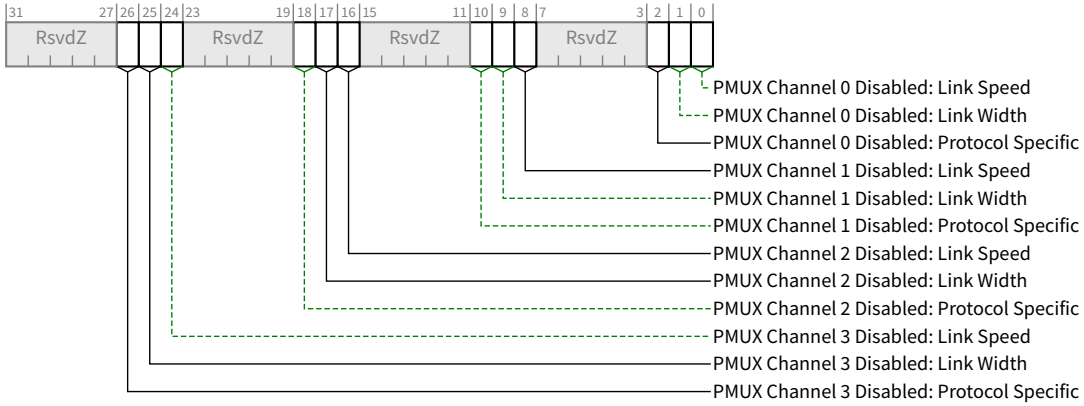


Figure G-11 PMUX Status Register

Table G-9 PMUX Status Register

Bit Location	Register Description	Attributes
0	***PMUX Channel 0 Disabled: Link Speed*** - If Set, Channel 0 is disabled because the Current Link Speed (Section 7.8.8) is not supported by Protocol Multiplexing or by the protocol assigned to Channel 0. This bit is 0 when no protocol is assigned to Channel 0 (i.e., Channel 0 Control field is 0h).	RO
1	***PMUX Channel 0 Disabled: Link Width*** - If Set, Channel 0 is disabled because the current Link Width is not supported by the protocol assigned to Channel 0. This bit is 0 when no protocol is assigned to Channel 0 (i.e., PMUX Channel 0 Assignment field is 0h).	RO
2	***PMUX Channel 0 Disabled: Protocol Specific*** - If Set, Channel 0 is disabled for protocol specific reasons. This bit is 0 when no protocol is assigned to Channel 0 (i.e., PMUX Channel 0 Assignment field is 0h).	RO
8	***PMUX Channel 1 Disabled: Link Speed*** - If Set, Channel 1 is disabled because the Current Link Speed (Section 7.8.8) is not supported by Protocol Multiplexing or by the protocol assigned to Channel 1. This bit is 0 when no protocol is assigned to Channel 1 (i.e., PMUX Channel 1 Assignment field is 0h).	RO
9	***PMUX Channel 1 Disabled: Link Width*** - If Set, Channel 1 is disabled because the current Link Width is not supported by the protocol assigned to Channel 1. This bit is 0 when no protocol is assigned to Channel 1 (i.e. PMUX Channel 1 Assignment field is 0h).	RO

Bit Location	Register Description	Attributes
10	***PMUX Channel 1 Disabled: Protocol Specific*** - If Set, Channel 1 is disabled for protocol specific reasons. This bit is 0 when no protocol is assigned to Channel 1 (i.e., PMUX Channel 1 Assignment field is 0h).	RO
16	***PMUX Channel 2 Disabled: Link Speed*** - If Set, Channel 2 is disabled because the Current Link Speed (Section 7.8.8) is not supported by Protocol Multiplexing or by the assigned protocol. This bit is 0 when no protocol is assigned to Channel 2 (i.e., PMUX Channel 2 Assignment field is 0h).	RO
17	***PMUX Channel 2 Disabled: Link Width*** - If Set, Channel 2 is disabled because the current Link Width is not supported by the assigned protocol. This bit is 0 when no protocol is assigned to Channel 2 (i.e., PMUX Channel 2 Assignment field is 0h).	RO
18	***PMUX Channel 2 Disabled: Protocol Specific*** - If Set, Channel 2 is disabled for protocol specific reasons. This bit is 0 when no protocol is assigned to Channel 2 (i.e., PMUX Channel 2 Assignment field is 0h).	RO
24	***PMUX Channel 3 Disabled: Link Speed*** - If Set, Channel 3 is disabled because the Current Link Speed (Section 7.8.8) is not supported by Protocol Multiplexing or by the assigned protocol. This bit is 0 when no protocol is assigned to Channel 3 (i.e., PMUX Channel 3 Assignment field is 0h).	RO
25	***PMUX Channel 3 Disabled: Link Width*** - If Set, Channel 3 is disabled because the current Link Width is not supported by the assigned protocol. This bit is 0 when no protocol is assigned to Channel 3 (i.e., PMUX Channel 3 Assignment field is 0h).	RO
26	***PMUX Channel 3 Disabled: Protocol Specific*** - If Set, Channel 3 is disabled for protocol specific reasons. This bit is 0 when no protocol is assigned to Channel 3 (i.e., PMUX Channel 3 Assignment field is 0h).	RO

G.5.5 *PMUX Protocol Array* (Offsets 10h through 48h)

The PMUX Protocol Array consists of up to 63 entries. The size of the PMUX Protocol Array is indicated by the PMUX Protocol Array Size field (see Section G.5.2).

Figure G-12 details the allocation of fields in each PMUX Protocol Array Entry. Table G-10 provides the respective bit definitions.

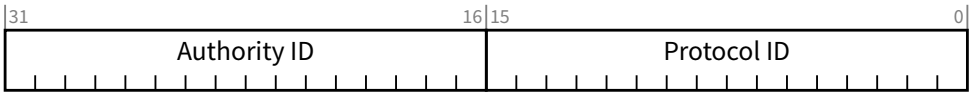


Figure G-12 PMUX Protocol Array Entry

Table G-10 PMUX Protocol Array Entry

Bit Location	Register Description	Attributes
15:0	***Protocol ID*** - In conjunction with Authority ID designates a specific protocol and the mechanism by which that protocol is mapped onto Protocol Multiplexing. This value is assigned by the Vendor associated with the Authority ID field.	RO

Bit Location	Register Description	Attributes
31:16	***Authority ID*** - Designates the authority controlling the values used in the Protocol ID field. The Authority ID field contains a Vendor ID as assigned by the PCI-SIG.	RO

The value 0000 0000h indicates an unimplemented PMUX Protocol Array Entry. The PMUX Protocol Array is indexed starting at 1.

PMUX Channel *n* is enabled and configured to support the protocol associated with PMUX Protocol Array Entry at index *m* when the PMUX Channel *n* Assignment field contains the value *m* (see Section G.5.3).

Entries in the PMUX Protocol Array with the Authority ID value 1 (0001h) represent protocols that are defined by the PCI-SIG.

Duplicate Entries in the PMUX Protocol Array may be used to represent multiple instances of a particular protocol. This permits software control of the mapping between PMUX Channel ID and a specific instance of a protocol.

IMPLEMENTATION NOTE
Multiple Protocol Instances

A Link may have a single PMUX Protocol assigned to multiple PMUX Channels. Each PMUX Channel is assigned to a different instance of the protocol. Each instance of a protocol corresponds to an entry in the PMUX Protocol Array.

Consider a Port that supports two instances of protocol X. Two entries in the PMUX Protocol Array would indicate protocol X (indexes A and B for example). To assign instance A to PMUX Channel 0 and instance B to PMUX Channel 2, place the value A in the PMUX Channel 0 Assignment field and the value B in the PMUX Channel 2 Assignment field.

Flow Control Update Latency and ACK Update Latency Calculations

This appendix is informational only and should not be considered normative. Earlier revisions of this specification outlined the method used to calculate the Flow Control Update Latency and Ack Update Latency. This appendix describes the method used to derive the values and preserves the UpdateFactor and AckFactor values.

H.1 Flow Control Update Latency

Recommended Flow Control Update Latency is described in the Implementation Note in Section 2.6.1.2 entitled, FC Information Tracked by Receiver.

Flow Control Update Latency tables were simplified in the 4.0 Revision of this specification. At that time, the original tables were moved to this appendix. Note that in the 4.0 Revision of this specification, Tables 2-47 and 2-48 were distinct, but identical tables. The 5.0 Revision contains a single table that applies to 8.0 GT/s or higher.

Original Tables		Simplified Tables	
Base 3.1	Base 4.0 or later	Base 4.0	Base 5.0 or later
Table 2-42	Table H-1	Table 2-45	Table 2-46
Table 2-43	Table H-2	Table 2-46	Table 2-47
Table 2-44	Table H-3	Table 2-47	Table 2-48
n/a	n/a	Table 2-48	

The values in the Tables are measured starting from when the Receiver Transaction Layer makes additional receive buffer space available by processing a received TLP, to when the first Symbol of the corresponding UpdateFC DLLP is transmitted. The values are calculated as a function of the largest TLP payload size and Link width using the formula:

$$\frac{(\text{Max\_Payload\_Size} + \text{TLPOverhead}) \times \text{UpdateFactor}}{\text{LinkWidth}} + \text{InternalDelay}$$

Equation H-1 Max UpdateFC Latency

where:

Max_Payload_Size
The value in the Max_Payload_Size field of the Device Control register. For a Multi-Function Device, it is recommended that the smallest Max_Payload_Size setting across all Functions[180] is used.

TLPOverhead
Represents the additional TLP components which consume Link bandwidth (TLP Prefix, header, LCRC, framing Symbols) and is treated here as a constant value of 28 Symbols.

180. For ARI Devices, the Max_Payload_Size is determined solely by the setting in Function 0, and thus the settings in the other Functions should be ignored.

UpdateFactor

Represents the number of maximum size TLPs sent during the time between UpdateFC receptions, and is used to balance Link bandwidth efficiency and receive buffer sizes - the value varies according to Max_Payload_Size and Link width. Table H-1 , Table H-2 , and Table H-3 below include the UpdateFactor(UF).

LinkWidth

The operating width of the Link

InternalDelay

Represents the internal processing delays for received TLPs and transmitted DLLPs, and is treated here as a constant value of 19 Symbol Times for 2.5 GT/s mode operation, 70 Symbol Times for 5.0 GT/s mode operation, and 115 Symbol Times for 8.0 GT/s and 16.0 GT/s modes of operation.

Table H-1 Maximum UpdateFC Transmission Latency Guidelines for 2.5 GT/s Mode Operation by Link Width and Max Payload (Symbol Times)

		Link Operating Width						
		x1	x2	x4	x8	x12	x16	x32
Max_Payload_Size (bytes)	128	237 UF = 1.4	128 UF = 1.4	73 UF = 1.4	67 UF = 2.5	58 UF = 3.0	48 UF = 3.0	33 UF = 3.0
	256	416 UF = 1.4	217 UF = 1.4	118 UF = 1.4	107 UF = 2.5	90 UF = 3.0	72 UF = 3.0	45 UF = 3.0
	512	559 UF = 1.0	289 UF = 1.0	154 UF = 1.0	86 UF = 1.0	109 UF = 2.0	86 UF = 2.0	52 UF = 2.0
	1024	1071 UF = 1.0	545 UF = 1.0	282 UF = 1.0	150 UF = 1.0	194 UF = 2.0	150 UF = 2.0	84 UF = 2.0
	2048	2095 UF = 1.0	1057 UF = 1.0	538 UF = 1.0	278 UF = 1.0	365 UF = 2.0	278 UF = 2.0	148 UF = 2.0
	4096	4143 UF = 1.0	2081 UF = 1.0	1050 UF = 1.0	534 UF = 1.0	706 UF = 2.0	534 UF = 2.0	276 UF = 2.0

Table H-2 Maximum UpdateFC Transmission Latency Guidelines for 5.0 GT/s Mode Operation by Link Width and Max Payload (Symbol Times)

		Link Operating Width						
		x1	x2	x4	x8	x12	x16	x32
Max_Payload_Size (bytes)	128	288 UF = 1.4	179 UF = 1.4	124 UF = 1.4	118 UF = 2.5	109 UF = 3.0	99 UF = 3.0	84 UF = 3.0
	256	467 UF = 1.4	268 UF = 1.4	169 UF = 1.4	158 UF = 2.5	141 UF = 3.0	123 UF = 3.0	96 UF = 3.0
	512	610 UF = 1.0	340 UF = 1.0	205 UF = 1.0	137 UF = 1.0	160 UF = 2.0	137 UF = 2.0	103 UF = 2.0
	1024	1122 UF = 1.0	596 UF = 1.0	333 UF = 1.0	201 UF = 1.0	245 UF = 2.0	201 UF = 2.0	135 UF = 2.0
	2048	2146 UF = 1.0	1108 UF = 1.0	589 UF = 1.0	329 UF = 1.0	416 UF = 2.0	329 UF = 2.0	199 UF = 2.0

		Link Operating Width						
		x1	x2	x4	x8	x12	x16	x32
	4096	4194 UF = 1.0	2132 UF = 1.0	1101 UF = 1.0	585 UF = 1.0	757 UF = 2.0	585 UF = 2.0	327 UF = 2.0

Table H-3 Maximum UpdateFC Transmission Latency Guidelines for 8.0 GT/s Operation by Link Width and Max Payload (Symbol Times)

		Link Operating Width						
		x1	x2	x4	x8	x12	x16	x32
Max_Payload_Size (bytes)	128	333 UF = 1.4	224 UF = 1.4	169 UF = 1.4	163 UF = 2.5	154 UF = 3.0	144 UF = 3.0	129 UF = 3.0
	256	512 UF = 1.4	313 UF = 1.4	214 UF = 1.4	203 UF = 2.5	186 UF = 3.0	168 UF = 3.0	141 UF = 3.0
	512	655 UF = 1.0	385 UF = 1.0	250 UF = 1.0	182 UF = 1.0	205 UF = 2.0	182 UF = 2.0	148 UF = 2.0
	1024	1167 UF = 1.0	641 UF = 1.0	378 UF = 1.0	246 UF = 1.0	290 UF = 2.0	246 UF = 2.0	180 UF = 2.0
	2048	2191 UF = 1.0	1153 UF = 1.0	634 UF = 1.0	374 UF = 1.0	461 UF = 2.0	374 UF = 2.0	244 UF = 2.0
	4096	4239 UF = 1.0	2177 UF = 1.0	1146 UF = 1.0	630 UF = 1.0	802 UF = 2.0	630 UF = 2.0	372 UF = 2.0

H.2 Ack Latency

Ack Latency tables were simplified in the 4.0 Revision of this specification. At that time, the original tables were moved to this appendix. Note that in the 4.0 Revision of this specification, Tables 3-9 and 3-10 were distinct, but identical tables. The 5.0 Revision contains a single table that applies to 8.0 GT/s or higher.

Original Tables		Simplified Tables	
Base 3.1	Base 4.0 or later	Base 4.0	Base 5.0 or later
Table 3-7	Table H-4	Table 3-7	Table 3-7
Table 3-8	Table H-5	Table 3-8	Table 3-8
Table 3-9	Table H-6	Table 3-9	Table 3-9
n/a	n/a	Table 3-10	

The Maximum Ack Latency limits are calculated using the formula:

$$\frac{(\text{Max\_Payload\_Size}+\text{TLPOverhead}) \times \text{UpdateFactor}}{\text{LinkWidth}} + \text{InternalDelay}$$

Equation H-2 Max Ack Latency

where:

Max_Payload_Size
is the value in the Max_Payload_Size field of the Device Control register. For ARI Devices, the Max_Payload_Size is determined solely by the setting in Function 0. For a non-ARI Multi-Function Device whose Max_Payload_Size settings are identical across all Functions, the common Max_Payload_Size setting must be used. For a non-ARI Multi-Function Device whose Max_Payload_Size settings are not identical across all Functions, the selected Max_Payload_Size setting is implementation specific, but it is recommended to use the smallest Max_Payload_Size setting across all Functions.

TLPOverhead
represents the additional TLP components which consume Link bandwidth (TLP Prefix, header, LCRC, framing Symbols) and is treated here as a constant value of 28 Symbols.

AckFactor
represents the number of maximum size TLPs which can be received before an Ack is sent, and is used to balance Link bandwidth efficiency and retry buffer size - the value varies according to Max_Payload_Size and Link width. Table H-4, Table H-5, and Table H-6 below include the AckFactor(AF).

LinkWidth
is the operating width of the Link.

InternalDelay
represents the internal processing delays for received TLPs and transmitted DLLPs, and is treated here as a constant value of 19 Symbol Times for 2.5 GT/s mode operation, 70 Symbol Times for 5.0 GT/s operation, and 115 Symbol Times for 8.0 GT/s and 16.0 GT/s operation.

Table H-4 Maximum Ack Latency Limit and AckFactor for 2.5 GT/s (Symbol Times)

		Link Operating Width						
		x1	x2	x4	x8	x12	x16	x32
Max_Payload_Size (bytes)	128	237 AF = 1.4	128 AF = 1.4	73 AF = 1.4	67 AF = 2.5	58 AF = 3.0	48 AF = 3.0	33 AF = 3.0
	256	416 AF = 1.4	217 AF = 1.4	118 AF = 1.4	107 AF = 2.5	90 AF = 3.0	72 AF = 3.0	45 AF = 3.0
	512	559 AF = 1.0	289 AF = 1.0	154 AF = 1.0	86 AF = 1.0	109 AF = 2.0	86 AF = 2.0	52 AF = 2.0
	1024	1071 AF = 1.0	545 AF = 1.0	282 AF = 1.0	150 AF = 1.0	194 AF = 2.0	150 AF = 2.0	84 AF = 2.0
	2048	2095 AF = 1.0	1057 AF = 1.0	538 AF = 1.0	278 AF = 1.0	365 AF = 2.0	278 AF = 2.0	148 AF = 2.0
	4096	4143 AF = 1.0	2081 AF = 1.0	1050 AF = 1.0	534 AF = 1.0	706 AF = 2.0	534 AF = 2.0	276 AF = 2.0

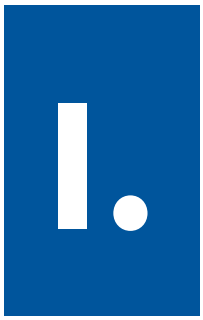
Table H-5 Maximum Ack Transmission Latency Limit and AckFactor for 5.0 GT/s (Symbol Times)

		Link Operating Width						
		x1	x2	x4	x8	x12	x16	x32
Max_Payload_Size (bytes)	128	288 AF = 1.4	179 AF = 1.4	124 AF = 1.4	118 AF = 2.5	109 AF = 3.0	99 AF = 3.0	84 AF = 3.0
	256	467 AF = 1.4	268 AF = 1.4	169 AF = 1.4	158 AF = 2.5	141 AF = 3.0	123 AF = 3.0	96 AF = 3.0
	512	610 AF = 1.0	340 AF = 1.0	205 AF = 1.0	137 AF = 1.0	160 AF = 2.0	137 AF = 2.0	103 AF = 2.0
	1024	1122 AF = 1.0	596 AF = 1.0	333 AF = 1.0	201 AF = 1.0	245 AF = 2.0	201 AF = 2.0	135 AF = 2.0
	2048	2146 AF = 1.0	1108 AF = 1.0	589 AF = 1.0	329 AF = 1.0	416 AF = 2.0	329 AF = 2.0	199 AF = 2.0
	4096	4194 AF = 1.0	2132 AF = 1.0	1101 AF = 1.0	585 AF = 1.0	757 AF = 2.0	585 AF = 2.0	327 AF = 2.0

Table H-6 Maximum Ack Transmission Latency Limit and AckFactor for 8.0 GT/s (Symbol Times)

		Link Operating Width						
		x1	x2	x4	x8	x12	x16	x32
Max_Payload_Size (bytes)	128	333 AF = 1.4	224 AF = 1.4	169 AF = 1.4	163 AF = 2.5	154 AF = 3.0	144 AF = 3.0	129 AF = 3.0
	256	512 AF = 1.4	313 AF = 1.4	214 AF = 1.4	203 AF = 2.5	186 AF = 3.0	168 AF = 3.0	141 AF = 3.0
	512	655 AF = 1.0	385 AF = 1.0	250 AF = 1.0	182 AF = 1.0	205 AF = 2.0	182 AF = 2.0	148 AF = 2.0
	1024	1167 AF = 1.0	641 AF = 1.0	378 AF = 1.0	246 AF = 1.0	290 AF = 2.0	246 AF = 2.0	180 AF = 2.0
	2048	2191 AF = 1.0	1153 AF = 1.0	634 AF = 1.0	374 AF = 1.0	461 AF = 2.0	374 AF = 2.0	244 AF = 2.0
	4096	4239 AF = 1.0	2177 AF = 1.0	1146 AF = 1.0	630 AF = 1.0	802 AF = 2.0	630 AF = 2.0	372 AF = 2.0

Async Hot-Plug Reference Model



This appendix presents a recommended reference model for async hot-plug. The reference model covers three areas:

- Async hot-plug initial configuration
- Async removal configuration and interrupt handling
- Async hot-add configuration and interrupt handling

There are no normative requirements in this section. The entire reference model is contained in a series of implementation notes. The reference model documents how the various hot-plug mechanisms are envisioned be used to implement a robust asynchronous hot-plug model, but does not mandate that they be used that way.

For brevity and readability, this section uses the following acronyms:

DLL
Data Link Layer

DSP
Downstream Port

FWF
firmware first

HPS
Hot-Plug Surprise

OOB
out-of-band

OS
operating system

PD
presence detect

SFW
system firmware

The reference model covers both the HPS and DPC mechanisms. DPC is the recommended mechanism. The reference model covers FWF for DPC but not for HPS.

The reference model provides a recommended framework for DPC software to support Containment Error Recovery (CER) along with async hot-plug. The reference model does not explicitly cover CER outside of async hot-plug, but certain aspects can be leveraged for DPC support of CER when async hot-plug is not being used.

SFW may use the System Firmware Intermediary (SFI) Capability for async hot-plug, orderly removal hot-plug, or other operations. This reference model does not rely on its functionality nor cover its use.

The reference model refers to various bits or fields outlined in Section 6.7.2 . For brevity, pointers to the associated registers are not replicated in this section.

IMPLEMENTATION NOTE

In-band Presence Detect Mechanism Deprecated for Async Hot-Plug

Due to architectural issues, the in-band (Physical-Layer-based) portion of the PD mechanism is deprecated for use with async hot-plug. One issue is that in-band PD as architected does not detect adapter removal during certain LTSSM states, notably the L1 and Disabled States. Another issue is that when both in-band and OOB PD are being used together, the Presence Detect State bit and its associated interrupt mechanism always reflect the logical OR of the inband and OOB PD states, and with some hot-plug hardware configurations, it is important for software to detect and respond to in-band and OOB PD events independently.

If OOB PD is being used and the associated DSP supports In-Band PD Disable, it is recommended that the In-Band PD Disable bit be Set, and the Presence Detect State bit and its associated interrupt mechanism be used exclusively for OOB PD.

As a substitute for in-band PD with async hot-plug, the reference model uses either the DPC or the DLL Link Active mechanism.

The reference model assumes and covers the configurations listed below. While these cover the bulk of the envisioned use cases, many minor variations are not explicitly covered. For example, there are multiple ways to determine if a slot supports OOB PD, and the reference model does not cover them. As another example, the reference model refers to adding or removing an adapter, but some hot-pluggable modules may include a Switch and multiple Endpoint components.

- Reference model assumptions:
  - DSPs support DLL Link Active Reporting
  - DSPs support In-Band PD Disable
  - Operating systems support both HPS and DPC, using DPC if available
- Reference model covers:
  - Slots that support HPS and/or DPC (SW never enables both at same time)
  - Slots with or without OOB PD
  - RPs with or without RP Extensions for DPC

I.1 Async Hot-Plug Initial Configuration

IMPLEMENTATION NOTE

Async Hot-Plug Initial Configuration

Basic Steps	HPS	DPC
Determine capability control entity	• OS requests, and is granted control of PCIe Native Hot-Plug • If FWF, SFW retains control of AER and DPC • Else OS requests and is granted control of AER and DPC	
OS and SFW determine which async hot-plug mechanism to use; OS/SFW interactions here are outside scope of this specification	• If DPC capability then ◦ If HPS bit not Set, use DPC ◦ Else attempt to Clear HPS bit (Section 6.7.4.4) ▪ If successful, use DPC ▪ Else use HPS • Else if HPS bit Set, use HPS • Else async hot-plug cannot be supported by this slot • Configure DSP for selected mechanism	
OS determines if adapter present	• If OOB PD supported, use it to determine if adapter is present ◦ Set In-band PD Disable bit (SFW may have it Set by default) ◦ Read PD State bit • Else allow Link to attempt to train; use DLL Link Active to determine if adapter is present (and at least minimally functional)	
If adapter is present, OS waits for adapter to become ready for configuration	• If using Device Readiness Status (DRS), begin configuration if/after DSP receives a DRS Message • If using CRS Software Visibility, can attempt first Configuration Read (of Vendor ID field) after 100 ms • Otherwise, must wait at least 1000 ms before attempting first Configuration Read	
OS configures for appropriate case	• If adapter is present, configure system for handling async removal • Else configure system for handling async hot-add	

I.2 Async Removal Configuration and Interrupt Handling

IMPLEMENTATION NOTE
Async Removal Configuration

Basic Steps	HPS	DPC
OS/SFW configure appropriate error handling	• If OS and driver support a non-CER error recovery approach, its policies may influence some of these error settings • Configure the error handling supported by HPS	• If OS and driver support CER, its policies may influence some of these error settings • Enable uncorrectable AER errors to trigger DPC, including Surprise Down • If using RP Extensions for DPC, configure RP PIO error handling • Configure RP Completion Timeout handling per platform and OS policies
OS/SFW configure async removal interrupts	• OS enables DLL State Changed interrupt	• If FWF, configure DPC for ERR_COR signaling to enable SFW handling • Else configure DPC for interrupt to enable OS handling
	• If OOB PD supported, OS enables PD Changed interrupt	

IMPLEMENTATION NOTE

Async Removal Interrupt Handling

Basic Steps	HPS	DPC
Service routine entry	• If PD Changed or DLL State Change, OS is interrupted	• If PD Changed, OS is interrupted ◦ If FWF, OS invokes SFW; preferably via OS Setting DPC Software Trigger bit (if implemented)
		• If DPC triggered ◦ If FWF, DPC sends ERR_COR to signal SFW ◦ Else DPC interrupts OS
Prevent further Link activity	• OS Sets Disable Link; Link goes down and stays down • OS polls DLL Link Active until Clear	• DPC automatically keeps Link down
Log errors from DSP's AER/DPC		• If FWF, SFW logs DSP errors ◦ SFW invokes OS and exits
	• OS logs and then clears DSP errors	
OS notifies impacted software/driver, which cease accessing the adapter		
OS determines if adapter is still present	• Some FFs with OOB PD automatically power off slot and/or disable switched signals • If OOB PD supported, use it to determine if adapter is physically present • If adapter not determined to be absent, re-enable the Link and slot ◦ As applicable, Clear Disable Link or release DPC ◦ Wait until Link trains or adapter is deemed absent or non-functional ◦ If non-functional, optionally perform a slot reset (if supported) ◦ If still non-functional, optionally power-cycle the slot (if supported)	
If OS determined adapter to be present	• OS waits for adapter to become ready for configuration • OS enumerates and configures the adapter • If DPC and FWF, OS invokes SFW to log adapter AER errors	

Basic Steps	HPS	DPC

	• OS logs adapter AER errors and then clears them
	• If OS determines the adapter is suitable for continued operation
	  ◦ OS configures for async removal handling
	  ◦ OS resumes driver
	• Else OS takes OS-specific action	

Else (adapter not present)	• OS ensures DSP is in a clean state ready for a new/different adapter • OS configures for async hot-add handling	

I.3 Async Hot-Add Configuration and Interrupt Handling

IMPLEMENTATION NOTE
Async Hot-Add Configuration

Basic Steps	HPS	DPC
OS configures for async hot-add handling • Enable Link to train if/when an adapter is inserted ◦ E.g., Clear Disable Link or release DPC if needed • If appropriate for form factor, enable power controller prior to adapter insertion • If OOB PD supported, enable OS interrupt on PD Changed • Else enable OS interrupt on DLL State Changed		

IMPLEMENTATION NOTE

Async Hot-Add Interrupt Handling

Basic Steps	HPS	DPC
OS is interrupted due to PD Changed or DLL State Changed		
OS waits for the adapter to become ready for configuration • If appropriate for form factor, enable power controller now (following adapter insertion) • Wait for DLL Link Active • Wait for adapter to become ready for configuration		
OS configures adapter per standard OS conventions		
OS configures for async removal handling		
OS calls driver to complete adapter configuration and begin normal operation		

Acknowledgements

The following persons were instrumental in the development of this specification:[181]

Name	Affiliation
Chamath Abhayagunawardhana	Intel Corporation
Ryan Abraham	Oracle Corporation
Shiva Aditham	Intel Corporation
Hong Ahn	Xilinx
Shay Aisman	Mellanox Technologies, Ltd.
Jasmin Ajanovic	Intel Corporation
Katsutoshi Akagi	NEC Corporation
Joe Allen	Tektronix, Inc.
Michael W. Altmann	Intel Corporation
Taha Amiralli	Advanced Micro Devices, Inc.
Boon Ang	VMware Corporation
Sujith Arramreddy	ServerWorks, Inc.
Antonio Asaro	Advanced Micro Devices, Inc.
Yuval Avnon	Marvell Semiconductor, Inc.
Jay Avula	ServerWorks, Inc.
Pervez Aziz	NVIDIA Corporation
Brian Baldwin	Synopsys, Inc.
Jasper Balraj	Intel Corporation
Nat Barbiero	Advanced Micro Devices, Inc.
Phillip Barnes	Hewlett-Packard Company
Suparna Behera	LSI Logic Corporation
Joseph A. Bennett	Intel Corporation
Richard Bell	Advanced Micro Devices, Inc.
Stuart Berke	Hewlett-Packard Company
Harish Bharadwaj	LSI Logic Corporation
Ajay V. Bhatt	Intel Corporation

181. Company affiliation listed is at the time of specification contributions.

Name	Affiliation
Gaurav Bhide	Advanced Micro Devices, Inc.
Harmeet Bhugra	IDT Corporation
Christiaan Bil	Intel Corporation
Bill Bissonette	Intel Corporation
Cass Blodget	Intel Corporation
Jeffrey D. Bloom	Intel Corporation
Mahesh Bohra	IBM Corporation
Naveen Bohra	Intel Corporation
Karl Bois	Hewlett Packard Enterprise
Austin Bolen	Dell Inc.
Jerry Bolen	Intel Corporation
Michele Bologna	Synopsys, Inc.
Wil de Bont	National Instruments
David Bouse	Intel Corporation
Hicham Bouzekri	ST-Ericsson
Gavin Bowlby	Marvell Semiconductor, Inc
Suri Brahmaroutu	Dell Inc.
Dave Brown	Integrated Device Technology
Tory Brown	Intel Corporation
Mike Brownell	Intel Corporation
Bala Cadambi	Intel Corporation
John Calvin	VTM
John Calvin	Tektronix, Inc.
Jun Cao	Synopsys, Inc.
Gord Caruk	Advanced Micro Devices, Inc.
Paul Cassidy	Synopsys, Inc.
James Chapple	Intel Corporation
Kabitha Chaturvedula	LSI Logic Corporation
Yogesh Chaudhary	Mentor Graphics
Santanu Chaudhuri	Intel Corporation
Rajinder Cheema	LSI Logic Corporation

Name	Affiliation
Albert Chen	LSI Logic Corporation
Chih-Cheh Chen	Intel Corporation
Jian Chen	Parade
Qunwei Chen	Advanced Micro Devices, Inc.
Tony Chen	Marvell Semiconductor, Inc
Jonathan Cheung	Advanced Micro Devices, Inc.
Yorick Cho	Advanced Micro Devices, Inc.
Dickson Chow	Advanced Micro Devices, Inc.
Gene Chui	IDT Corporation
Shawn Clayton	Emulex Corporation
Mark Clements	IBM Corporation
Debra Cohen	Intel Corporation
Eric Combs	LSI Logic Corporation
Brad Congdon	Intel Corporation
Mike Converse	IDT Corporation
Justin Coppin	Hewlett-Packard Company
Joe Cowan	Hewlett-Packard Enterprise
Carrie Cox	IBM Corporation
H. Clay Cranford	IBM Corporation
Kenneth C. Creta	Intel Corporation
Pamela Cristo	Emulex Corporation
Sanjay Dabral	Intel Corporation
Eric Dahlen	Intel Corporation
Tugrul Daim	Intel Corporation
Ron Dammann	Intel Corporation
Sumit Das	Texas Instruments Incorporated
Debendra Das Sharma	Intel Corporation
Nicole Daugherty	Intel Corporation
Glen Dearth	Advanced Micro Devices, Inc.
Eric DeHaemer	Intel Corporation
Fei Deng	Intel Corporation

Name	Affiliation
Dan DeSetto	Intel Corporation
Karishma Dhruv	LSI Logic Corporation
Gary Dick	Cadence Design Systems, Inc.
Robert Dickson	Oracle Corporation
Bob Divivier	IDT Corporation
Hormoz Djahanshahi	Microsemi Corporation
Daniel Dreps	IBM Corporation
Ken Drottar	Intel Corporation
Dave Dunning	Intel Corporation
Rick Eads	Keysight
Rick Eads	Agilent Technologies, Inc.
Gregory L. Ebert	Intel Corporation
Imtinan Elahi	NVIDIA Corporation
Yaron Elboim	Intel Corporation
Bassam Elkhoury	Intel Corporation
Salem Emara	Advanced Micro Devices, Inc.
Mike Engbretson	Tektronix, Inc.
Ohad Falik	Intel Corporation
David Fair	Intel Corporation
Blaise Fanning	Intel Corporation
John Feehrer	Oracle Corporation
Wes Ficken	GLOBALFOUNDRIES Inc.
Joshua Filliater	LSI Logic Corporation
Michael Fleischer-Reumann	Agilent Technologies, Inc.
Pelle Fornberg	Intel Corporation
Jeff Fose	Emulex Corporation
Jim Foster	IBM Corporation
Mike Foxcroft	Advanced Micro Devices, Inc.
Douglas Freimuth	IBM Corporation
Daniel S. Froelich	Intel Corporation
SivaPrasad Gadey	Intel Corporation

Name	Affiliation
Yoni Galezer	Mellanox Technologies, Ltd.
Eric Geisler	Intel Corporation
Gordon Getty	Agilent Technologies, Inc.
Gregory Geyfman	Intel Corporation
John Geldman	Toshiba Memory Corporation
Stefano Giaconi	Intel Corporation
Steve Glaser	NVIDIA Corporation
Thorsten Goetzelmann	Keysight
Marc A. Goldschmidt	Intel Corporation
Dean Gonzales	Advanced Micro Devices, Inc.
Alan Goodrum	Hewlett-Packard Company
Robert Gough	Intel Corporation
Lucian Gozu	Neterion Corporation
Ilya Granovsky	IBM Corporation
Brien Gray	Intel Corporation
Richard Greene	Intel Corporation
Stephen Greenwood	ATI Technologies Inc.
Michael J. Greger	Intel Corporation
Buck Gremel	Intel Corporation
Andrew Gruber	Advanced Micro Devices, Inc.
George Gruber	Qualcomm
Vijay Gudur	LSI Logic Corporation
Dale Gulick	Advanced Micro Devices, Inc.
Liping Guo	Marvell Semiconductor, Inc.
Mickey Gutman	Intel Corporation
Clifford D. Hall	Intel Corporation
Stephen H. Hall	Intel Corporation
Joseph Hamel	IBM Corporation
Ken Haren	Intel Corporation
David Harriman	Intel Corporation
Hiromitsu Hashimoto	NEC Corporation

Name	Affiliation
George R. Hayek	Intel Corporation
Wenmu He	Texas Instruments Incorporated
Matt Hendrick	Intel Corporation
Hamish Hendry	Cadence Design Systems, Inc.
Michael Herz	Blackberry
Bent Hessen-Schmidt	Tektronix, Inc.
Hanh Hoang	Intel Corporation
Joe Hock	Intel Corporation
Michael Hopgood	NVIDIA Corporation
Dorcas Hsia	NVIDIA Corporation
James Huang	Advanced Micro Devices, Inc.
Robert Huang	NVIDIA Corporation
Yifan Huang	Amphenol-FCI
Kamlesh Hujwant	Advanced Micro Devices, Inc.
Mark Hummel	Advanced Micro Devices, Inc.
Mikal Hunsaker	Intel Corporation
Joaquin Ibanez	Synopsys, Inc.
Yasser Ibrahim	Microsoft Corporation
Franko Itay	Intel Corporation
Carl Jackson	Hewlett-Packard Company
David R. Jackson	Intel Corporation
Praveen Jain	NVIDIA Corporation
Martin James	Cadence Design Systems, Inc.
Duane January	Intel Corporation
James E. Jaussi	Intel Corporation
Mike Jenkins	LSI Logic Corporation
Peter Jenkins	GLOBALFOUNDRIES Inc.
Hong Jiang	Intel Corporation
Viek Joshi	Intel Corporation
Dave Kaffine	Oracle Corporation
David Kahmayhew	Oracle Corporation

Name	Affiliation
Ravi Kammaje	LSI Logic Corporation
Girish Karanam	LSI Logic Corporation
Kapil Karkra	Intel Corporation
Navnit Kashyap	Cadence Design Systems, Inc.
Chad Kendall	Broadcom Inc.
Mukund Kharti	Dell Computer Corporation
Gyanaranjan Khuntia	Mentor Graphics
David Kimble	Texas Instruments Incorporated
Lavi Koch	Mellanox Technologies, Ltd.
Mohammad Kolbehdari	Intel Corporation
Abhimanyu Kolla	Intel Corporation
Ganesh Kondapuram	Intel Corporation
Kwok Kong	IDT Corporation
Michael Krause	Hewlett-Packard Enterprise
Gopi Krishnamurthy	Cadence Design Systems, Inc.
Sreenivas Krishnan	NVIDIA Corporation
Kumaran Krishnasamy	Broadcom Inc.
Steve Krooswyk	Samtec
Manjari Kulkarni	Intel Corporation
Akhilesh Kumar	Intel Corporation
Mohan J. Kumar	Intel Corporation
Richard Kunze	Intel Corporation
Hugh Kurth	Oracle Corporation
Seh Kwa	Intel Corporation
Ricky Lai	Hewlett Packard Enterprise
Sunny Lam	Intel Corporation
Dave Landsman	Sandisk
Brian Langendorf	NVIDIA Corporation
Raymond R. Law	Intel Corporation
Dror Lazar	Intel Corporation
Brian L'Ecuyer	Agilent Technologies, Inc.

Name	Affiliation
Beomtek Lee	Intel Corporation
Clifford D. Lee	Intel Corporation
David M. Lee	Intel Corporation
Edward Lee	Advanced Micro Devices, Inc.
Moshe Leibowitz	IBM Corporation
Tony L. Lewis	Intel Corporation
Mike Li	Wavecrest Corporation
Paul Li	Pericom Semiconductor Corporation
Stephen Li	Texas Instruments Incorporated
Ying Li	NVIDIA Corporation
Tao Liang	Intel Corporation
Andrew K. Lillie	Intel Corporation
Wendy Liu	Advanced Micro Devices, Inc.
Jeff Lukanc	IDT Corporation
Jeffrey Lu	IBM Corporation
Betty Luk	Advanced Micro Devices, Inc.
Ngoc Luu	Advanced Micro Devices, Inc.
Kenneth Ma	Broadcom Inc.
Stephen Ma	Advanced Micro Devices, Inc.
Zorik Machulsky	IBM Corporation
Mallik Mahalingam	VMware, Inc.
Kevin Main	Texas Instruments Incorporated
Steven Makow	IBM Corporation
Tony Mangefeste	Microsoft Corporation
Nilange Manisha	Intel Corporation
Steve Manning	Advanced Micro Devices, Inc.
Gold Mao	VIA Technologies, Inc.
Jarek Marczewski	Advanced Micro Devices, Inc.
Mark Marlett	LSI Logic Corporation
Alberto Martinez	Intel Corporation
Andrew Martwick	Intel Corporation

Name	Affiliation
Paul Mattos	GLOBALFOUNDRIES Inc.
Robert A. Mayer	Intel Corporation
David Mayhew	(Stargen, Inc.) Advanced Micro Devices, Inc.
Mohiuddin Mazumder	Intel Corporation
Mehdi Mechaik	Cadence Design Systems, Inc.
Mehdi M. Mechaik	NVIDIA Corporation
Pranav H. Mehta	Intel Corporation
Vishal Mehta	NVIDIA Corporation
Richard Mellitz	Intel Corporation
Adi Menachem	Valens Semiconductor, Ltd.
Cindy Merkin	Dell Computer Corporation
Slobodan Milijevic	Microsemi Corporation
Dennis Miller	Intel Corporation
Jason Miller	Oracle Corporation
Robert J. Miller	Intel Corporation
Michael Mirmak	Intel Corporation
Suneel Mitbander	Intel Corporation
Mohammad Mobin	NVIDIA Corporation
Daniel Moertl	IBM Corporation
Lee Mohrmann	National Instruments Corporation
Puga Nathal Moises	Intel Corporation
Richard Moore	Cavium Inc.
Wayne Moore	Intel Corporation
Douglas R. Moran	Intel Corporation
Terry Morris	Hewlett-Packard Enterprise
Jeff C. Morriss	Intel Corporation
Linna Mu	Avago Technologies
Sridhar Muthrasanallur	Intel Corporation
Suresh Babu M. V.	LSI Logic Corporation
Gautam V. Naik	LSI Logic Corporation
Mohan K. Nair	Intel Corporation

Name	Affiliation
Mukund Narasimhan	Intel Corporation
Alon Naveh	Intel Corporation
Ramin Neshati	Intel Corporation
Surena Neshvad	Intel Corporation
Andy Ng	IDT Corporation
Manisha Nilange	Intel Corporation
Paul Nitza	Marvell Semiconductor, Inc
Hajime Nozaki	NEC Corporation
Kugao Ohuchi	NEC Corporation
Vitor Oliveira	Synopsys, Inc.
Olufemi Oluwafemi	Intel Corporation
Takuya Omura	Synopsys, Inc.
Peter Onufryk	Integrated Device Technology
Mike Osborn	Advanced Micro Devices, Inc.
Jake Oshins	Microsoft Corporation
Randy Ott	Intel Corporation
Jonathan Owen	Advanced Micro Devices, Inc.
Ali Oztaskin	Intel Corporation
David Pabisz	Oracle Corporation
Shreeram Palghat	Intel Corporation
Jim Panian	Qualcomm
Marc Pegolotti	ServerWorks, Inc.
Henry Peng	Intel Corporation
Akshay Pethe	Intel Corporation
Chris Pettey	NextIO, Inc.
Tien Pham	Hewlett Packard Enterprise
Lu-vong Phan	Intel Corporation
Tony Pierce	Microsoft Corporation
Edmund Poh	Molex, Inc.
Harshit Poladia	Intel Corporation
Jim Prijic	Intel Corporation

Name	Affiliation
Dave Puffer	Intel Corporation
Duane Quiet	Intel Corporation
Jeffrey D. Rabe	Intel Corporation
Andy Raffman	Microsoft Corporation
Kianoush Rahbar	Intel Corporation
Guru Rajamani	Intel Corporation
Ramesh Raman	LSI Logic Corporation
Adee Ran	Intel Corporation
Todd Rasmus	IBM Corporation
Renato Recio	IBM Corporation
Ramakrishna Reddy	LSI Logic Corporation
Jack Regula	PLX Technology, Inc.
Dick Reohr	Intel Corporation
Curtis Ridgeway	LSI Logic Corporation
Dwight Riley	Hewlett-Packard Enterprise
Yoav Rozenbert	Mellanox Technologies, Ltd.
Chris Runhaar	NVIDIA Corporation
Rajanataraj S.	LSI Logic Corporation
Devang Sachdev	NVIDIA Corporation
Gene Saghi	Broadcom Inc.
Rajesh Sankaran	Intel Corporation
Bill Sauber	Dell Computer Corporation
Joseph Schachner	Teledyne LeCroy
Mike Schaecher	Oracle Corporation
Joe Schaefer	Intel Corporation
Michael Scheid	Broadcom Inc.
Daren Schmidt	Intel Corporation
Mark Schmisseur	Intel Corporation
Richard Schober	NVIDIA Corporation
Zale Schoenborn	Intel Corporation
Rick Schuckle	Dell Computer Corporation

Name	Affiliation
Richard Schumacher	Hewlett-Packard Enterprise
Jeremiah Schwartz	Intel Corporation
Tudor Secasiu	Intel Corporation
Kazunori Seki	Synopsys, Inc.
Oren Sela	Mellanox Technologies, Ltd.
Kevin Senohrabek	Advanced Micro Devices, Inc.
Kalev Sepp	Tektronix, Inc.
Prashant Sethi	Intel Corporation
Ankur Shah	Intel Corporation
Vasudevan Shanmugasundaram	Intel Corporation
Wesley Shao	Oracle Corporation
Prateek Sharma	LSI Logic Corporation
Charlie Shaver	Hewlett-Packard Company
Robert Sheldon	Oracle Corporation
John Sheplock	IBM Corporation
Wenjun Shi	NVIDIA Corporation
Milton Shih	Oracle Corporation
Mark Shillingburg	Agilent Technologies
Oren Shirak	Mellanox Technologies, Ltd.
Sarvesh Shrivastava	Qualcomm
Bill Simms	NVIDIA Corporation
Vinita Singhal	NVIDIA Corporation
Dan Slocombe	Cadence Design Systems, Inc.
Brian Small	Northwest Logic, Inc.
George Smith	Microsoft Corporation
Shamnad SN	LSI Logic Corporation
Rick Sodke	PMC-Sierra
Gary Solomon	Intel Corporation
Richard Solomon	Synopsys, Inc.
Gao Song	IDT Corporation
Brad Sonksen	Marvell Semiconductor, Inc

Name	Affiliation
Walter Soto	Broadcom Inc.
Fulvio Spagna	Intel Corporation
Jason Squire	Molex, Inc.
Patrick Stabile	Oracle Corporation
Sean O. Stalley	Intel Corporation
Stan Stanski	IBM Corporation
Hermann Stehling	Bitifeye Digital Test Solutions
John Stonick	Synopsys, Inc.
Ron Swartz	Intel Corporation
Tim Symons	PMC-Sierra
Miki Takahashi	NEC Corporation
Gerry Talbot	Advanced Micro Devices, Inc.
Anthony Tam	Advanced Micro Devices, Inc.
Wanru Tao	Oracle Corporation
Mark Taylor	NVIDIA Corporation
Matthew Tedone	LSI Logic Corporation
Grigori Temkine	Advanced Micro Devices, Inc.
Peter Teng	NEC Corporation
Bruce A. Tennant	Intel Corporation
Larry Tesdall	Marvell Semiconductor, Inc
Andrew Thornton	Microsoft Corporation
Steven Thurber	IBM Corporation
Mike Tobin	Intel Corporation
Ben Toby	Hewlett Packard Enterprise
Duke Tran	Broadcom Inc.
Tan V. Tran	Intel Corporation
Alok Tripathi	Intel Corporation
William Tsu	NVIDIA Corporation
Alexander Umansky	Huawei Technologies Co.
Chris Van Beek	Intel Corporation
Arie van der Hoeven	Microsoft Corporation

Name	Affiliation
Andrew Vargas	Intel Corporation
Robertson Velez	ATI Technologies Inc.
Archana Vasudevan	LSI Logic Corporation
Kiran Velicheti	Intel Corporation
Balaji Vembu	Intel Corporation
Gary Verdun	Dell Computer Corporation
Sushil Verghese	Broadcom Inc.
Divya Vijayaraghavan	Altera Corporation
Ravindra Viswanath	LSI Logic Corporation
Pete D. Vogt	Intel Corporation
Andrew Volk	Intel Corporation
Mahesh Wagh	Intel Corporation
Clint Walker	Intel Corporation
Davis Walker	Microsoft Corporation
Hui Wang	IDT Corporation
Kai A. Wang	Intel Corporation
Leo Warmuth	NXP Semiconductors
Dan Wartski	Intel Corporation
Neil Webb	Cadence Design Systems, Inc.
Eric Wehage	Intel Corporation
Dong Wei	Hewlett-Packard Company
Ron Weimer	QLogic Corporation
Amir Wiener	Intel Corporation
Marc Wells	Intel Corporation
Rob Wessel	Hewlett-Packard Company
Stephen F. Whalley	Intel Corporation
Bryan White	Intel Corporation
Paul Whittemore	Oracle Corporation
Amir Wiener	Intel Corporation
Timothy Wig	Intel Corporation
Craig Wiley	Parade

Name	Affiliation
Jim Williams	Emulex Corporation
Theodore L. Willke	Intel Corporation
Dawn Wood	Intel Corporation
John Wood	Emulex Corporation
David Woodral	QLogic Corporation
David Wooten	Microsoft Corporation
Zhi Wong	Altera Corporation
David Wormus	LSI Logic Corporation
David Wu	Broadcom Inc.
William Wu	Broadcom Inc.
Zuoguo Wu	Intel Corporation
Liu Xin	IDT Corporation
Dan Yaklin	Texas Instruments Incorporated
Howard Yan	Intel Corporation
Jane Yan	Oracle Corporation
Al Yanes	IBM Corporation
Gin Yee	Advanced Micro Devices, Inc.
Ahmed Younis	Xilinx, Inc.
Chi-Ho Yue	NVIDIA Corporation
Wayne Yun	Advanced Micro Devices, Inc.
Dave Zenz	Dell Computer Corporation
Shubing Zhai	IDT Corporation
Wei Zhang	Broadcom Inc.
Bo Zhang	Intel Corporation
Guoqing Zhang	Cadence Design Systems, Inc.
Kevin Ziegler	Tektronix, Inc.
Yaping Zhou	NVIDIA Corporation