H2D Req to a Type 2 device on addresses that belong to the Device-attached Memory. An H2D Req from the host that targets Device-attached memory can cause coherency issues if the device were to respond with data and, more generally speaking, protocol corner cases.

To avoid these issues, both HDM-D Type 2 devices and HDM-DB Type 2 devices are required to:

- Detect H2D Req that target Device-attached Memory
- When detected, unconditionally respond with RspIHitI, disregarding all internal states and without changing any internal states (e.g., don't touch the cache)

3.3 CXL.mem

3.3.1 Introduction

The CXL Memory Protocol is called CXL.mem, and it is a transactional interface between the CPU and Memory. It uses the phy and link layer of CXL when communicating across dies. The protocol can be used for multiple different Memory attach options including when the Memory Controller is located in the Host CPU, when the Memory Controller is within an Accelerator device, or when the Memory Controller is moved to a memory buffer chip. It applies to different Memory types (e.g., volatile, persistent, etc.) and configurations (e.g., flat, hierarchical, etc.) as well.

The CXL.mem provides 3 basic coherence models for CXL.mem Host-managed Device Memory (HDM) address regions exposed by the CXL.mem protocol:

- HDM-H (Host-only Coherent): Used only for Type 3 Devices
- HDM-D (Device Coherent): Used only for legacy Type 2 Devices that rely on CXL.cache to manage coherence with the Host
- HDM-DB (Device Coherent using Back-Invalidate): Can be used by Type 2 Devices or Type 3 Devices

Note: The view of the address region must be consistent on the CXL.mem path between the Host and the Device.

The coherency engine in the CPU interfaces with the Memory (Mem) using CXL.mem requests and responses. In this configuration, the CPU coherency engine is regarded as the CXL.mem Master and the Mem device is regarded as the CXL.mem Subordinate. The CXL.mem Master is the agent which is responsible for sourcing CXL.mem requests (e.g., reads, writes, etc.) and a CXL.mem Subordinate is the agent which is responsible for responding to CXL.mem requests (e.g., data, completions, etc.).

When the Subordinate maps HDM-D/HDM-DB, CXL.mem protocol assumes the presence of a device coherency engine (DCOH). This agent is assumed to be responsible for implementing coherency related functions such as snooping of device caches based on CXL.mem commands and update of Metadata fields.

Support for memory with Metadata is optional but this needs to be negotiated with the Host in advance. If the device supports "Metabits Storage" Feature, this mechanism may be used to negotiate the Metadata configuration. Other negotiation mechanisms are beyond the scope of this specification. If Metadata is not supported by device-attached memory, the DCOH will still need to use the Host supplied Metadata updates to interpret the commands. If Metadata is supported by device-attached memory, it can be used by Host to implement a coarse snoop filter for CPU sockets. In the HDM-H address region, the usage is defined by the Host. The protocol allows for 2 bits of Metadata to be stored and returned.

CXL.mem transactions from Master to Subordinate are called "M2S" and transactions from Subordinate to Master are called "S2M".

Within M2S transactions, there are three message classes:

- Request without data - generically called Requests (Req)
- Request with Data - (RwD)
- Back-Invalidate Response - (BIRsp)

Similarly, within S2M transactions, there are three message classes:

- Response without data - generically called No Data Response (NDR)
- Response with data - generically called Data Response (DRS)
- Back-Invalidate Snoop - (BISnp)

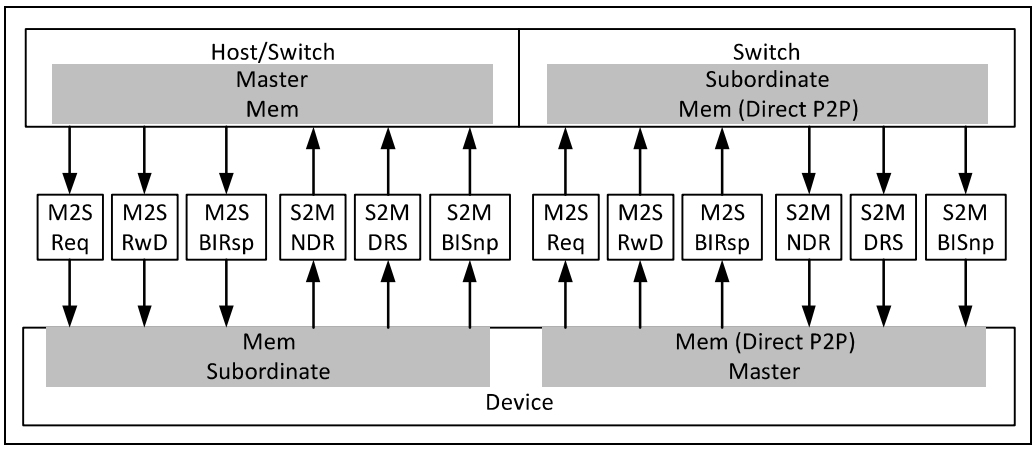
The next sections describe the above message classes and opcodes in detail. Each message in will support 3 variants: 68B Flit, 256B Flit, and PBR Flit. The use of each of these will be negotiated in the physical layer for each link as defined in Chapter 6.0.

3.3.2 CXL.mem Channel Description

In general, the CXL.mem channels work independently of one another to ensure that forward progress is maintained. Details of the specific ordering allowances and requirements between channels are captured in Section 3.4. Within a channel there are no ordering rules, but exceptions to this are described in Section 3.3.12.

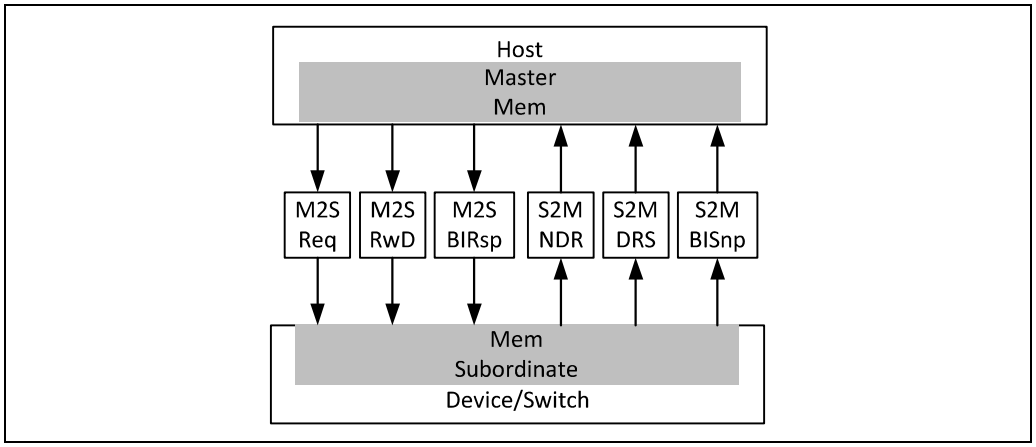
The device interface for CXL.mem defines 6 channels on primary memory protocol and an additional 6 to support direct P2P as shown in Figure 3-18. Devices that support HDM-DB must support the BI* channels (S2M BISnp and M2S BIRsp). Type 2 devices that use the HDM-D memory region may not have the BI* channels. Type 3 devices (Memory Expansion) may support HDM-DB to support direct peer-to-peer on CXL.io. MLD and G-FAM devices may use HDM-DB to enable multi-host coherence and direct peer-to-peer on CXL.mem. The HDM-DB regions will be known by software and programmed as such in the decode registers and these regions will follow the protocol flows, using the BISnp channels as defined in Appendix C, "Memory Protocol Tables."

Figure 3-18. CXL.mem Channels for Devices



For Hosts, the number of channels are defined in Figure 3-19. The channel definition is the same as for devices.

Figure 3-19. CXL.mem Channels for Hosts



3.3.2.1 Direct P2P CXL.mem for Accelerators

In certain topologies, an accelerator (Type 1, Type 2, or Type 3) device may optionally be enabled to communicate with peer Type 3 memories with CXL.mem protocol. Support for such communication is provided by an additional set of CXL.mem channels, with their directions reversed from conventional CXL.mem as shown in as shown in Figure 3-18. These channels exist only on a link between the device and the switch downstream port to which the link is attached. Ordering requirements, message formats, and channel semantics are the same as for conventional CXL.mem. Topologies supporting Direct P2P.mem require an accelerator (requester device) and a target Type 3 peer memory device which are both directly connected to a PBR Edge DSP. PBR routing is required because not all CXL.mem messages contain sufficient information for an HBR switch to determine whether to route between a device and the host or a device and a peer device. Edge DSPs contain tables (FAST and LDST) which enable routing to the proper destination.

Details related to the device in-out dependence covering standard CXL.mem target and the source of Direct P2P CXL.mem and are covered in Table 3-59.

3.3.2.2 Snoop Handling with Direct P2P CXL.mem

It is possible for a device that is using the Direct P2P CXL.mem interface to receive a snoop on H2D Req for an address that the device had previously requested over its P2P CXL.mem interface. This could occur, for example, if the host has snoop filtering disabled. Conversely, the device could receive an S2M BISnp from a peer for a line that it had acquired over CXL.cache through the host.

As a result, devices that use the Direct P2P CXL.mem interface are required to track which interface was used when a cacheline was requested and respond normally to snoops using this channel. If the device receives a snoop on a different interface, the device shall respond as though it does not have the address cached returning RspIHitI or BIRspI and shall not change the cacheline state.

IMPLEMENTATION NOTE

How the device tracks which interface was used to request each cacheline is implementation dependent. One method of tracking could be for the device to maintain a table of address ranges, programmed by software with an indication for each range whether the CXL.cache or Direct P2P CXL.mem interface should be used. This table could then be looked up when snoops are received. Other methods may also be used.

3.3.3 Back-Invalidate Snoop

To enable a device to implement an inclusive Snoop Filter for tracking host caching of device memory, a Back-Invalidate Snoop (BISnp) is initiated from the device to change the cache state of the host. The flows related to this channel are captured in Section 3.5.1. The definition of "inclusive Snoop Filter" for the purpose of CXL is a device structure that tracks cacheline granular host caching and is a limited size that is a small subset of the total Host Physical Address space supported by the device.

In 68B flits, only the CXL.cache D2H Request flows can be used for device-attached memory to manage coherence with the host as shown in Section 3.5.2.3. This flow is used for addresses with the HDM-D memory attribute. A major constraint with this flow is that the D2H Req channel can be blocked waiting on forward progress of the M2S Request channel which disallows an inclusive Snoop Filter architecture. For the HDM-DB memory region, the BISnp channel (instead of CXL.cache) is used to resolve coherence. CXL host implementations may have a mix of devices with HDM-DB and HDM-D below a Root Port.

The rules related to Back-Invalidate are spread around in different areas of the specification. The following list captures a summary and pointers to requirements:

- Ordering rules in Section 3.4
- Conflict detection flows and blocking in Section 3.5.1
- Protocol Tables in Section C.1.2
- BI-ID configuration in Section 9.14
- If an outstanding S2M BISnp is pending to an address the device must block M2S Req to the same address until the S2M BISnp is completed with the corresponding M2S BIRsp
- M2S RwD channel must complete/drain without dependence on M2S Req or S2M BISnp

IMPLEMENTATION NOTE

Detailed performance implications of the implementation of an Inclusive Snoop Filter are beyond the scope of this specification, but high-level considerations are captured here:

- The number of cachelines that are tracked in an Inclusive Snoop Filter is determined based on host-processor caching of the address space. This is a function of the use model and the cache size in the host processor with upsizing of 4x or more. The 4x is based on an imprecise estimation of the unknowns in future host implementations and mismatch in Host cache ways/sectors as compared to Snoop-Filter ways/sectors.
- Device should have the capability to track BISnp messages triggered by Snoop Filter capacity evictions without immediately blocking requests on the M2S Req channel when the Inclusive Snoop Filter becomes full. In the case that the BISnp tracking structure becomes full the M2S Req channel will need to be blocked for functional correctness, but the design should size this BISnp tracker to ensure that blocking of the M2S Req channel is a rare event.
- The state per cacheline could be implemented as 2 states or 3 states. For 2 states, it would track the host in I vs. A, where A-state would represent "Any" possible MESI state in the host. For 3 states, it would add the precision of S-state tracking in which the Host may have at most a shared copy of the cacheline.

3.3.4 QoS Telemetry for Memory

QoS Telemetry for Memory is a mechanism for memory devices to indicate their current load level (DevLoad) in each response message for CXL.mem requests and each completion for (CXL.io) UIO requests. This enables the host or peer requester to meter the issue rate of CXL.mem requests and UIO requests to portions of devices, individual devices, or groups of devices as a function of their load level, optimizing the performance of those memory devices while limiting fabric congestion. This is especially important for CXL hierarchies containing multiple memory types (e.g., DRAM and persistent memory), Multi-Logical-Device (MLD) components, and/or G-FAM Devices (GFDs).

In addition to use cases with hosts that access memory devices, QoS Telemetry for memory supports the UIO Direct P2P to HDM (see Section 7.7.9) and Direct P2P CXL.mem for Accelerators (see Section 7.7.10) use cases. For these, the peer requester for each UIO or .mem request receives a DevLoad indication in each UIO completion or .mem response. For the UIO Direct P2P use case, the peer requester may be native PCIe or CXL. Within this section, "hosts/peers" is a shorthand for referring to host and/or peer requesters that access HDM devices.

Certain aspects of QoS Telemetry are mandatory for current CXL memory devices while other aspects are optional. CXL switches have no unique requirements for supporting QoS Telemetry. It is strongly recommended for Hosts to support QoS Telemetry as guided by the reference model contained in this section. For peer requesters, the importance of supporting QoS Telemetry depends on the device type, its capabilities, and its specific use case(s).

3.3.4.1 QoS Telemetry Overview

The overall goal of QoS Telemetry is for memory devices to provide immediate and ongoing DevLoad feedback to their associated hosts/peers, for use in dynamically adjusting their request-rate throttling. If a device or set of Devices become overloaded, the associated hosts/peers increase their amount of request rate throttling. If such

Devices become underutilized, the associated hosts/peers reduce their amount of request rate throttling. QoS Telemetry is architected to help hosts/peers avoid overcompensating and/or undercompensating.

Host/peer memory request rate throttling is optional and primarily implementation specific.

Table 3-29. Impact of DevLoad Indication on Host/Peer Request Rate Throttling

DevLoad Indication Returned in Responses	Host/Peer Request Rate Throttling
Light Load	Reduce throttling (if any) soon
Optimal Load	Make no change to throttling
Moderate Overload	Increase throttling immediately
Severe Overload	Invoke heavy throttling immediately

To accommodate memory devices supporting multiple types of memory more optimally, a device is permitted to implement multiple QoS Classes, which are identified sets of traffic, between which the device supports differentiated QoS and significant performance isolation. For example, a device supporting both DRAM and persistent memory might implement two QoS Classes, one for each type of supported memory. Providing significant performance isolation may require independent internal resources (e.g., individual request queues for each QoS Class).

This version of the specification does not provide architected controls for providing bandwidth management between device QoS Classes.

MLDs provide differentiated QoS on a per-LD basis. MLDs have architected controls specifying the allocated bandwidth fraction for each LD when the MLD becomes overloaded. When the MLD is not overloaded, LDs can use more than their allocated bandwidth fraction, up to specified fraction limits based on maximum sustained device bandwidth.

GFDs provide differentiated QoS on a per-host/peer basis. GFDs have architected controls that specify a QoS Limit Fraction value for each host/peer, based on maximum sustained device bandwidth.

HDM-DB devices send BISnp requests and receive BIRsp responses as a part of processing requests that they receive from host/peer requesters. BISnp and BIRsp messages shall not be tracked by QoS Telemetry mechanisms. If a BISnp triggers a host/peer requester writing back cached data, those transactions will be tracked by QoS Telemetry.

The DevLoad indication from CXL 1.1 memory devices will always indicate Light Load, allowing those devices to operate as best they can with hosts/peers that support QoS Telemetry, though they cannot have their memory request rate actively metered by the host/peer. Light Load is used instead of Optimal Load in case any CXL 1.1 devices share the same host/peer throttling range with current memory devices. If CXL 1.1 devices were to indicate Optimal Load, they would overshadow the DevLoad of any current devices indicating Light Load.

3.3.4.2 Reference Model for Host/Peer Support of QoS Telemetry

Host/peer support for QoS Telemetry is strongly recommended but not mandatory.

QoS Telemetry provides no architected controls for mechanisms in hosts/peers. However, if a host/peer implements independent throttling for multiple distinct sets of memory devices through a given port, the throttling must be based on HDM ranges, which are referred to as host/peer throttling ranges.

The reference model in this section covers recommended aspects for how a host/peer should support QoS Telemetry. Such aspects are not mandatory, but they should help maximize the effectiveness of QoS Telemetry in optimizing memory device performance while providing differentiated QoS and reducing CXL fabric congestion.

Each host/peer is assumed to support distinct throttling levels on a throttling-range basis, represented by Throttle[Range]. Throttle[Range] is periodically adjusted by conceptual parameters NormalDelta and SevereDelta. During each sampling period for a given Throttle[Range], the host/peer records the highest DevLoad indication reported for that throttling range, referred to as LoadMax.

Table 3-30. Recommended Host/Peer Adjustment to Request Rate Throttling

LoadMax Recorded by Host/Peer	Recommended Adjustment to Request Rate Throttling
Light Load	Throttle[Range] decremented by NormalDelta
Optimal Load	Throttle[Range] unchanged
Moderate Overload	Throttle[Range] incremented by NormalDelta
Severe Overload	Throttle[Range] incremented by SevereDelta

Any increments or decrements to Throttle[Range] should not overflow or underflow legal values, respectively.

Throttle[Range] is expected to be adjusted periodically, every tH nanoseconds unless a more immediate adjustment is warranted. The tH parameter should be configurable by platform-specific software, and ideally configurable on a per-throttling-range basis. When tH expires, the host/peer should update Throttle[Range] based on LoadMax, as shown in Table 3-30, and then reset LoadMax to its minimal value.

Round-trip fabric time is the sum of the time for a request message to travel from host/peer to Device, plus the time for a response message to travel from Device to host/peer. The optimal value for tH is anticipated to be a bit larger than the average round-trip fabric time for the associated set of devices (e.g., a few hundred nanoseconds). To avoid overcompensation by the host/peer, time is needed for the received stream of DevLoad indications in responses to reflect the last Throttle[Range] adjustment before the host/peer makes a new adjustment.

If the host/peer receives a Moderate Overload or Severe Overload indication, it is strongly recommended for the host/peer to make an immediate adjustment in throttling, without waiting for the end of the current tH sampling period. Following that, the host/peer should reset LoadMax and then wait tH nanoseconds before making an additional throttling adjustment, to avoid overcompensating.

3.3.4.3 Memory Device Support for QoS Telemetry

3.3.4.3.1 QoS Telemetry Register Interfaces

An MLD must support a specified set of MLD commands from the MLD Component Command Set as documented in Section 7.6.7.4. These MLD commands provide access to a variety of architected capability, control, and status registers for a Fabric Manager to use via the FM API.

A GFD must support a specified set of GFD commands from the GFD Component Management Command Set as documented in Section 8.2.10.9.10. These GFD commands provide access to a variety of architected capability, control, and status registers for a Fabric Manager to use via the FM API.

If an SLD supports the Memory Device Command set, it must support a specified set of SLD QoS Telemetry commands. See Section 8.2.10.9. These SLD commands provide access to a variety of architected capability, control, and status fields for management by system software via the CXL Device Register interface.

Each "architected QoS Telemetry" register is one that is accessible via the above mentioned MLD commands, GFD commands, and/or SLD commands.

3.3.4.3.2 Memory Device QoS Class Support

Each CXL memory device may support one or more QoS Classes. The anticipated typical number is one to four, but higher numbers are not precluded. If a device supports only one type of media, it may be common for it to support one QoS Class. If a device supports two types of media, it may be common for it to support two QoS Classes. A device supporting multiple QoS Classes is referred to as a multi-QoS device.

This version of the specification does not provide architected controls for providing bandwidth management between device QoS Classes. Still, it is strongly recommended that multi-QoS devices track and report DevLoad indications for different QoS Classes independently, and that implementations provide as much performance isolation between different QoS Classes as possible.

3.3.4.3.3 Memory Device Internal Loading (IntLoad)

A CXL memory device must continuously track its internal loading, referred to as IntLoad. A multi-QoS device should do so on a per-QoS-Class basis.

A device must determine IntLoad based at least on its internal request queuing. For example, a simple device may monitor the instantaneous request queue depth to determine which of the four IntLoad indications to report. It may also incorporate other internal resource utilizations, as summarized in Table 3-31.

Table 3-31. Factors for Determining IntLoad

IntLoad	Queuing Delay inside Device	Device Internal Resource Utilization
Light Load	Minimal	Readily handles more requests
Optimal Load	Modest to Moderate	Optimally utilized
Moderate Overload	Significant	Limiting throughput and/or degrading efficiency
Severe Overload	High	Heavily overloaded and/or degrading efficiency

The actual method of IntLoad determination is device-specific, but it is strongly recommended that multi-QoS devices implement separate request queues for each QoS Class. For complex devices, it is recommended for them to determine IntLoad based on internal resource utilization beyond just request queue depth monitoring.

Although the IntLoad described in this section is a primary factor in determining which DevLoad indication is returned in device responses, there are other factors that may need to be considered, depending upon the situation (see Section 3.3.4.3.4 and Section 3.3.4.3.5).

3.3.4.3.4 Egress Port Backpressure

Even under a consistent Light Load, a memory device may experience flow control backpressure at its egress port. This is readily caused if an RP is oversubscribed by multiple memory devices below a switch. Prolonged egress port backpressure usually indicates that one or more upstream traffic queues between the device and the RP are full, and the delivery of responses from the device to the host/peer is significantly

delayed. This makes the QoS Telemetry feedback loop less responsive and the overall mechanism less effective. Egress Port Backpressure is an optional normative mechanism to help mitigate the negative effects of this condition.

IMPLEMENTATION NOTE

Egress Port Backpressure Leading to Larger Request Queue Swings

When the QoS Telemetry feedback loop is less responsive, the device's request queue depth is prone to larger swings than normal.

When the queue depth is increasing, the delay in the host/peer receiving Moderate Overload or Severe Overload indications results in the queue getting more full than normal, in extreme cases filling completely and forcing the ingress port to exert backpressure to incoming downstream traffic.

When the queue depth is decreasing, the delay in the host/peer receiving Light Load indications results in the queue getting more empty than normal, in extreme cases emptying completely, and causing device throughput to drop unnecessarily.

Use of the Egress Port Backpressure mechanism helps avoid upstream traffic queues between the device and its RP from filling for extended periods, reducing the delay of responses from the device to the host/peer. This makes the QoS Telemetry feedback loop more responsive, helping avoid excessive request queue swings.

IMPLEMENTATION NOTE

Minimizing Head-of-Line Blocking with Upstream Responses from MLDs/ GFDs

When one or more upstream traffic queues become full between the MLD and one or more of its congested RPs, head-of-line (HOL) blocking associated with this congestion can delay or block traffic targeting other RPs that are not congested.

Egress port backpressure for extended periods usually indicates that the ingress port queue in the Downstream Switch Port above the device is often full. Responses in that queue targeting congested RPs can block responses targeting uncongested RPs, reducing overall device throughput unnecessarily.

Use of the Egress Port Backpressure mechanism helps reduce the average depth of queues carrying upstream traffic. This reduces the delay of traffic targeting uncongested RPs, increasing overall device throughput.

The Egress Port Congestion Supported capability bit and the Egress Port Congestion Enable control bit are architected QoS Telemetry bits, which indicate support for this optional mechanism plus a means to enable or disable it. The architected Backpressure Average Percentage status field returns a current snapshot of the measured egress port average congestion.

QoS Telemetry architects two thresholds for the percentage of time that the egress port experiences flow control backpressure. This condition is defined as the egress port having flits or messages waiting for transmission but is unable to transmit them due to a lack of suitable flow control credits. If the percentage of congested time is greater than or equal to Egress Moderate Percentage, the device may return a DevLoad indication of Moderate Overload. If the percentage of congested time is greater than or

equal to Egress Severe Percentage, the device may return a DevLoad indication of Severe Overload. The actual DevLoad indication returned for a given response may be the result of other factors as well.

A hardware mechanism for measuring Egress Port Congestion is described in Section 3.3.4.3.9.

3.3.4.3.5 Temporary Throughput Reduction

There are certain conditions under which a device may temporarily reduce its throughput. Envisioned examples include a non-volatile memory (NVM) device undergoing media maintenance, a device cutting back its throughput for power/thermal reasons, and a DRAM device performing refresh. If a device is significantly reducing its throughput capacity for a temporary period, it may help mitigate this condition by indicating Moderate Overload or Severe Overload in its responses shortly before the condition occurs and only as long as really necessary. This is a device-specific optional mechanism.

The Temporary Throughput Reduction mechanism can give proactive advanced warning to associated hosts/peers, which can then increase their throttling in time to avoid the device's internal request queue(s) from filling up and potentially causing ingress port congestion. The optimum amount of time for providing advanced warning is highly device-specific, and a function of several factors, including the current request rate, the amount of device internal buffering, the level/duration of throughput reduction, and the fabric round-trip time.

A device should not use the mechanism unless conditions truly warrant its use. For example, if the device is currently under Light Load, it's probably not necessary or appropriate to indicate an Overload condition in preparation for a coming event. Similarly, a device that indicates an Overload condition should not continue to indicate the Overload condition past the point where it's needed.

The Temporary Throughput Reduction Supported capability bit and the Temporary Throughput Reduction Enable control bit are architected QoS Telemetry bits, which indicate support for this optional mechanism plus a means to enable or disable it.

> **IMPLEMENTATION NOTE**
>
> **Avoid Unnecessary Use of Temporary Throughput Reduction**
>
> Ideally, a device should be designed to limit the severity and/or duration of its temporary throughput reduction events enough to where the use of this mechanism is not needed.

3.3.4.3.6 DevLoad Indication by Multi-QoS and Single-QoS SLDs

For SLDs, the DevLoad indication returned in each response is determined by the maximum of the device's IntLoad, Egress Port Congestion state, and Temporary Throughput Reduction state, as detailed in Section 3.3.4.3.3, Section 3.3.4.3.4, and Section 3.3.4.3.5. For example, if IntLoad indicates Light Load, Egress Port Congestion indicates Moderate Overload, and Temporary Throughput Reduction does not indicate an overload, the resulting DevLoad indication for the response is Moderate Overload.

3.3.4.3.7 DevLoad Indication by Multi-QoS and Single-QoS MLDs

For MLDs, the DevLoad indication returned in each response is determined by the same factors as for SLDs, with additional factors used for providing differentiated QoS on a per-LD basis. Architected controls specify the allocated bandwidth for each LD as a fraction of total LD traffic when the MLD becomes overloaded. When the MLD is not

overloaded, LDs can use more than their allocated bandwidth fraction, up to specified fraction limits based on maximum sustained device bandwidth, independent of overall LD activity.

Bandwidth utilization for each LD is measured continuously based on current requests being serviced, plus the recent history of requests that have been completed.

Current requests being serviced are tracked by ReqCnt[LD] counters, with one counter per LD. The ReqCnt counter for an LD is incremented each time a request for that LD is received. The ReqCnt counter for an LD is decremented each time a response by that LD is transmitted. ReqCnt reflects instantaneous "committed" utilization, allowing the rapid reflection of incoming requests, especially when requests come in bursts.

The recent history of requests completed is tracked by CmpCnt[LD, Hist] registers, with one set of 16 Hist registers per LD. An architected configurable Completion Collection Interval control for the MLD determines the time interval over which transmitted responses are counted in the active (newest) Hist register/counter. At the end of each interval, the Hist register values for the LD are shifted from newer to older Hist registers, with the oldest value being discarded, and the active (newest) Hist register/counter being cleared. Further details on the hardware mechanism for CmpCnt[LD, Hist] are described in Section 3.3.4.3.10.

Controls for LD bandwidth management consist of per-LD sets of registers called QoS Allocation Fraction[LD] and QoS Limit Fraction[LD]. For each LD, QoS Allocation Fraction specifies the fraction of current device utilization allocated for the LD across all its QoS classes. QoS Limit Fraction for each LD specifies the fraction of maximum sustained device utilization as a fixed limit for the LD across all its QoS classes, independent of overall MLD activity.

Bandwidth utilization for each LD is based on the sum of its associated ReqCnt and CmpCnt[Hist] counters/registers. CmpCnt[Hist] reflects recently completed requests, and Completion Collection Interval controls how long this period of history covers (i.e., how quickly completed requests are "forgotten"). CmpCnt reflects recent utilization to help avoid overcompensating for bursts of requests.

Together, ReqCnt and CmpCnt[Hist] provide a simple, fair, and tunable way to compute average utilization. A shorter response history emphasizes instantaneous committed utilization, improving responsiveness. A longer response history smooths the average utilization, reducing overcompensation.

ReqCmpBasis is an architected control register that provides the basis for limiting each LD's utilization of the device, independent of overall MLD activity. Because ReqCmpBasis is compared against the sum of ReqCnt[ ] and CmpCnt[ ], its maximum value must be based on the maximum values of ReqCnt[ ] and CmpCnt[ ] summed across all configured LDs. The maximum value of Sum(ReqCnt[*]) is a function of the device's internal queuing and how many requests it can concurrently service. The maximum value of Sum(CmpCnt[*,*]) is a function of the device's maximum request service rate over the period of completion history recorded by CmpCnt[ ], which is directly influenced by the setting of Completion Collection Interval.

The FM programs ReqCmpBasis, the QoS Allocation Fraction array, and the QoS Limit Fraction array to control differentiated QoS between LDs. The FM is permitted to derate ReqCmpBasis below its maximum sustained estimate as a means of limiting power and heat dissipation.

To determine the DevLoad indication to return in each response, the device performs the following calculation:

```
Calculate TotalLoad = max(IntLoad[QoS], Egress Port Congestion state, Temporary
Throughput Reduction state);
```

```
Calculate ReqCmpTotal and populate ReqCmpCnt[LD] array element

    ReqCmpTotal = 0;

    For each LD

        ReqCmpCnt[LD] = ReqCnt[LD] + Sum(CmpCnt[LD, *]);

        ReqCmpTotal += ReqCmpCnt[LD];
```

IMPLEMENTATION NOTE

Avoiding Recalculation of ReqCmpTotal and ReqCmpCnt[ ] Array

ReqCmpCnt[ ] is an array that avoids having to recalculate its values later in the algorithm.

To avoid recalculating ReqCmpTotal and ReqCmpCnt[ ] array from scratch to determine the DevLoad indication to return in each response, it is strongly recommended that an implementation maintains these values on a running basis, only incrementally updating them as new requests arrive and responses are transmitted. The details are implementation specific.

IMPLEMENTATION NOTE

Calculating the Adjusted Allocation Bandwidth

When the MLD is overloaded, some LDs may be over their allocation while others are within their allocation.

- Those LDs under their allocation (especially inactive LDs) contribute to a "surplus" of bandwidth that can be distributed across active LDs that are above their allocation.
- Those LDs over their allocation claim "their fair share" of that surplus based on their allocation, and the load value for these LDs is based on an "adjusted allocated bandwidth" that includes a prorated share of the surplus.

This adjusted allocation bandwidth algorithm avoids anomalies that otherwise occur when some LDs are using well below their allocation, especially if they are idle.

In subsequent algorithms, certain registers have integer and fraction portions, optimized for implementing the algorithms in dedicated hardware. The integer portion is described as being 16 bits unsigned, although it is permitted to be smaller or larger as needed by the specific implementation. It must be sized such that it will never overflow during normal operation. The fractional portion must be 8 bits. These registers are indicated by their name being in italics.

IMPLEMENTATION NOTE

Registers with Integer and Fraction Portions

These registers can hold the product of a 16-bit unsigned integer and an 8-bit fraction, resulting in 24 bits with the radix point being between the upper 16 bits and the lower 8 bits. Rounding to an integer is readily accomplished by adding 0000.80h (0.5 decimal) and truncating the lower 8 bits.

```
If TotalLoad is Moderate Overload or Severe Overload, calculate the adjusted
```

```
allocated bandwidth:
    ClaimAllocTotal = 0;
    SurplusTotal = 0;
    For each LD
            AllocCnt = QoS Allocation Fraction[LD] * ReqCmpTotal;
            If this LD is the (single) LD associated with the response
                AllocCntSaved = AllocCnt;
            If ReqCmpCnt[LD] > AllocCnt then
                ClaimAllocTotal += AllocCnt;
            Else
                SurplusTotal += AllocCnt - ReqCmpCnt[LD];
    For the single LD associated with the response
        If ReqCmpCnt[LD] > (AllocCntSaved + AllocCntSaved * SurplusTotal /
ClaimAllocTotal) then LD is over its adjusted allocated bandwidth;   // Use this
result in the subsequent table
```

IMPLEMENTATION NOTE

Determination of an LD Being Above its Adjusted Allocated Bandwidth

The preceding equation requires a division, which is relatively expensive to implement in hardware dedicated for this determination. To enable hardware making this determination more efficiently, the following derived equivalent equation is strongly recommended:

ReqCmpCnt[LD] > (*AllocCntSaved* + *AllocCntSaved* * *SurplusTotal* / *ClaimAllocTotal*)

(ReqCmpCnt[LD] * *ClaimAllocTotal*) > (*AllocCntSaved* * *ClaimAllocTotal* + *AllocCntSaved* * *SurplusTotal*)

(ReqCmpCnt[LD] * *ClaimAllocTotal*) > (*AllocCntSaved* * (*ClaimAllocTotal* + *SurplusTotal*))

```
// Perform the bandwidth limit calculation for this LD

If ReqCmpCnt[LD] >  QoS Limit Fraction [LD] * ReqCmpBasis then LD is over its limit
BW;
```

Table 3-32. Additional Factors for Determining DevLoad in MLDs (Sheet 1 of 2)

TotalLoad	LD over Limit BW?	LD over Adjusted Allocated BW?	Returned DevLoad Indication
Light Load or Optimal Load	No	-	TotalLoad
	Yes	-	Moderate Overload

Table 3-32. Additional Factors for Determining DevLoad in MLDs (Sheet 2 of 2)

TotalLoad	LD over Limit BW?	LD over Adjusted Allocated BW?	Returned DevLoad Indication
Moderate Overload	No	No	Optimal Load
	No	Yes	Moderate Overload
	Yes	-	Moderate Overload
Severe Overload	-	No	Moderate Overload
	-	Yes	Severe Overload

The preceding table is based on the following policies for LD bandwidth management:

- The LD is always subject to its QoS Limit Fraction
- For TotalLoad indications of Light Load or Optimal Load, the LD can exceed its QoS Allocation Fraction, up to its QoS Limit Fraction
- For TotalLoad indications of Moderate Overload or Severe Overload, LDs with loads up to QoS Allocation Fraction get throttled less than LDs with loads that exceed QoS Allocation Fraction

3.3.4.3.8 DevLoad Indication by Multi-QoS and Single-QoS GFDs

DevLoad indication for GFDs is similar to that for MLDs, with the exception that 12-bit GFD host/peer requester PIDs (RPIDs) scale much higher than the 4-bit LDs for MLDs, and the QoS Allocation Fraction mechanism (based on current device utilization) is not supported for GFDs due to architectural scaling challenges. However, the QoS Limit Fraction mechanism (based on a fixed maximum sustained device utilization) is supported for GFDs, and architected controls specify the fraction limits.

Bandwidth utilization for each RPID is measured continuously based on current requests being serviced, plus the recent history of requests that have been completed.

Current requests that are being serviced are tracked by ReqCnt[RPID] counters, with one counter per RPID. The ReqCnt counter for an RPID is incremented each time a request from that RPID is received. The ReqCnt counter for an RPID is decremented each time a response to that RPID is transmitted. ReqCnt reflects instantaneous "committed" utilization, allowing the rapid reflection of incoming requests, especially when requests come in bursts.

The recent history of requests completed is tracked by CmpCnt[RPID, Hist] registers, with one set of 16 Hist registers per RPID. An architected configurable Completion Collection Interval control for the GFD determines the time interval over which transmitted responses are counted in the active (newest) Hist register/counter. At the end of each interval, the Hist register values for the RPID are shifted from newer to older Hist registers, with the oldest value being discarded, and the active (newest) Hist register/counter being cleared. Further details on the hardware mechanism for CmpCnt[RPID, Hist] are described in Section 3.3.4.3.10.

Controls for RPID bandwidth management consist of per-RPID sets of registers called QoS Limit Fraction[RPID]. QoS Limit Fraction for each RPID specifies the fraction of maximum sustained device utilization as a fixed limit for the RPID across all its QoS classes, independent of overall GFD activity.

Bandwidth utilization for each RPID is based on the sum of its associated ReqCnt and CmpCnt[Hist] counters/registers. CmpCnt[Hist] reflects recently completed requests, and Completion Collection Interval controls how long this period of history covers (i.e., how quickly completed requests are "forgotten"). CmpCnt reflects recent utilization to help avoid overcompensating for bursts of requests.

Together, ReqCnt and CmpCnt[Hist] provide a simple, fair, and tunable way to compute average utilization. A shorter response history emphasizes instantaneous committed utilization, thus improving responsiveness. A longer response history smooths the average utilization, thus reducing overcompensation.

ReqCmpBasis is an architected control register that provides the basis for limiting each RPID's utilization of the device, independent of overall GFD activity. Because ReqCmpBasis is compared against the sum of ReqCnt[ ] and CmpCnt[ ], its maximum value must be based on the maximum values of ReqCnt[ ] and CmpCnt[ ] summed across all configured RPIDs. The maximum value of Sum(ReqCnt[*]) is a function of the device's internal queuing and how many requests it can concurrently service. The maximum value of Sum(CmpCnt[*,*]) is a function of the device's maximum request service rate over the period of completion history recorded by CmpCnt[ ], which is directly influenced by the setting of Completion Collection Interval.

The FM programs ReqCmpBasis and the QoS Limit Fraction array to control differentiated QoS between RPIDs. The FM is permitted to derate ReqCmpBasis below its maximum sustained estimate as a means of limiting power and heat dissipation.

To determine the DevLoad indication to return in each response, the device performs the following calculation:

```
Calculate TotalLoad = max(IntLoad[QoS], Egress Port Congestion state, Temporary
Throughput Reduction state);

// Perform the bandwidth limit calculation for this RPID

If ReqCmpCnt[RPID] > QoS Limit Fraction[RPID] * ReqCmpBasis then the RPID is over
its limit BW;
```

Table 3-33. Additional Factors for Determining DevLoad in MLDs/GFDs

TotalLoad	RPID over Limit BW?	Returned DevLoad Indication
Light Load or Optimal Load	No	TotalLoad
	Yes	Moderate Overload
Moderate Overload	No	Moderate Overload
	Yes	Severe Overload
Severe Overload	-	Severe Overload

Table 3-33 is based on the following policies for RPID bandwidth management:

- The RPID is always subject to its QoS Limit Fraction
- For TotalLoad indications of Moderate Overload, RPIDs with loads up to QoS Limit Fraction get throttled less than RPIDs with loads that exceed QoS Limit Fraction

3.3.4.3.9 Egress Port Congestion Measurement Mechanism

This hardware mechanism measures the average egress port congestion on a rolling percentage basis.

FCBP (Flow Control Backpressured): this binary condition indicates the instantaneous state of the egress port. It is true if the port has messages or flits available to transmit but is unable to transmit any of them due to a lack of suitable flow control credits.

Backpressure Sample Interval register: this architected control register specifies the fixed interval in nanoseconds at which FCBP is sampled. It has a range of 0-31. One hundred samples are recorded, so a setting of 1 yields 100 ns of history. A setting of 31 yields 3.1 us of history. A setting of 0 disables the measurement mechanism, and it must indicate an average congestion percentage of 0.

BPhist[100] bit array: this stores the 100 most-recent FCBP samples. It is not accessible by software.

Backpressure Average Percentage: when this architected status register is read, it indicates the current number of Set bits in BPhist[100]. It ranges in value from 0 to 100.

The actual implementation of BPhist[100] and Backpressure Average Percentage is device specific. Here is a possible implementation approach:

- BPhist[100] is a shift register
- Backpressure Average Percentage is an up/down counter
- With each new FCBP sample:
  - If the new sample (not yet in BPhist) and the oldest sample in BPhist are both 0 or both 1, no change is made to Backpressure Average Percentage.
  - If the new sample is 1 and the oldest sample is 0, increment Backpressure Average Percentage.
  - If the new sample is 0 and the oldest sample is 1, decrement Backpressure Average Percentage.
- Shift BPhist[100], discarding the oldest sample and entering the new sample

3.3.4.3.10 Recent Transmitted Responses Measurement Mechanism

This hardware mechanism measures the number of recently transmitted responses on a per-host/peer basis in the most recent 16 intervals of a configured time period. Hosts are identified by a Requester ID (ReqID), which is the LD-ID for MLDs and the RPID for GFDs.

Completion Collection Interval register: this architected control register specifies the interval over which transmitted responses are counted in an active Hist register. It has a range is 0-127. A setting of 1 yields 16 nanoseconds of history. A setting of 127 yields about 2 us of history. A setting of 0 disables the measurement mechanism, and it must indicate a response count of 0.

CmpCnt[ReqID, 16] registers: these registers track the total of recent transmitted responses on a per-host/peer basis. CmpCnt[ReqID, 0] is a counter and is the newest value, while CmpCnt[ReqID, 1:15] are registers. These registers are not directly visible to software.

For each ReqID, at the end of each Completion Collection Interval:

- The 16 CmpCnt[ReqID, *] register values are shifted from newer to older
- The CmpCnt[ReqID, 15] Hist register value is discarded
- The CmpCnt[ReqID, 0] register is cleared and it is armed to count transmitted responses in the next interval

3.3.5 M2S Request (Req)

The Req message class generically contains reads, invalidates, and signals going from the Master to the Subordinate.

Table 3-34. M2S Request Fields (Sheet 1 of 2)

Field	Width (Bits)			Description
	68B Flit	256B Flit	PBR Flit	
Valid	1			The valid signal indicates that this is a valid request
MemOpcode	4			Memory Operation: This specifies which, if any, operation needs to be performed on the data and associated information. Details in Table 3-35.
SnpType	3			Snoop Type: This specifies what snoop type, if any, needs to be issued by the DCOH and the minimum coherency state required by the Host. Details in Table 3-38. This field is used to indicate the Length Index for the TEUpdate opcode.
MetaField	2			Metadata Field: Up to 3 Metadata Fields can be addressed. This specifies which, if any, Metadata Field needs to be updated. Details of Metadata Field in Table 3-36. If the Subordinate does not support memory with Metadata, this field will still be used by the DCOH for interpreting Host commands as described in Table 3-37.
MetaValue	2			Metadata Value: When MetaField is not No-Op, this specifies the value to which the field needs to be updated. Details in Table 3-37. If the Subordinate does not support memory with Metadata, this field will still be used by the device coherence engine for interpreting Host commands as described in Table 3-37. For the TEUpdate message, this field carries the TE state change value where 00b is TE cleared and 01b is TE set.
Tag	16			The Tag field is used to specify the source entry in the Master which is pre-allocated for the duration of the CXL.mem transaction. This value needs to be reflected with the response from the Subordinate so the response can be routed appropriately. The exceptions are the MemRdFwd and MemWrFwd opcodes as described in Table 3-35. **Note:** The Tag field has no explicit requirement to be unique.
Address[5]	1	0		Address[5] is provisioned for future usages such as critical chunk first for 68B flit, but this is not included in a 256B flit.
Address[51:6]	46			This field specifies the Host Physical Address associated with the MemOpcode.
LD-ID[3:0]	4		0	Logical Device Identifier: This identifies a Logical Device within a Multiple-Logical Device. Not applicable in PBR mode where SPID infers this field.
SPID	0		12	Source PID
DPID	0		12	Destination PID
CKID	0	13		Context Key ID: Optional key ID that references preconfigured key material utilized for device-based data-at-rest encryption. If the device has been configured to utilize CKID-based device encryption and locked utilizing the CXL Trusted Execution Environment (TEE) Security Protocol (TSP), then this field shall be valid for Data Read access types (MemRd/MemRdTEE/ MemRdData*/MemSpecRd*) and treated as reserved for other messages.

Table 3-34. M2S Request Fields (Sheet 2 of 2)

Field	Width (Bits)			Description
	68B Flit	256B Flit	PBR Flit	
RSVD	6	7		Reserved
TC	2			Traffic Class: This can be used by the Master to specify the Quality of Service associated with the request. This is reserved for future usage.
Total	**87**	**100**	**120**	

Table 3-35. M2S Req Memory Opcodes (Sheet 1 of 2)

Opcode	Description	Encoding
MemInv	Invalidation request from the Master. Primarily for Metadata updates. No data read or write required. If SnpType field contains valid commands, perform required snoops.	0000b
MemRd	Normal memory data read operation. If MetaField contains valid commands, perform Metadata updates. If SnpType field contains valid commands, perform required snoops.	0001b
MemRdData	Normal Memory data read operation. MetaField has no impact on the coherence state. MetaValue is to be ignored. Instead, update Meta0-State as follows: If initial Meta0-State value = 'I', update Meta0-State value to 'A' Else, no update required If SnpType field contains valid commands, perform required snoops. MetaField encoding of Extended Meta-State (EMS) follows the rules for it in Table 3-36.	0010b
MemRdFwd	This is an indication from the Host that data can be directly forwarded from device-attached memory to the device without any completion to the Host. This is only sent as a result of a CXL.cache D2H read request to device-attached memory that is mapped as HDM-D. The Tag field contains the reflected CQID sent along with the D2H read request. The SnpType is always No-Op for this Opcode. The caching state of the line is reflected in the Meta0-State value. **Note:** This message is not sent to devices that have device-attached memory that is mapped only as HDM-H or HDM-DB.	0011b
MemWrFwd	This is an indication from the Host to the device that it owns the line and can update it without any completion to the Host. This is only sent as a result of a CXL.cache D2H write request to device-attached memory that is mapped as HDM-D. The Tag field contains the reflected CQID sent along with the D2H write request. The SnpType is always No-Op for this Opcode. The caching state of the line is reflected in the Meta0-State value. **Note:** This message is not sent to devices that have device-attached memory that is mapped only as HDM-H or HDM-DB.	0100b
MemRdTEE[1]	Same as MemRd but with the Trusted Execution Environment (TEE) attribute. See Section 11.5.4.5 for description of TEE attribute handling.	0101b
MemRdDataTEE[1]	Same as MemRdData but with the Trusted Execution Environment (TEE) attribute. See Section 11.5.4.5 for description of TEE attribute handling.	0110b
MemInvTEE	Same as MemInv but with the Trusted Execution Environment (TEE) attribute. See Section 11.5.4.5 for description of TEE attribute handling.	0111b
MemSpecRd	Memory Speculative Read is issued to start a memory access before the home agent has resolved coherence to reduce access latency. This command does not receive a completion message. The Tag, MetaField, MetaValue, and SnpType are reserved. See Section 3.5.3.1 for a description of the use case.	1000b
MemInvNT	This is similar to the MemInv command except that the NT is a hint that indicates the invalidation is non-temporal and the writeback is expected soon. However, this is a hint and not a guarantee. If the target is locked utilizing TSP, the target shall decode this opcode as MemInvP. If the target is not locked, the target shall decode this opcode as MemInvNT. See Section 11.5 for TSP.	1001b
MemInvP	Memory invalidation with precise TE State. If the target is locked utilizing TSP, the target shall decode this opcode as MemInvP. If the target is not locked, the target shall decode this opcode as MemInvNT. See Section 11.5 for TSP.	

Table 3-35. M2S Req Memory Opcodes (Sheet 2 of 2)

Opcode	Description	Encoding
MemClnEvct	Memory Clean Evict is a message that is similar to MemInv, but intent to indicate host going to I-state and does not require Meta0-State return. This message is supported only to the HDM-DB address region.	1010b
MemInvPTEE	Same as MemInvP but with the Trusted Execution Environment (TEE) attribute. See Section 11.5.4.5 for description of TEE attribute handling.	1011b
MemSpecRdTEE[1]	Same as MemSpecRd but with Trusted Execution Environment (TEE) attribute. See Section 11.5.4.5 for description of TEE attribute handling.	1100b
TEUpdate[1]	Update of the TE state for the memory region. The memory region update is defined by the length-index field (passed in SnpType bits). The lower address bits in the message may be set to allow routing of the message to reach the correct interleave set target; however, the lower bits are masked to the natural alignment of the length when updating TE state. The MetaValue field defines the new TE state that supports 00b to clear and 01b to set. See details of the use of this message in Section 11.5.4.5.3.	1101b
MemClnEvctTEE	Same as MemClnEvct but with the Trusted Execution Environment (TEE) attribute. See Section 11.5.4.5 for description of TEE attribute handling.	1110b
MemClnEvctU	Same as MemClnEvct but TE State is not conveyed and assumed to be unknown.	1111b

1. Supported only in 256B and PBR Flit messages and considered Reserved in 68B Flit messages.

Table 3-36. Metadata Field Definition

MetaField	Description	Encoding
Meta0-State	Update the Metadata bits with the value in the Metadata Value field. Details of MetaValue associated with Meta0-State in Table 3-37.	00b
Extended Meta-State (EMS)	This encoding has different interpretation in different channels: • M2S Req usage indicates that the request requires the Extended MetaValue to be returned from the device in the response unless an error condition occurs. • M2S RwD and S2M DRS use this to indication that the Extended MetaValue is attached to the message as a Trailer. This size of the MetaValue is configurable up to 32 bits. • Other channels do not use this encoding and it should be considered Reserved. For HDM-DB, the MetaValue is defined in Table 3-37 for coherence resolution, Reserved for HDM-H. This encoding is not used for HDM-D.	01b
Reserved	Reserved	10b
No-Op	No Metadata operation. The MetaValue field is Reserved. For NDR/DRS messages that would return Metadata, this encoding can be used in case of an error in Metadata storage (standard 2-bits or EMD) or if the device does not store Metadata.	11b

Table 3-37. Meta0-State Value Definition (HDM-D/HDM-DB Devices)[1]

Description	Encoding
Invalid (I): Indicates the host does not have a cacheable copy of the line. The DCOH can use this information to grant exclusive ownership of the line to the device. **Note:** When paired with a MemOpcode = MemInv and SnpType = SnpInv, this is used to communicate that the device should flush this line from its caches, if cached, to device-attached memory resulting in all caches ending in I.	00b
Explicit No-Op: Used only when MetaField is Extended Meta-State in HDM-DB requests to indicate that a coherence state update is not requested. For all other cases this is considered a Reserved.	01b
Any (A): Indicates the host may have a shared, exclusive, or modified copy of the line. The DCOH can use this information to interpret that the Host likely wants to update the line and the device should not be given a copy of the line without resolving coherence with the host using the flow appropriate for the memory type.	10b
Shared (S): Indicates the host may have at most a shared copy of the line. The DCOH can use this information to interpret that the Host does not have an exclusive or modified copy of the line. If the device wants a shared or current copy of the line, the DCOH can provide this without informing the Host. If the device wants an exclusive copy of the line, the DCOH must resolve coherence with the Host using the flow appropriate for the memory type.	11b

1. HDM-H use case in Type 3 devices have Meta0-State definition that is host specific, so the definition in this table does not apply for the HDM-H address region in devices.

Table 3-38. Snoop Type Definition

SnpType Description	Description	Encoding
No-Op	No snoop needs to be performed	000b
SnpData	Snoop may be required - the requester needs at least a Shared copy of the line. Device may choose to give an exclusive copy of the line as well.	001b
SnpCur	Snoop may be required - the requester needs the current value of the line. Requester guarantees the line will not be cached. Device need not change the state of the line in its caches, if present.	010b
SnpInv	Snoop may be required - the requester needs an exclusive copy of the line.	011b
Reserved	Reserved	1xxb

Valid uses of M2S request semantics are described in Table 3-39 but are not the complete set of legal flows. For a complete set of legal combinations, see Appendix C.

Table 3-39. M2S Req Usage (Sheet 1 of 2)

M2S Req	MetaField	Meta Value	SnpType	S2M NDR	S2M DRS	Description
MemRd	Meta0-State	A	SnpInv	Cmp-E	MemData	The Host wants an exclusive copy of the line
MemRd	Meta0-State	S	SnpData	Cmp-S or Cmp-E	MemData	The Host wants a shared copy of the line
MemRd	No-Op	N/A[1]	SnpCur	Cmp	MemData	The Host wants a non-cacheable but current value of the line
MemRd	No-Op	N/A[1]	SnpInv	Cmp	MemData	The Host wants a non-cacheable value of the line and the device should invalidate the line from its caches
MemInv	Meta0-State	A	SnpInv	Cmp-E	N/A	The Host wants ownership of the line without data
MemInvNT	Meta0-State	A	SnpInv	Cmp-E	N/A	The Host wants ownership of the line without data. However, the Host expects this to be non-temporal and may do a writeback soon.

Table 3-39. M2S Req Usage (Sheet 2 of 2)

M2S Req	MetaField	Meta Value	SnpType	S2M NDR	S2M DRS	Description
MemInv	Meta0-State	I	SnpInv	Cmp	N/A	The Host wants the device to invalidate the line from its caches
MemRdData	No-Op	N/A[1]	SnpData	Cmp-S or Cmp-E	MemData	The Host wants a cacheable copy in either exclusive or shared state
MemClnEvct	Meta0-State	I	No-Op	Cmp	N/A	Host is dropping E or S state from its cache and leaving the line in I-state. This message allows the Device to clean the Snoop Filter (or BIAS table).

1. N/A in the MetaValue indicates that the entire field is considered Reserved (cleared to 0 by sender and ignored by receiver).

3.3.6 M2S Request with Data (RwD)

The Request with Data (RwD) message class generally contains writes from the Master to the Subordinate.

Table 3-40. M2S RwD Fields (Sheet 1 of 2)

Field	Width (Bits)			Description
	68B Flit	256B Flit	PBR Flit	
Valid	1			The valid signal indicates that this is a valid request.
MemOpcode	4			Memory Operation: This specifies which, if any, operation needs to be performed on the data and associated information. Details in Table 3-41.
SnpType	3			Snoop Type: This specifies what snoop type, if any, needs to be issued by the DCOH and the minimum coherency state required by the Host. Details in Table 3-38.
MetaField	2			Metadata Field: Up to 3 Metadata Fields can be addressed. This specifies which, if any, Metadata Field needs to be updated. Details of Metadata Field in Table 3-36. If the Subordinate does not support memory with Metadata, this field will still be used by the DCOH for interpreting Host commands as described in Table 3-37.
MetaValue	2			Metadata Value: When MetaField is not No-Op, this specifies the value the field needs to be updated to. Details in Table 3-37. If the Subordinate does not support memory with Metadata, this field will still be used by the device coherence engine for interpreting Host commands as described in Table 3-37.
Tag	16			The Tag field is used to specify the source entry in the Master which is pre-allocated for the duration of the CXL.mem transaction. This value needs to be reflected with the response from the Subordinate so the response can be routed appropriately. For BIConflict, the tag encoding must use the same value as the pending M2S Req message (if one exists) which the BISnp found to be in conflict. This requirement is necessary to use Tag for fabric ordering of S2M NDR (Cmp* and BIConflictAck ordering for same tag). **Note:** The Tag field has no explicit requirement to be unique.
Address[51:6]	46			This field specifies the Host Physical Address associated with the MemOpcode.
Poison	1			The Poison bit indicates that the data contains an error. The handling of poisoned data is device specific. See Chapter 12.0 for more details.

Table 3-40. M2S RwD Fields (Sheet 2 of 2)

Field	Width (Bits)			Description
	68B Flit	256B Flit	PBR Flit	
TRP (formerly BEP)	0	1		Trailer Present: Indicates that a trailer is included on the message. The trailer size for RwD is defined in Table 3-43. The trailer is observed in the Link Layer as a G-Slot following a 64B data payload. The baseline requirement for this bit is to enable only Byte Enables for partial writes (MemWrPtl). This bit is also optionally extended for Extend-Metadata indication. **Note:** This bit was formerly referred to as Byte-Enables Present (BEP), but has been redefined as part of an optional extension to support message trailers.
LD-ID[3:0]	4		0	Logical Device Identifier: This identifies a logical device within a multiple-logical device. Not applicable in PBR messages where SPID infers this field.
SPID	0		12	Source PID
DPID	0		12	Destination PID
CKID	0	13		Context Key ID: Optional key ID that references preconfigured key material utilized for device-based data-at-rest encryption. If the device has been configured to utilize CKID-based device encryption and locked utilizing the CXL Trusted Execution Environment (TEE) Security Protocol (TSP), then this field shall be valid for accesses that carry a non-reserved payload or cause a memory read to occur (MemWr*, MemRdFill*) and reserved for other cases (BIConflict).
RSVD	6	9		Reserved
TC	2			Traffic Class: This can be used by the Master to specify the Quality of Service associated with the request. This is reserved for future usage.
Total	**87**	**104**	**124**	

Table 3-41. M2S RwD Memory Opcodes (Sheet 1 of 2)

Opcode	Description	Encoding
MemWr	Memory write command. Used for full cacheline writes. If MetaField contains valid commands, perform Metadata updates. If SnpType field contains valid commands, perform required snoops. If the snoop hits a Modified cacheline in the device, the DCOH will invalidate the cache and write the data from the Host to device-attached memory.	0001b
MemWrPtl	Memory Write Partial. Contains 64 byte enables, one for each byte of data. If MetaField contains valid commands, perform Metadata updates. If SnpType field contains valid commands, perform required snoops. If the snoop hits a Modified cacheline in the device, the DCOH will need to perform a merge, invalidate the cache, and write the contents back to device-attached memory. **Note:** This command cannot be used with host-side memory encryption unless byte-enable encodings are aligned with encryption boundaries (32B aligned is an example which may be allowed).	0010b
BIConflict	Part of conflict flow for BISnp indicating that the host observed a conflicting coherent request to the same cacheline address. See Section 3.5.1 for details. This message carries a 64B payload as required by the RwD channel, but the payload bytes are reserved (cleared to all 0s). This message is sent on the RwD channel because the dependence rules on this channel allow for a low-complexity flow from a deadlock-avoidance point of view.	0100b

Table 3-41. M2S RwD Memory Opcodes (Sheet 2 of 2)

Opcode	Description	Encoding
MemRdFill[1]	This is a simple read command equivalent to MemRd but never changes coherence state (MetaField=No-Op, SnpType=No-Op). The use of this command is intended for partial write data that is merging in the host with host-side encryption. With host-side encryption, it is not possible to merge partial data in the device as an attribute of the way encryption works. This message carries a 64B payload as required by the RwD channel; however, the payload bytes are reserved (i.e., cleared to all 0s). This message is sent on the RwD channel because the dependence rules on this channel allow for a low-complexity flow from a deadlock-avoidance point of view.	0101b
MemWrTEE[1]	Same as MemWr but with the Trusted Execution Environment (TEE) attribute. See Section 11.5.4.5 for description of TEE attribute handling.	1001b
MemWrPtlTEE[1]	Same as MemWrPtl but with the Trusted Execution Environment (TEE) attribute. See Section 11.5.4.5 for description of TEE attribute handling.	1010b
MemRdFillTEE[1]	Same as MemRdFill but with the Trusted Execution Environment (TEE) attribute. See Section 11.5.4.5 for description of TEE attribute handling.	1101b
Reserved	Reserved	<Others>

1. Supported only in 256B and PBR Flit messages and considered reserved in 68B Flit messages.

The definition of other fields are consistent with M2S Req (see Section 3.3.12). Valid uses of M2S RwD semantics are described in Table 3-42 but are not complete set of legal flows. For a complete set of legal combinations, see Appendix C.

Table 3-42. M2S RwD Usage

M2S RwD	MetaField	Meta Value	SnpType	S2M NDR	Description
MemWr	Meta0-State	I	No-Op	Cmp	The Host wants to write the cacheline back to memory and does not retain a cacheable copy.
MemWr	Meta0-State	A	No-Op	Cmp	The Host wants to write the cacheline back to memory and retains a cacheable copy in shared, exclusive or modified state.
MemWr	Meta0-State	I	SnpInv	Cmp	The Host wants to write the cacheline to memory and does not retain a cacheable copy. In addition, the Host did not get ownership of the cacheline before doing this write and needs the device to snoop-invalidate its caches before performing the writeback to memory.
MemWrPtl	Meta0-State	I	SnpInv	Cmp	Same as the above row except the data being written is partial and the device needs to merge the data if it finds a copy of the cacheline in its caches.

3.3.6.1 Trailer Present for RwD (256B Flit)

In 256B Flit mode, a Trailer Present bit (TRP; formerly BEP, Byte-Enables Present) bit is included with the message header that indicates whether a Trailer slot is included at the end of the message. The trailer can be up to 96 bits.

Byte Enables field is 64 bits wide and indicates which of the bytes are valid for the contained data.

The Extended Metadata (EMD) trailer can be up to 32 bits. Section 8.2.4.31 describes the registers that aid in discovery of device's EMD capability and EMD related configuration of the device. The mechanism for discovering the host's EMD capabilities and EMD related configuration of the host is host-specific. The host and the device must be configured in a consistent manner.

Table 3-43. RwD Trailers

Opcode/ Message	MetaField	TRP	Trailer Size Required	Description
MemWr/ MemWrTEE	EMS	1	32 bits	Trailer bits[31:0] defined as EMD.
	No-OP/MS0	0	No Trailer	
MemWrPtl/ MemWrPtlTEE	EMS	1	96 bits	Trailer bits[63:0] defined as Byte Enables. Trailer bits[95:64] defined as EMD.
	No-Op/MS0		64 bits	Trailer bits[63:0] defined as Byte Enables.
<Others>	N/A	0	No Trailer	Other combinations do not encode trailers.

3.3.7 M2S Back-Invalidate Response (BIRsp)

The Back-Invalidate Response (BIRsp) message class contains response messages from the Master to the Subordinate as a result of Back-Invalidate Snoops. This message class is not supported in 68B Flit mode.

Table 3-44. M2S BIRsp Fields

Field	Width (Bits)			Description
	68B Flit	256B Flit	PBR Flit	
Valid	N/A	1		The valid signal indicates that this is a valid response.
Opcode		4		Response type with encodings in Table 3-45.
BI-ID		12	0	BI-ID of the device that is the destination of the message. See Section 9.14 for details on how this field is assigned to devices. Not applicable in PBR messages where DPID infers this field.
BITag		12		Tracking ID from the device.
LowAddr		2		The lower 2 bits of Cacheline address (Address[7:6]). This is needed to differentiate snoop responses when a Block Snoop is sent and receives snoop response for each cacheline. For block response (opcode names *Blk), this field is reserved.
SPID		0	12	Source PID
DPID		0	12	Destination PID
RSVD		9		
Total		**40**	**52**	

Table 3-45. M2S BIRsp Memory Opcodes (Sheet 1 of 2)

Opcode	Description	Encoding
BIRspI	Host completed the Back-Invalidate Snoop for one cacheline and the host cache state is I.	0000b
BIRspS	Host completed the Back-Invalidate Snoop for one cacheline and the host cache state is S.	0001b
BIRspE	Host completed the Back-Invalidate Snoop for one cacheline and the host cache state is E.	0010b
BIRspIBlk	Same as BIRspI except that the message applies to the entire block of cachelines. The size of the block is explicit in the BISnp*Blk message for which this is a response.	0100b

Table 3-45. M2S BIRsp Memory Opcodes (Sheet 2 of 2)

Opcode	Description	Encoding
BIRspSBlk	Same as BIRspS except that the message applies to the entire block of cachelines. The size of the block is explicit in the BISnp*Blk message for which this is a response.	0101b
BIRspEBlk	Same as BIRspE except that the message applies to the entire block of cachelines. The size of the block is explicit in the BISnp*Blk message for which this is a response.	0110b
Reserved	Reserved	<Others>

3.3.8 S2M Back-Invalidate Snoop (BISnp)

The Back-Invalidate Snoop (BISnp) message class contains Snoop messages from the Subordinate to the Master. This message class is not supported in 68B Flit mode.

Table 3-46. S2M BISnp Fields

Field	Width (Bits)			Description
	68B Flit	256B Flit	PBR Flit	
Valid	N/A	1		The valid signal indicates that this is a valid request.
Opcode		4		Snoop type with encodings in Table 3-47.
BI-ID		12	0	BI-ID of the device that issued the message. See Section 9.14 for details on how this field is assigned. Not applicable in PBR messages where SPID infers this field.
BITag		12		Tracking ID from the device.
Address[51:6]		46		Host Physical Address. For *Blk opcodes, the lower 2 bits (Address[7:6]) are encoded as defined in Table 3-48. Used for all other opcodes that represent the standard definition of Host Physical Address.
SPID		0	12	Source PID
DPID		0	12	Destination PID
RSVD		9		
Total		**84**	**96**	

Table 3-47. S2M BISnp Opcodes (Sheet 1 of 2)

Opcode	Description	Encoding
BISnpCur	Device requesting Current copy of the line but not requiring caching state.	0000b
BISnpData	Device requesting Shared or Exclusive copy.	0001b
BISnpInv	Device requesting Exclusive Copy.	0010b
BISnpCurBlk	Same as BISnpCur except covering 2 or 4 cachelines that are naturally aligned and contiguous. The Block Enable encoding is in Address[7:6] and defined in Table 3-48. The host may give per cacheline response or a single block response applying to all cachelines in the block. More details are in Section 3.3.8.1.	0100b
BISnpDataBlk	Same as BISnpData except covering 2 or 4 cachelines that are naturally aligned and contiguous. The Block Enable encoding is in Address[7:6] and defined in Table 3-48. The host may give per cacheline response or a single block response applying to all cachelines in the block. More details are in Section 3.3.8.1.	0101b

Table 3-47. S2M BISnp Opcodes (Sheet 2 of 2)

Opcode	Description	Encoding
BISnpInvBlk	Same as BISnpInv except covering 2 or 4 cachelines that are naturally aligned and contiguous. The Block Enable encoding is in Address[7:6] and defined in Table 3-48. The host may give per cacheline response or a single block response applying to all cachelines in the block. More details are in Section 3.3.8.1.	0110b
BISnpCurTEE	Same as BISnpCur but with the Trusted Execution Environment (TEE) attribute. See Section 11.5.4.5 for description of TEE attribute handling.	1000b
BISnpDataTEE	Same as BISnpData but with the Trusted Execution Environment (TEE) attribute. See Section 11.5.4.5 for description of TEE attribute handling.	1001b
BISnpInvTEE	Same as BISnpInv but with the Trusted Execution Environment (TEE) attribute. See Section 11.5.4.5 for description of TEE attribute handling.	1010b
BISnpCurBlkTEE	Same as BISnpCurBlk but with the Trusted Execution Environment (TEE) attribute. See Section 11.5.4.5 for description of TEE attribute handling.	1100b
BISnpDataBlkTEE	Same as BISnpDataBlk but with the Trusted Execution Environment (TEE) attribute. See Section 11.5.4.5 for description of TEE attribute handling.	1101b
BISnpInvBlkTEE	Same as BISnpInvBlk but with the Trusted Execution Environment (TEE) attribute. See Section 11.5.4.5 for description of TEE attribute handling.	1110b
Reserved	Reserved	<Others>

3.3.8.1 Rules for Block Back-Invalidate Snoops

A Block Back-Invalidate Snoop applies to multiple naturally aligned contiguous cachelines (2 or 4 cachelines). The host must ensure that coherence is resolved for each line and may send combined or individual responses for each in arbitrary order. In the presence of address conflicts, it is necessary that the host resolve conflicts for each cacheline separately. This special address encoding applies only to BISnp*Blk messages.

Table 3-48. Block (Blk) Enable Encoding in Address[7:6]

Addr[7:6]	Description
00b	Reserved
01b	Lower 128B block is valid, Lower is defined as Address[7]=0
10b	Upper 128B block, Upper is defined as Address[7]=1
11b	256B block is valid

3.3.9 S2M No Data Response (NDR)

The NDR message class contains completions and indications from the Subordinate to the Master.

Table 3-49. S2M NDR Fields

Field	Width (Bits)			Description
	68B Flit	256B Flit	PBR Flit	
Valid	1			The valid signal indicates that this is a valid request.
Opcode	3			Memory Operation: This specifies which, if any, operation needs to be performed on the data and associated information. Details in Table 3-50.
MetaField	2			Metadata Field: For devices that support memory with Metadata, this field may be encoded with Meta0-State in response to an M2S Req. For devices that do not support memory with Metadata or in response to an M2S RwD, this field must be set to the No-Op encoding. No-Op may also be used by devices if the Metadata is unreliable or corrupted in the device.
MetaValue	2			Metadata Value: If MetaField is No-Op, this field is don't care; otherwise, it is Metadata Field as read from memory.
Tag	16			Tag: This is a reflection of the Tag field sent with the associated M2S Req or M2S RwD.
LD-ID[3:0]	4		0	Logical Device Identifier: This identifies a logical device within a multiple-logical device. Not applicable in PBR messages where DPID infers this field.
DevLoad	2			Device Load: Indicates device load as defined in Table 3-51. Values are used to enforce QoS as described in Section 3.3.4.
DPID	0		12	Destination PID
RSVD	0	10		
Total	**30**	**40**	**48**	

Opcodes for the NDR message class are defined in Table 3-50.

Table 3-50. S2M NDR Opcodes

Opcode	Description	Encoding
Cmp	Completions for Writebacks, Reads and Invalidates.	000b
Cmp-S	Indication from the DCOH to the Host for Shared state.	001b
Cmp-E	Indication from the DCOH to the Host for Exclusive ownership.	010b
Cmp-M	Indication from the DCOH to the Host for Modified state. This is optionally supported by host implementations and devices must support disabling of this response.	011b
BI-ConflictAck[1]	Completion of the Back-Invalidate conflict handshake.	100b
CmpTEE[1]	Completion for Writes (MemWr*) with TEE intent. Does not apply to any M2S Req.	101b
CmpTEE-S	Indication from the DCOH to the Host for Shared state with TEE intent.	110b
CmpTEE-E	Indication from the DCOH to the Host for Exclusive ownership with TEE intent.	111b

1. Only support in 256B flit mode.

Table 3-51 defines the DevLoad value used in NDR and DRS messages. The encodings were assigned to allow CXL 1.1 backward compatibility such that the 00b value would cause the least impact in the host.

Table 3-51. DevLoad Definition

DevLoad Value	Queuing Delay inside Device	Device Internal Resource Utilization	Encoding
Light Load	Minimal	Readily handles more requests	00b
Optimal Load	Modest to Moderate	Optimally utilized	01b
Moderate Overload	Significant	Limiting request throughput and/or degrading efficiency	10b
Severe Overload	High	Heavily overloaded and/or degrading efficiency	11b

Definition of other fields are the same as for M2S message classes.

3.3.10 S2M Data Response (DRS)

The DRS message class contains memory read data from the Subordinate to the Master.

The fields of the DRS message class are defined in Table 3-52.

Table 3-52. S2M DRS Fields

Field	Width (Bits)			Description
	68B Flit	256B Flit	PBR Flit	
Valid	1			The valid signal indicates that this is a valid request.
Opcode	3			Memory Operation: This specifies which, if any, operation needs to be performed on the data and associated information. Details in Table 3-53.
MetaField	2			Metadata Field: For devices that support memory with Metadata, this field can be encoded as Meta0-State. For devices that do not, this field must be encoded as No-Op. No-Op encoding may also be used by devices if the Metadata is unreliable or corrupted in the device.
MetaValue	2			Metadata Value: If MetaField is No-Op, this field is don't care; otherwise, it must encode the Metadata field as read from Memory.
Tag	16			Tag: This is a reflection of the Tag field sent with the associated M2S Req or M2S RwD.
Poison	1			The Poison bit indicates that the data contains an error. The handling of poisoned data is Host specific. See Chapter 12.0 for more details.
LD-ID[3:0]	4		0	Logical Device Identifier: This identifies a logical device within a multiple-logical device. Not applicable in PBR mode where DPID infers this field.
DevLoad	2			Device Load: Indicates device load as defined in Table 3-51. Values are used to enforce QoS as described in Section 3.3.4.
DPID	0		12	Destination PID
TRP	0	1		Trailer Present: Indicates that a trailer is included after the 64B payload. The Trailer size and legal encodings for DRS are defined in Table 3-54.
RSVD	9	8		
Total	**40**	**40**	**48**	

Table 3-53. S2M DRS Opcodes

Opcode	Description	Encoding
MemData	Memory read data. Sent in response to Reads.	000b
MemData-NXM	Memory Read Data to Non-existent Memory region. This response is only used to indicate that the device or the switch was unable to positively decode the address of the MemRd as either HDM-H or HDM-D*. Must encode the payload with all 1s and set poison if poison is enabled. This special opcode is needed because the host will have expectation of a DRS only for HDM-H or a DRS+NDR for HDM-D*, and this opcode allows devices/switches to send a single response to the host, allowing a deallocation of host tracking structures in an otherwise ambiguous case. See Section 3.3.11 for additional details.	001b
MemDataTEE[1]	Same as MemData but in response to MemRd* with TEE attribute.	010b
Reserved	Reserved	<Others>

1. Only support in 256B Flit mode.

3.3.10.1 Trailer Present for DRS (256B Flit)

In 256B Flit mode, a Trailer Present (TRP) bit is included with the message header that indicates whether a trailer slot is included with the message. The trailer can be up to 32 bits for DRS.

The TRP bit can be inferred by other field decode as defined in Table 3-54 for DRS. It is included to enable simple decode in the Link Layer.

The Extended Metadata (EMD) trailer is the only trailer supported. The Extended Metadata (EMD) trailer can be up to 32 bits. Section 8.2.4.31 describes the registers that aid in discovery of device's EMD capability and EMD related configuration of the device. The mechanism for discovering the host's EMD capabilities and EMD related configuration of the host is host-specific. The host and the device must be configured in a consistent manner.

Table 3-54. DRS Trailers

Opcode/ Message	MetaField	TRP	Trailer Size Required	Description
MemData/ MemDataTEE	EMS	1	32 bits	Trailer bits[31:0] defined as EMD.
	No-OP/MS0	0	No Trailer	
<Others>	N/A	0	No Trailer	

3.3.11 Responses for Requests Targeting NXM

Device responses to CXL.mem requests differ between HDM-H regions and HDM-D/ HDM-DB regions, which creates an ambiguity when device receives a CXL.mem request it cannot map to a specific memory region. In this situation, devices shall respond according to Table 3-55. CXL.mem Responses for Requests to Non-existent Memory requesting device must accept and properly handle these responses regardless of its memory region decode results.

The ambiguity mentioned above is for reads and for some MemInv* cases. For reads, the response is DRS only for HDM-H or a DRS+NDR for HDM-D*. For MemInv*, HDM-H returns Cmp opcode and HDM-D/HDM-DB may expect only Cmp-E or Cmp-S as show in Table C-3, "HDM-DB Memory Requests with TE state".

The capability to support MemData-NXM is exposed in the "CXL HDM Decoder Capability Register" bit 20 (see Section 8.2.4.20.1).

Table 3-55. CXL.mem Responses for Requests to Non-existent Memory

CXL.mem Request[1]	Device Response when NXM
MemRd, MemRdData, MemRdFill, MemRdTEE, MemRdDataTEE, MemRdFillTEE	MemData-NXM See Table 8-27, "CXL.mem Read Response - Error Cases" for additional details.
MemInv, MemInvNT, MemClnEvct, MemWr, MemWrPtl, MemWrTEE, MemWrPtlTEE	Cmp

1. TEE requests have a non-TEE response to allow requester to enforce appropriate security policy.

3.3.12 Forward Progress and Ordering Rules

- Req may be blocked by BISnp to the Host, but RwD cannot be blocked by BISnp to the Host.
  - This rule impacts RwD MemWr* to Shared FAM HDM-DB uniquely requiring SnpType=No-Op to avoid causing BISnp to other requesters that are sharing the memory which could deadlock. The resulting is a requirement that the requester must first get ownership of the cacheline using M2S Req message referred to as a 2-phase write as described in Section 2.4.4.
- A CXL.mem Request in the M2S Req channel must not pass a MemRdFwd or a MemWrFwd, if the Request and MemRdFwd or MemWrFwd are to the same cacheline address.
  - Reason: As described in Table 3-35, MemRdFwd and MemWrFwd opcodes, sent on the M2S Req channel are, in fact, responses to CXL.cache D2H requests. The reason the response for certain CXL.cache D2H requests are on CXL.mem M2S Req channel is to ensure subsequent requests from the Host to the same address remain ordered behind it. This allows the host and device to avoid race conditions. Examples of transaction flows using MemRdFwd are shown in Figure 3-35 and Figure 3-40. Apart from the above, there is no ordering requirement for the Req, RwD, NDR, and DRS message classes or for different addresses within the Req message class.
- NDR and DRS message classes, each, need to be pre-allocated at the request source. This guarantees that the responses can sink and ensures forward progress.
- On CXL.mem, write data is only guaranteed to be visible to a later access after the write is complete.
- CXL.mem requests need to make forward progress at the device without any dependency on any device initiated request except for BISnp messages. This includes any request from the device on CXL.io or CXL.cache.
- S2M and M2S Data transfer of a cacheline must occur with no interleaved transfers.

IMPLEMENTATION NOTE

There are two cases of bypassing with device-attached memory where messages in the M2S RwD channel may pass messages for the same cacheline address in M2S Req channel.

1. Host generated weakly ordered writes (as showing in Figure 3-32) may bypass MemRdFwd and MemWrFwd. The result is the weakly ordered write may bypass older reads or writes from the Device.
2. For Device initiated RdCurr to the Host, the Host will send a MemRdFwd to the device after resolving coherency (as shown in Figure 3-35). After sending the MemRdFwd the Host may have an exclusive copy of the line (because RdCurr does not downgrade the coherency state at the target) allowing the Host to subsequently modify this line and send a MemWr to this address. This MemWr will not be ordered with respect to the previously sent MemRdFwd.

Both examples are legal because weakly ordered stores (in Case #1) and RdCurr (in Case #2) do not guarantee strong consistency.

3.3.12.1 Buried Cache State Rules for HDM-D/HDM-DB

Buried Cache state for CXL.mem protocol refers to the state of the cacheline registered by the Master's Home Agent logic (HA) for a cacheline address when a new Req or RwD message is being sent. This cache state could be a cache that is controlled by the host, but does not cover the cache in the device that is the owner of the HDM-D/HDM-DB memory. These rules are applicable to only HDM-D/HDM-DB memory where the device is managing coherence.

For implementations that allow multiple outstanding requests to the same address, the possible future cache state must be included as part of the buried cache state. To avoid this complexity, it is recommended to limit to one Req/RwD per cacheline address.

Buried Cache state rules for Master-issued CXL.mem Req/RwD messages:

- Must not issue a MemRd/MemInv/MemInvNT (MetaValue=I) if the cacheline is buried in Modified, Exclusive, or Shared state.
- Shall not issue a MemRd/MemInv/MemInvNT (MetaValue=S) or MemRdData if the cacheline is buried in Modified or Exclusive state, but is allowed to issue when the host has Shared or Invalid state.
- May issue a MemRd/MemInv/MemInvNT (MetaValue = A) from any state.
- May issue a MemRd/MemInv/MemInvNT (MetaField = No-Op) from any state. Note that the final host cache state may result in a downgraded state such as Invalid when initial buried state exists and conflicting BISnp results in the buried state being downgraded.
- May issue MemClnEvct from Shared or Exclusive state.
- May issue MemWr with SnpType=SnpInv only from I-state. Use of this encoding is not allowed for HDM-DB memory regions in which coherence extends to multiple hosts (e.g., Coherent Shared FAM as described in Section 2.4.4).
- MemWr with SnpType=No-Op may be issued only from Modified state.

Note: The Master may silently degrade clean cache state (E to S, E to I, S to I) and as such the Subordinate may have more conservative view of the Master's cache state. This section is discussing cache state from the Master's view.

Table 3-56 summarizes the Req message and RwD message allowance for Buried Cache state. MemRdFwd/MemWrFwd/BIConflict are excluded from this table because they are response messages.

Table 3-56. Allowed Opcodes for HDM-D/HDM-DB Req and RwD Messages per Buried Cache State

CXL.mem Req/RwD				Buried Cache State			
Opcodes	MetaField	MetaValue	SnpType	Modified	Exclusive	Shared	Invalid
MemRdData	All Legal Combinations		All Legal Combinations			X	X
MemClnEvct					X	X	
MemRd/ MemInv/ MemInvNT	MS0/EMD	A		X[1]	X	X	X
		S				X	X
		I					X
	No-Op	N/A		X[1]	X	X	X
	EMD	Explicit No-Op					
MemWr	All Legal Combinations		No-Op	X			
			SnpInv				X
MemRdFill/ MemRdFillTEE/ MemRdDataTEE/ MemRdTEE/ MemWrTEE	N/A (Commands not supported for HDM-D/HDM-DB)						

1. Requesters that have active reads with buried-M state must expect data return to be stale. It is up to the requester to ensure that possible stale data case is handled in all cases including conflicts with BISnp.

3.4 Transaction Ordering Summary

This section presents CXL ordering rules in a series of tables and descriptions. Table 3-57 captures the upstream ordering cases. Table 3-58 captures the downstream ordering cases.

For CXL.mem and CXL.cache, the term upstream describes traffic on all S2M and D2H message classes, and the term downstream describes traffic on all M2S and H2D message classes, regardless of the physical direction of travel.

Where upstream and downstream traffic coexist in the same physical direction within PBR switches and on Inter Switch Links (ISLs) or on links from a device that issues direct P2P CXL.mem, the upstream and downstream Ordering Tables each apply to their corresponding subset of the traffic and each subset shall be independent and not block one another.

Table 3-59 lists the Device in-out dependence. Table 3-60 lists the Host in-out dependence. Additional detail is provided in Section 3.2.2.1 for CXL.cache and in Section 3.3.12 for CXL.mem.

In Table 3-57 and Table 3-58, the columns represent a first-issued message and the rows represent a subsequently issued message. The table entry indicates the ordering relationship between the two messages. The table entries are defined as follows:

- Yes: The second message (row) must be allowed to pass the first message (column) to avoid deadlock. (When blocking occurs, the second message is required to pass the first message.)
- Y/N: There are no ordering requirements. The second message may optionally pass the first message or may be blocked by it.

- No: The second message must not be allowed to pass the first message. This is required to support the protocol ordering model.

Note: Passing, where permitted, must not be allowed to cause the starvation of any message class.

Table 3-57. Upstream Ordering Summary

Row Pass Column?	CXL.io TLPs (Col 2-5)	S2M NDR/DRS D2H Rsp/Data (Col 6)	D2H Req (Col 7)	S2M BISnp (Col 13)
CXL.io TLPs (Row A-D)	PCIe Base	Yes(1)	Yes(1)	Yes(1)
S2M NDR/DRS D2H Rsp/Data (Row E)	Yes(1)	a. No(3) b. Y/N	Yes(2)	Yes(2)(4)
D2H Req (Row F)	Yes(1)	Y/N	Y/N	Y/N
S2M BISnp (Row M)	Yes(1)(4)	Y/N	Yes(4)	Y/N

Explanation of row and column headers:

M7 requires BISnp to pass D2H Req in accordance with dependence relationship: D2H Req depends on M2S Req depends on S2M BISnp.

E6a requires that within the NDR channel, BIConflictAck must not pass prior Cmp* messages with the same Cacheline Address (implied by the tag field).

E6b other cases not covered by rule E6a are Y/N.

Color-coded rationale for cells in Table 3-57	
Yes(1)	CXL architecture requirement for ARB/MUX.
Yes(2)	CXL.cachemem: Required for deadlock avoidance.
No(3)	Type 2/3 devices where BIConflictAck must not pass prior Cmp* to the same address.
Yes(4)	Required for deadlock avoidance with the introduction of the BISnp channel. For CXL.io Unordered I/O, this is necessary because Unordered I/O can trigger BISnp.

Table 3-58. Downstream Ordering Summary (Sheet 1 of 2)

Row Pass Column?	CXL.io TLPs (Col 2-5)	M2S Req (Col 8)	M2S RwD (Col 9)	H2D Req (Col 10)	H2D Rsp (Col 11)	H2D Data (Col 12)	M2S BIRsp (Col 14)
CXL.io TLPs (Row A-D)	PCIe Base	Yes(1)	Yes(1)	Yes(1)	Yes(1)	Yes(1)	Yes(1)
M2S Req (Row G)	Yes(1)	a. No(5) b. Y/N	Y/N	Y/N(3)	Y/N	Y/N	Y/N
M2S RwD (Row H)	Yes(1)(6)	a. Yes(6) b. Y/N	Y/N	Yes(3)	Y/N	Y/N	Y/N
H2D Req (Row I)	Yes(1)	Yes(2)(6)	a. Yes(2) b. Y/N	Y/N	a. No(4) b. Y/N	Y/N(3)	Y/N

Table 3-58. Downstream Ordering Summary (Sheet 2 of 2)

Row Pass Column?	CXL.io TLPs (Col 2-5)	M2S Req (Col 8)	M2S RwD (Col 9)	H2D Req (Col 10)	H2D Rsp (Col 11)	H2D Data (Col 12)	M2S BIRsp (Col 14)
H2D Rsp (Row J)	Yes(1)	Yes(2)	Yes(2)	Yes(2)	Y/N	Y/N	Y/N
H2D Data (Row K)	Yes(1)	Yes(2)	Yes(2)	Yes(2)	Y/N	Y/N	Y/N
M2S BIRsp (Row N)	Yes(1)(6)	Yes(2)	Yes(2)	Yes(2)	Y/N	Y/N	Y/N

Explanation of row and column headers:

In Downstream direction pre-allocated channels are kept separate because of unique ordering requirements in each.

Color-coded rationale for cells in Table 3-58	
Yes(1)	CXL architecture requirement for ARB/MUX.
Yes(2)	CXL.cachemem: Required for deadlock avoidance.
Yes(3)	CXL.cachemem: Performance optimization.
Y/N(3)	CXL.cachemem: Non-blocking recommended for performance optimization.
No(4)	Type 1/2 device: Snoop push GO requirement.
No(5)	Type 2 device: MemRd*/MemInv* push Mem*Fwd requirement.
Yes(6)	Required for deadlock avoidance with the introduction of the BISnp channel.

Explanation of table entries:

G8a MemRd*/MemInv* must not pass prior Mem*Fwd messages to the same cacheline address. This rule is applicable only for HDM-D memory regions in devices which result in receiving Mem*Fwd messages (Type 3 devices with no HDM-D don't need to implement this rule). This rule does not apply to Type 2 devices that implement the HDM-DB memory region which use the BI* channels because they do not support Mem*Fwd.

G8b All other cases not covered by rule G8a do not have ordering requirements (Y/N).

H8a applies to components that support the BISnp/BIRsp message classes to ensure that the RwD channel can drain to the device even if the Req channel is blocked.

H8b applies to components that do not support the BISnp/BIRsp message classes.

I9a applies for PBR-capable switches, for ISLs, and for devices that can initiate P2P CXL.mem. (Possible future use case for Host-to-Host CXL.mem will require host to apply this ordering rule.)

I9b applies to all other cases.

I11a Snoops must not pass prior GO* messages to the same cacheline address. GO messages do not carry the address, so implementations where address cannot be inferred from UQID in the GO message will need to strictly apply this rule across all messages.

I11b Other case not covered by I11a are Y/N.

Table 3-59. Device In-Out Ordering Summary

Row (in) Independent of Column (out)?	CXL.io TLPs (Col A-D)	S2M NDR/DRS D2H Rsp/Data (Col E)	D2H Req (Col F)	S2M BISnp (Col M)	M2S Req (Col N)[1]	M2S RwD (Col O)[1]	M2S BIRsp (Col P)[1]
CXL.io TLPs (Row 2-5)	PCIe Base	Y/N(1)	Y/N(1)	Y/N(1)	Y/N(1)	Y/N(1)	Y/N(1)
		Yes(3)	Yes(3)	Yes(3)	Yes(3)	Yes(3)	Yes(3)
M2S Req (Row 8)	Yes(1)	Y/N	Yes(2)	Y/N	Yes(2)	Y/N	Y/N
M2S RwD (Row 9)	Yes(1)(2)	Y/N	Yes(2)	Yes(2)	Yes(2)	Yes(2)	Y/N
H2D Req (Row 10)	Yes(1)	Y/N	Yes(2)	Yes(2)	Yes(2)	Yes(2)	Y/N
H2D Rsp (Row 11)	Yes(1)	Yes(2)	Yes(2)	Yes(2)	Yes(2)	Yes(2)	Yes(2)
H2D Data (Row 12)	Yes(1)	Yes(2)	Yes(2)	Yes(2)	Yes(2)	Yes(2)	Yes(2)
M2S BIRsp (Row 14)	Yes(1)(2)	Yes(2)	Yes(2)	Yes(2)	Yes(2)	Yes(2)	Yes(2)
S2M NDR/DRS (Row 15)[1]	Yes(1)	Yes(2)	Yes(2)	Yes(2)	Yes(2)	Yes(2)	Y/N
S2M BISnp (Row 16)[1]	Yes(1)	Y/N	Yes(2)	Yes(2)	Yes(2)	Y/N	Y/N

1. These rows and columns are supported only by devices that have Direct P2P CXL.mem enabled.

In the device ordering, the row represents incoming message class and the column represents the outgoing message class. The cases in this table show when incoming must be independent of outgoing (Yes) and when it is allowed to block incoming based on outgoing (Y/N).

Color-coded rationale for cells in Table 3-59	
Yes(1)	CXL.cachemem is independent of outgoing CXL.io.
Y/N(1)	CXL.io traffic, except UIO Completions, may be blocked by CXL.cachemem.
Yes(2)	CXL.cachemem: Required for deadlock avoidance.
Yes(3)	CXL UIO completions are independent of CXL.cachemem.

Table 3-60. Host In-Out Ordering Summary (Sheet 1 of 2)

Row (in) Independent of Column (out)?	CXL.io TLPs (Col A-D)	M2S Req (Col G)	M2S RwD (Col H)	H2D Req (Col I)	H2D Rsp (Col J)	H2D Data (Col K)	M2S BIRsp (Col N)
CXL.io TLPs (Row 2-5)	PCIe Base	Y/N(1)	Y/N(1)	Y/N(1)	Y/N(1)	Y/N(1)	Y/N(1)
		Yes(3)	Yes(3)	Yes(3)	Yes(3)	Yes(3)	Yes(3)

Table 3-60. Host In-Out Ordering Summary (Sheet 2 of 2)

Row (in) Independent of Column (out)?	CXL.io TLPs (Col A-D)	M2S Req (Col G)	M2S RwD (Col H)	H2D Req (Col I)	H2D Rsp (Col J)	H2D Data (Col K)	M2S BIRsp (Col N)
S2M NDR/DRS D2H Rsp/Data (Row 6)	Yes(1)(2)	Yes(2)	Yes(2)	Yes(2)	Y/N	Y/N	Y/N
D2H Req (Row 7)	Yes(1)	Y/N	Y/N	Y/N	Y/N	Y/N	Y/N
S2M BISnp (Row 13)	Yes(1)(2)	Yes(2)	Y/N	Y/N	Y/N	Y/N	Y/N

In the host ordering, the row represents incoming message class and the column represents the outgoing message class. The cases in this table show when incoming must be independent of outgoing (Yes) and when it is allowed to block incoming based on outgoing (Y/N).

Color-coded rationale for cells in Table 3-60	
Yes(1)	Incoming CXL.cachemem must not be blocked by outgoing CXL.io.
Y/N(1)	Incoming CXL.io may be blocked by outgoing CXL.cachemem.
Yes(2)	CXL.cachemem: Required for deadlock avoidance.
Yes(3)	CXL UIO completions are independent of CXL.cachemem.

3.5 Transaction Flows to Device-attached Memory

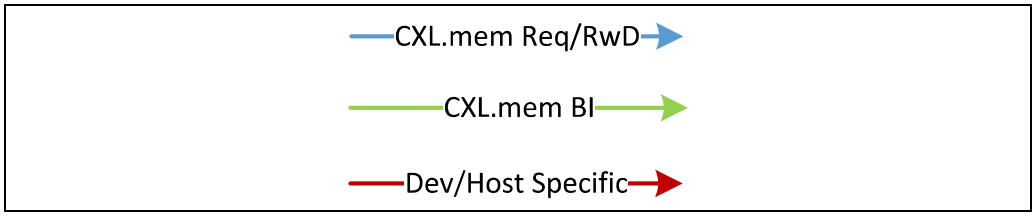
3.5.1 Flows for Back-Invalidate Snoops on CXL.mem

3.5.1.1 Notes and Assumptions

The Back-Invalidate Snoop (BISnp) channel provides a dedicated channel S2M to allow the owner of an HDM region to snoop a host that may have a cached copy of the line. The forward progress rules as defined in Section 3.4 ensure that the device can complete the BISnp while blocking new requests (M2S Req).

The term Snoop Filter (SF) in the following diagrams is a structure in the device that is inclusively tracking any host caching of device memory and is assumed to have a size that may be less than the total possible caching in the host. The Snoop Filter is kept inclusive of host caching by sending "Back-Invalidate Snoops" to the host when it becomes full. This full trigger that forces the BISnp is referred to as "SF Victim". In the diagrams, an "SF Miss" that is caused by an M2S request implies that the device must also allocate a new SF entry if the host is requesting a cached copy of the line. When allocating an SF entry, it may also trigger an SF Victim for a different cacheline address if the SF is full. Figure 3-20 provides the legend for the Back-Invalidate Snoop flow diagrams that appear in the subsections that follow. The "CXL.mem BI" type will cover the BI channel messages and any conflict message/flow (e.g., BIConflict) that flow on the RwD channels. Note that the "Dev/Host Specific" messages are just short-hand flows for the type of flow expected in the host or device.

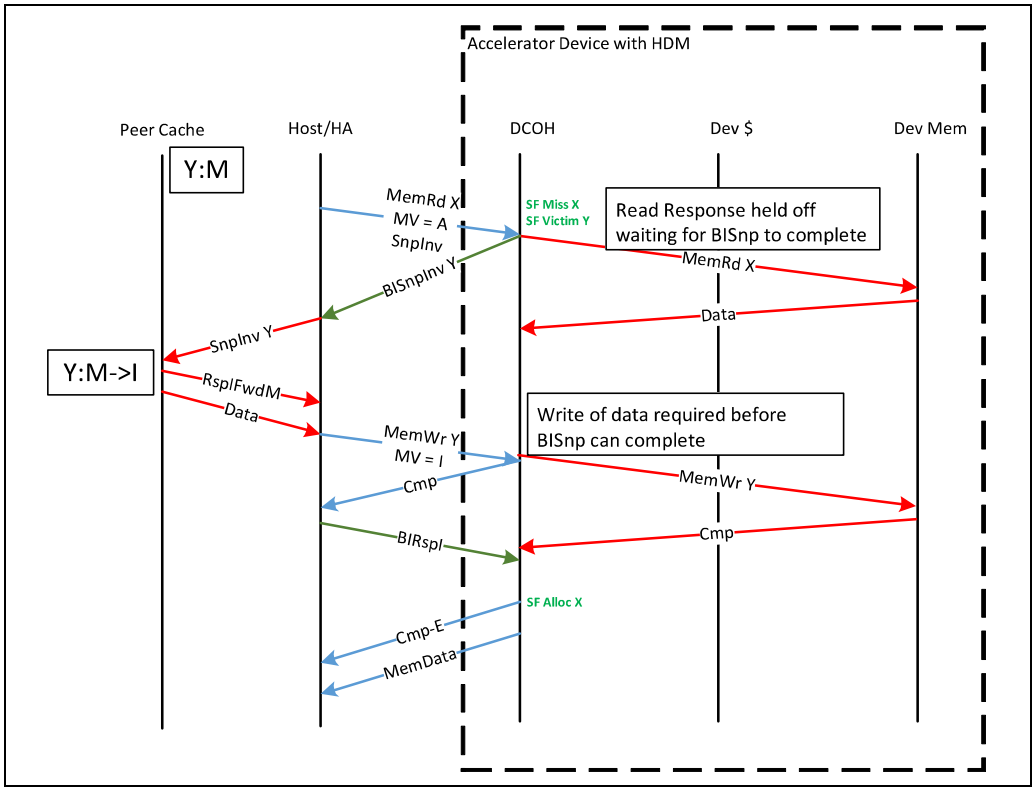
Figure 3-20. Flows for Back-Invalidate Snoops on CXL.mem Legend



3.5.1.2 BISnp Blocking Example

Figure 3-21 starts out with MemRd that is an SF Miss in the device. The SF is full, which prevents SF allocation; thus, the device must create room in the SF by triggering an SF Victim for Address Y before it can complete the read. In this example, the read to device memory Address X is started in parallel with the BISnpInv to Address Y, but the device will be unable to complete the MemRd until it can allocate an SF which requires the BISnp to Y to complete. As part of the BISnpInv, the host finds modified data for Y which must be flushed to the device before the BISnpInv can complete. The device completes the MemWr to Y, which allows the host to complete the BISnpInv to Y with the BIRspI. That completion allows the SF allocation to occur for Address X, which enables the Cmp-E and MemData to be sent.

Figure 3-21. Example BISnp with Blocking of M2S Req

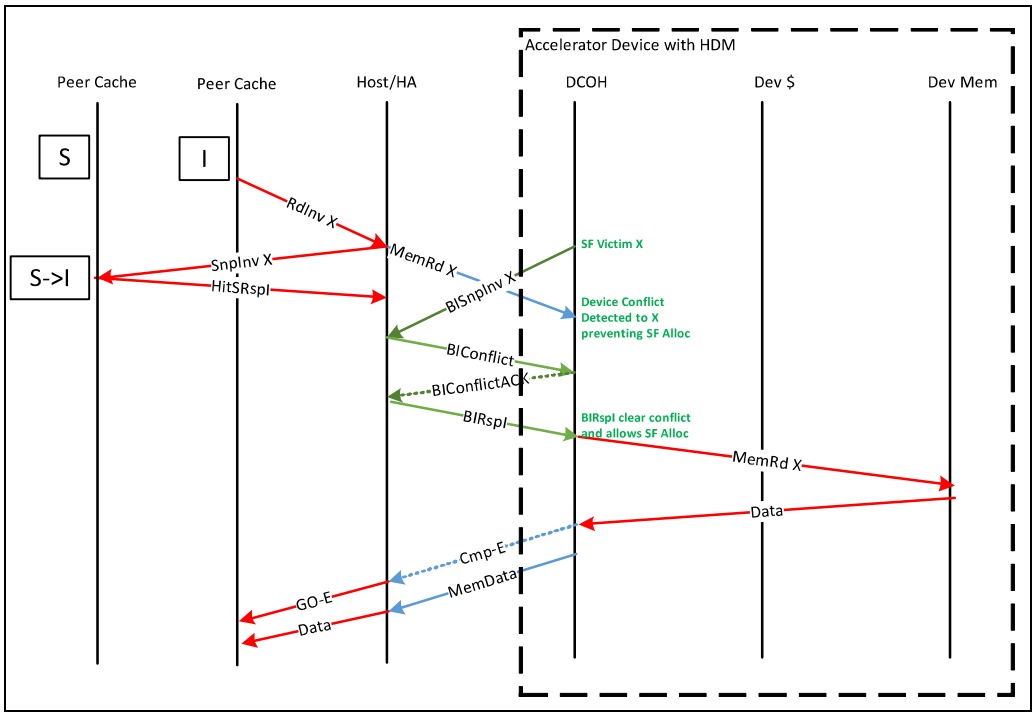


3.5.1.3 Conflict Handling

A conflict is defined as a case where S2M BISnp and M2S Req are active at the same time to the same address. There are two cases to consider: Early Conflict and Late Conflict. The two cases are ambiguous to the host side of the link until observation of a Cmp message relative to BIConflictAck. The conflict handshake starts when by the host detecting a BISnp to the same address as a pending Req. The host sends a BIConflict with the Tag of the M2S Req and device responds to a BIConflict with a BIConflictAck which must push prior Cmp* messages within the NDR channel. This ordering relationship is fundamental to allow the host to correctly resolve the two cases.

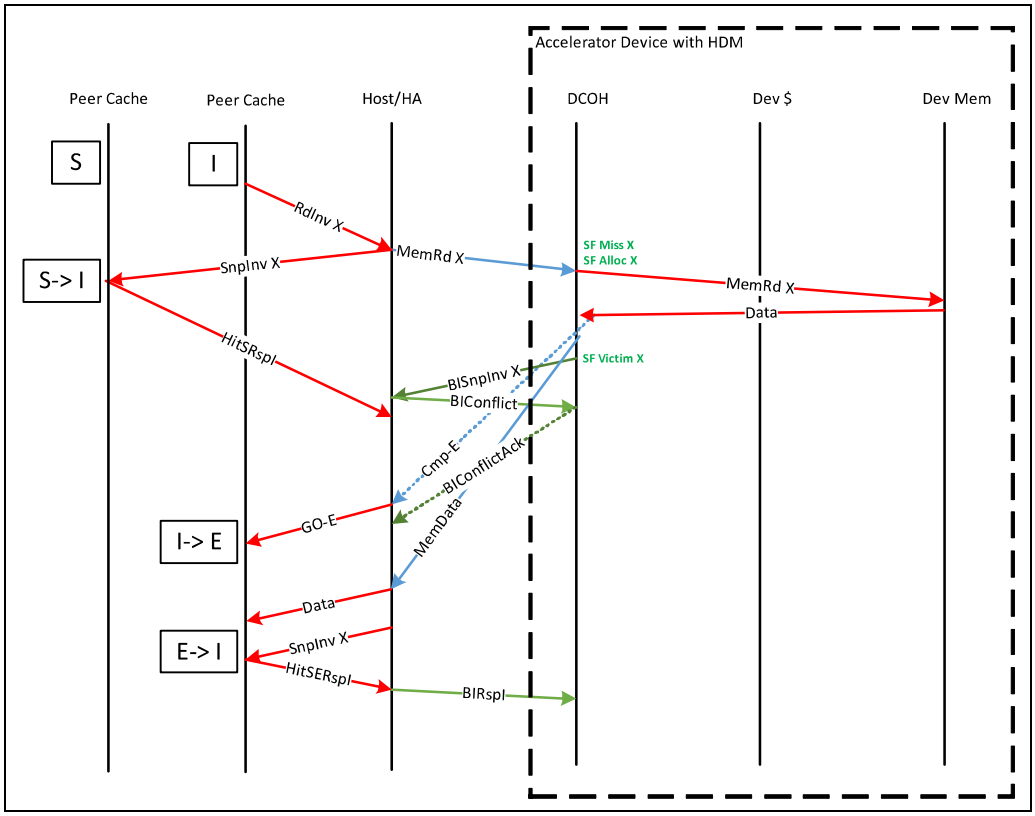
The Early Conflict case in Figure 3-22 is defined as a case where M2S Req is blocked (or in flight) at the device while S2M BISnp is active. The host observing BIConflictAck before Cmp-E determines the M2S MemRd is still pending so that it can reply with RspI.

Figure 3-22. BISnp Early Conflict



Late conflict is captured in Figure 3-23 and is defined as the case where M2S Req was processed and completions are in flight when BISnp is started. In the example below, the Cmp-E message is observed at the host before BIConflictAck, so the host must process the BISnpInv with E-state ownership, which requires it to degrade E to I before completing the BISnpInv with BIRspI. Note that MemData has no ordering requirement and can be observed either before or after the BIConflictAck, although this example shows it after which delays the host's ability to immediately process the internal SnpInv X.

Figure 3-23. BISnp Late Conflict



3.5.1.4 Block Back-Invalidate Snoops

To support increased efficient snooping the BISnp channel defines messages that can Snoop multiple cachelines in the host in a single message. These messages support either 2 or 4 cachelines where the base address must be naturally aligned with the length (128B or 256B). The host is allowed to respond with either a single block response or individual snoop responses per cacheline.

Figure 3-24 is an example of a Block response case. In this example the host receives the BISnpInvBlk for Y, which is a 256B block. Internally the host logic is showing resolving coherence by snooping Y0 and Y2 and the host HA tracker knows the other portions of the block Y1 and Y3 are already in the invalid state, so it does not need to snoop for that portion of the 256B block. Once snoop responses for Y0 and Y2 are completed, the Host HA can send the BIRspIBlk indicating that the entire block is in I-state within the host, thereby allowing the device to have Exclusive access to the block. This results in the SF in I-state for the block and the device cache in E-state.

Figure 3-24. Block BISnp with Block Response

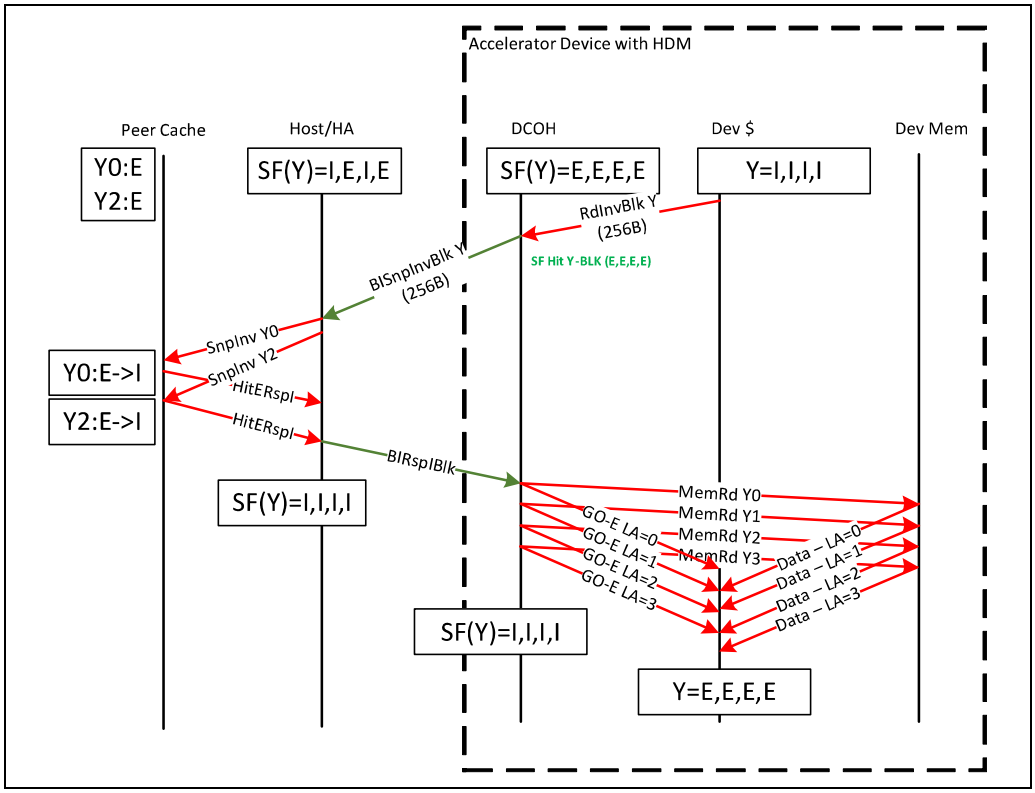
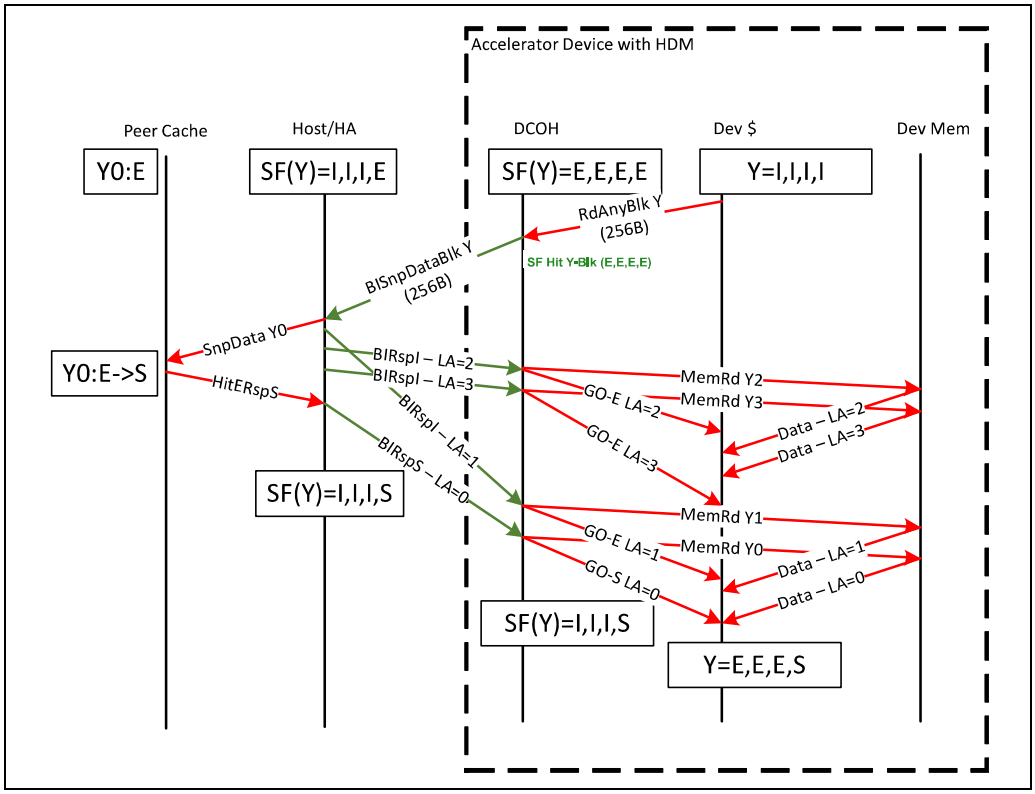


Figure 3-25 is an example where the host sends individual cacheline responses on CXL.mem for each cacheline of the block. The host encodes the 2-bit Lower Address (LowAddr) of the cacheline (Address[7:6]) with each cacheline response to allow the device to determine for which portion of the block the response is intended. The device may see the response messages in any order, which is why LA must be explicitly sent. In a Block, BISnp Address[7:6] is used to indicate the offset and length of the block as defined in Table 3-48 and is naturally aligned to the length.

Figure 3-25. Block BISnp with Cacheline Response



3.5.2 Flows for Type 1 Devices and Type 2 Devices

3.5.2.1 Notes and Assumptions

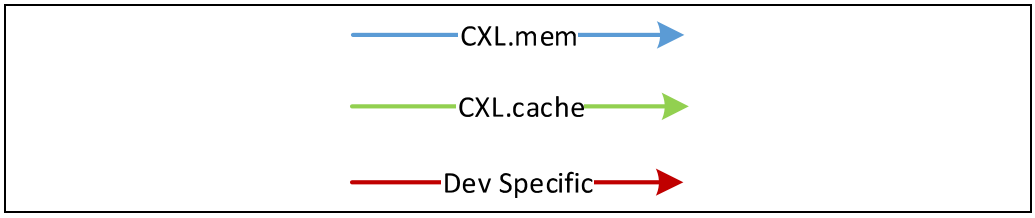
The transaction flow diagrams below are intended to be illustrative of the flows between the Host and device for access to device-attached Memory using the Bias-Based Coherency mechanism described in Section 2.2.2. However, these flows are not comprehensive of every Host and device interaction. The diagrams below make the following assumptions:

- The device contains a coherency engine which is called DCOH in the diagrams below.
- The DCOH contains a Snoop Filter which tracks any caches (called Dev cache) implemented on the device. This is not strictly required, and the device is free to choose an implementation specific mechanism as long as the coherency rules are obeyed.
- The DCOH contains host coherence tracking logic for the device-attached memory. This tracking logic is referred to as a Bias Table in the context of the HDM-D memory region. For HDM-DB, it is referred to as a Directory or a Host Snoop Filter. The implementation of this is device specific.
- The device-specific aspects of the flow, illustrated using red flow arrows, need not conform exactly to the diagrams below. These can be implemented in a device-specific manner.
- Device-attached Memory exposed in a Type 2 device can be either HDM-D or HDM-DB. HDM-D will resolve coherence using a request that is issued on CXL.cache and

the Host will send a Mem*Fwd as a response on the CXL.mem Req channel. The HDM-DB region uses the separate CXL.mem BISnp channel to manage coherence with detailed flows covered in Section 3.5.1. This section will indicate where the flows differ.

Figure 3-26 provides the legend for the diagrams that follow.

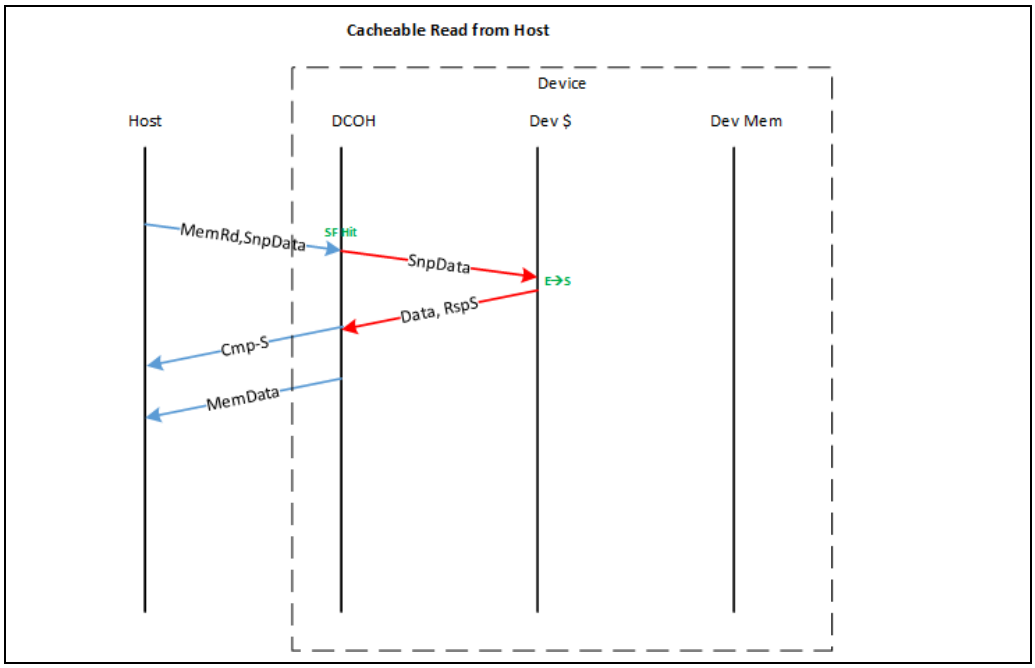
Figure 3-26. Flows for Type 1 Devices and Type 2 Devices Legend



3.5.2.2 Requests from Host

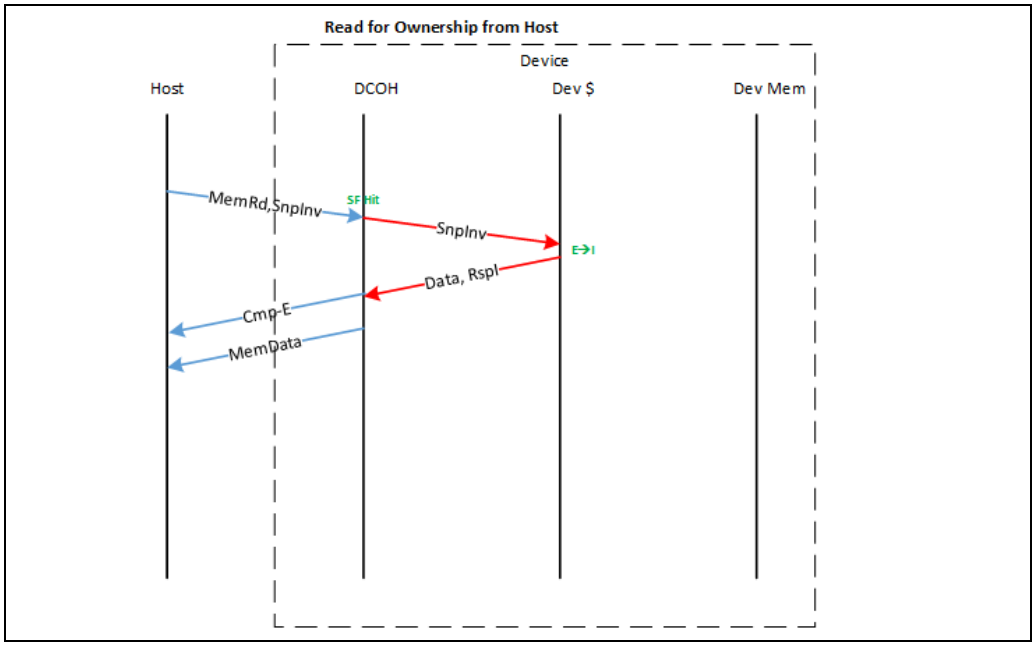
Please note that the flows shown in this section (Requests from Host) do not change on the CXL interface regardless of the bias state of the target region. This effectively means that the device needs to give the Host a consistent response, as expected by the Host and shown in Figure 3-27.

Figure 3-27. Example Cacheable Read from Host



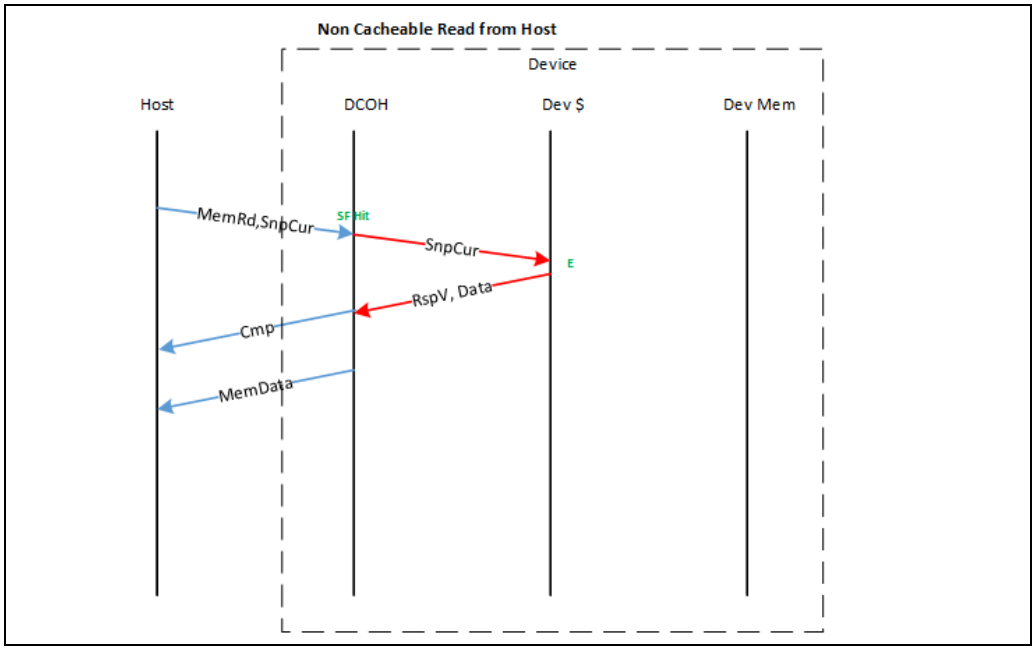
In the above example, the Host requested a cacheable non-exclusive copy of the line. The non-exclusive aspect of the request is communicated using the "SnpData" semantic. In this example, the request got a snoop filter hit in the DCOH, which caused the device cache to be snooped. The device cache downgraded the state from Exclusive to Shared and returned the Shared data copy to the Host. The Host is told of the state of the line using the Cmp-S semantic.

Figure 3-28. Example Read for Ownership from Host



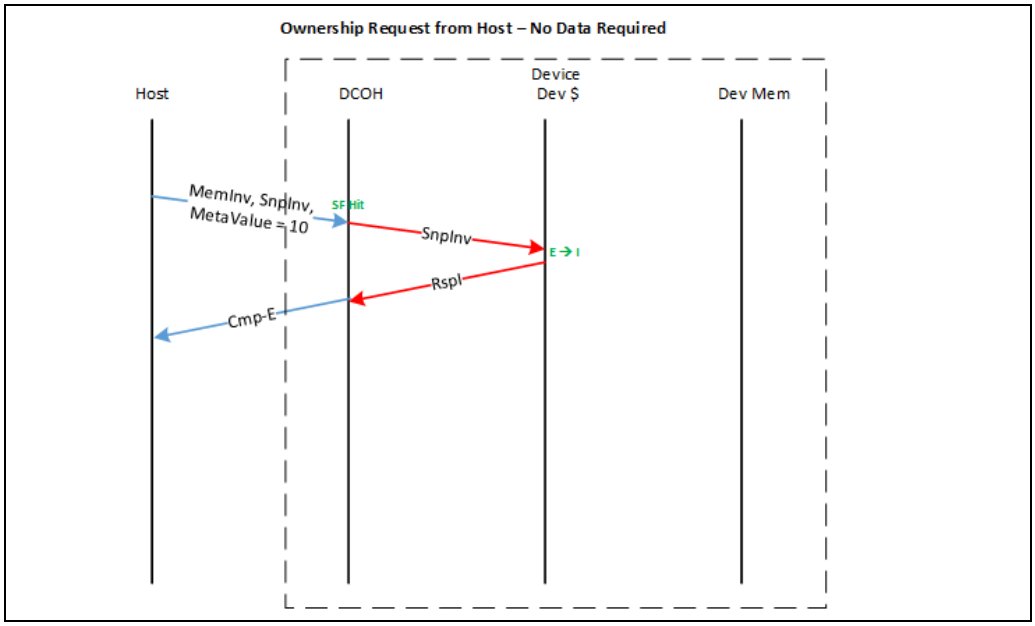
In the above example, the Host requested a cacheable exclusive copy of the line. The exclusive aspect of the request is communicated using the "SnpInv" semantic, which asks the device to invalidate its caches. In this example, the request got a snoop filter hit in the DCOH, which caused the device cache to be snooped. The device cache downgraded the state from Exclusive to Invalid and returned the Exclusive data copy to the Host. The Cmp-E semantic is used to communicate the line state to the Host.

Figure 3-29. Example Non Cacheable Read from Host



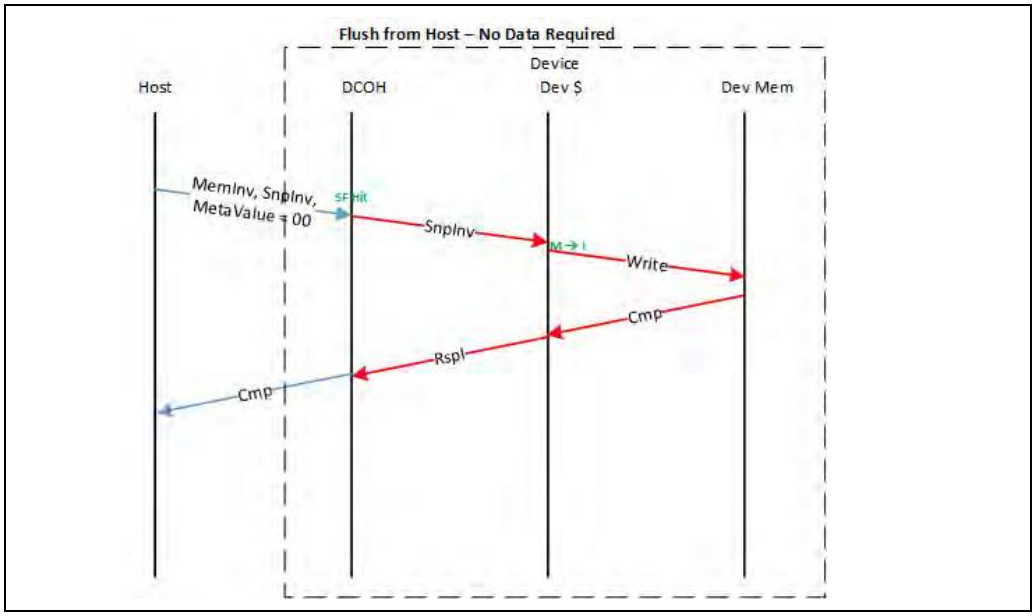
In the above example, the Host requested a non-cacheable copy of the line. The non-cacheable aspect of the request is communicated using the "SnpCur" semantic. In this example, the request got a snoop filter hit in the DCOH, which caused the device cache to be snooped. The device cache did not need to change its caching state; however, it gave the current snapshot of the data. The Host is told that it is not allowed to cache the line using the Cmp semantic.

Figure 3-30. Example Ownership Request from Host - No Data Required



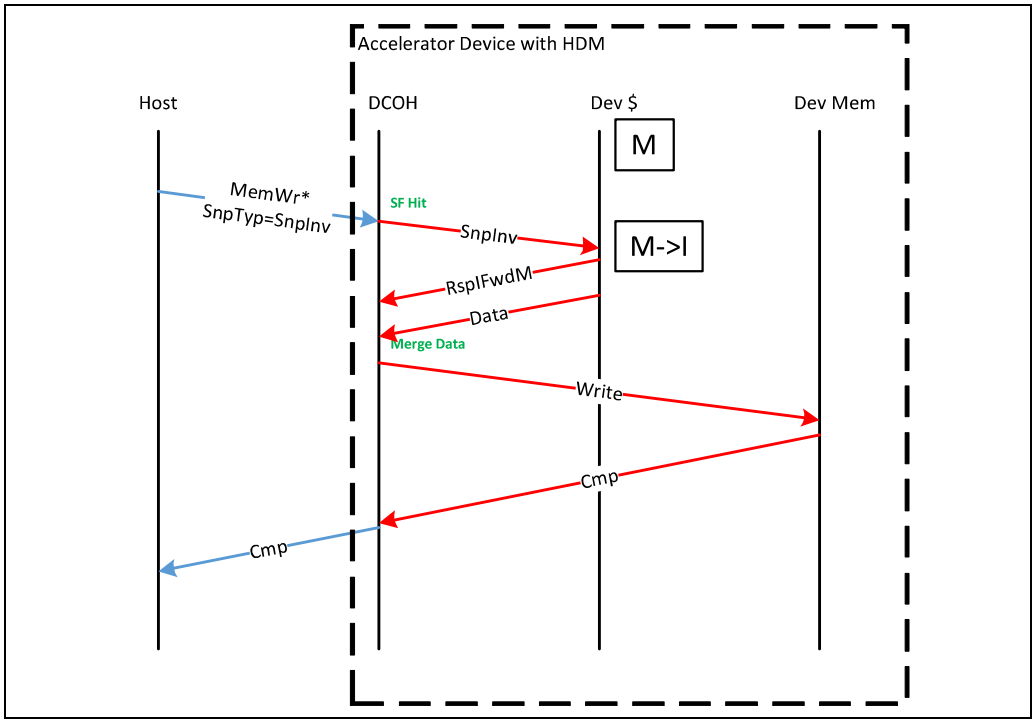
In the above example, the Host requested exclusive access to a line without requiring the device to send data. It communicates that to the device using an opcode of MemInv with a MetaValue of 10b (Any), which is significant in this case. It also asks the device to invalidate its caches with the SnpInv command. The device invalidates its caches and gives exclusive ownership to the Host as communicated using the Cmp-E semantic.

Figure 3-31. Example Flush from Host



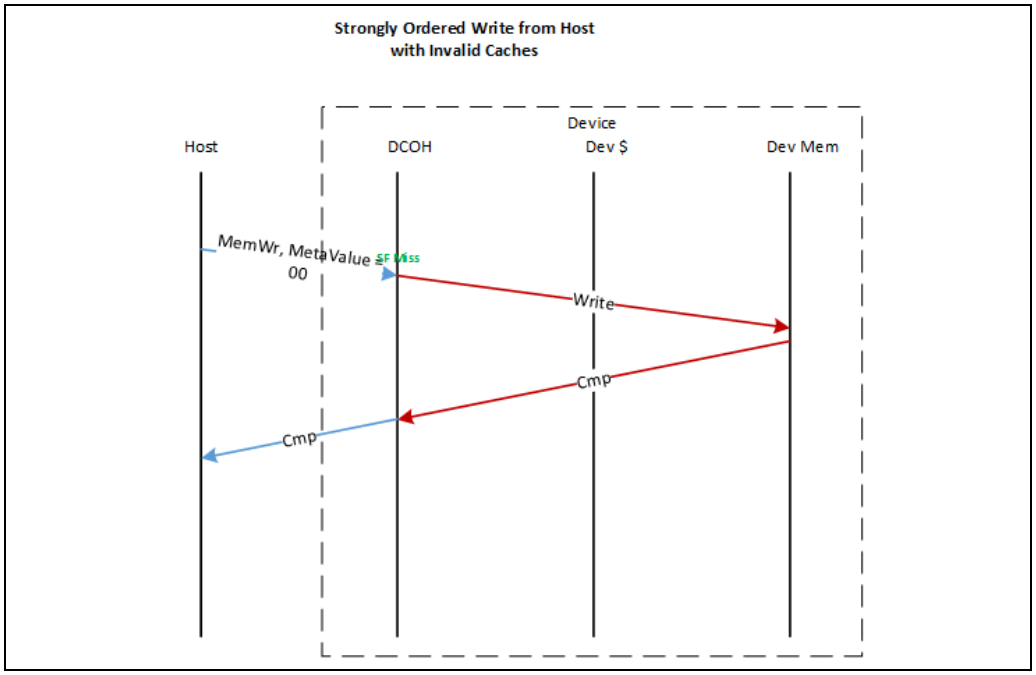
In the above example, the Host wants to flush a line from all caches, including the device's caches, to device memory. To do so, it uses an opcode of MemInv with a MetaValue of 00b (Invalid) and a SnpInv. The device flushes its caches and returns a Cmp indication to the Host.

Figure 3-32. Example Weakly Ordered Write from Host



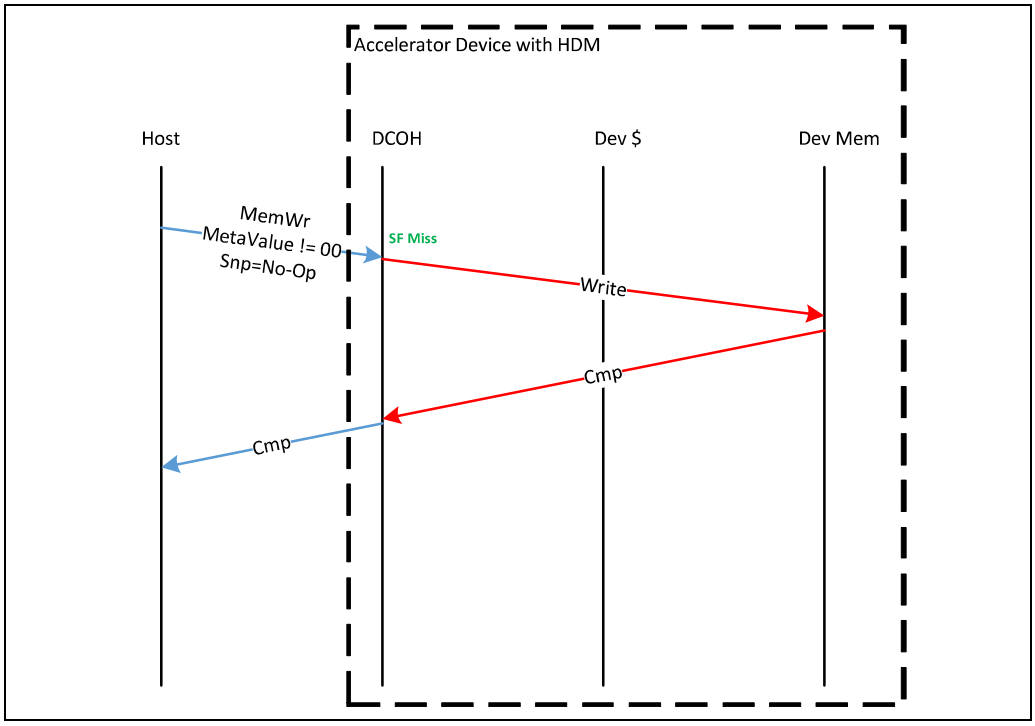
In the above example, the Host issues a weakly ordered write (partial or full line). The weakly ordered semantic is communicated by the embedded SnpInv. In this example, the device had a copy of the line cached. This resulted in a merge within the device before writing it back to memory and sending a Cmp indication to the Host. The term "weakly ordered" in this context refers to an expected-use model in the host CPU in which ordering of the data is not guaranteed until after the Cmp message is received. This is in contrast to a "data visibility is guaranteed with the host" CPU cache in M-state.

Figure 3-33. Example Write from Host with Invalid Host Caches



In the above example, the Host performed a write while guaranteeing to the device that it no longer has a valid cached copy of the line. The fact that the Host didn't need to snoop the device's caches means that the Host previously acquired an exclusive copy of the line. The guarantee on no valid cached copy is indicated by a MetaValue of 00b (Invalid).

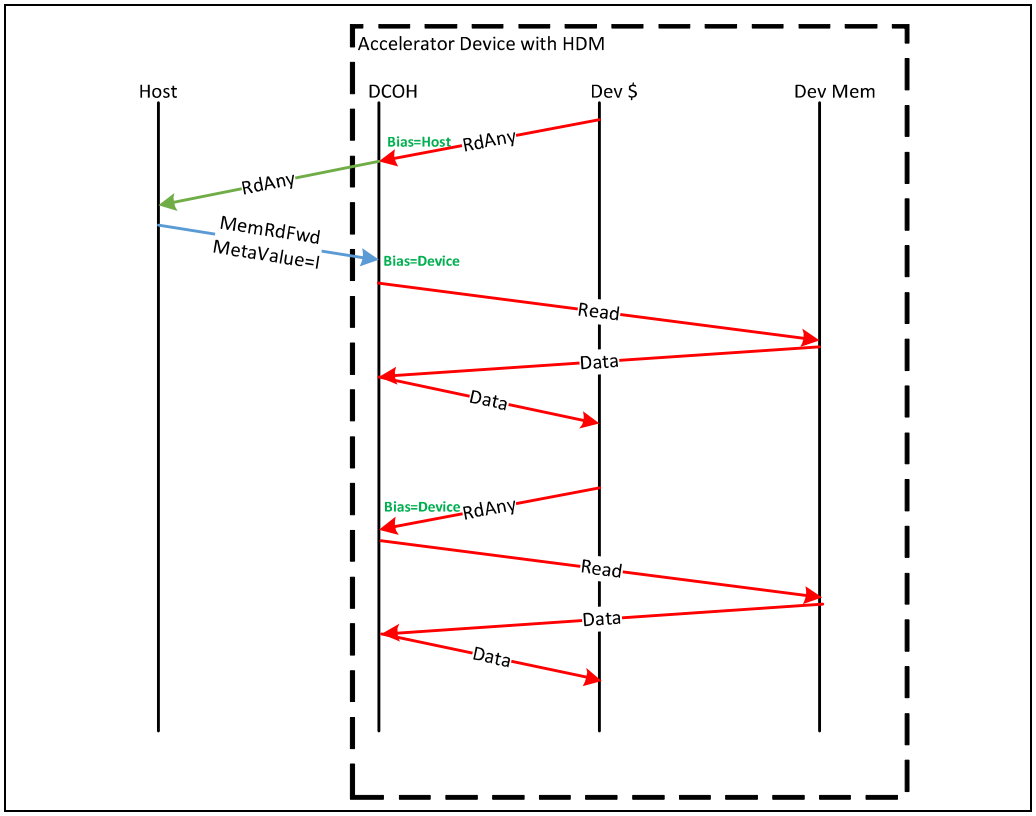
Figure 3-34. Example Write from Host with Valid Host Caches



The above example is the same as the previous one except that the Host chose to retain a valid cacheable copy of the line after the write. This is communicated to the device using a MetaValue of not 00b (Invalid).

3.5.2.3 Requests from Device in Host and Device Bias

Figure 3-35. Example Device Read to Device-attached Memory (HDM-D)



The two flows in Figure 3-35 both start with an internal CXL.cache request (RdAny) that targets the device's HDM-D address region.

In the first flow in Figure 3-35, a device read to device attached memory happened to find the line in Host bias. Because it is in Host bias, the device needs to send the request to the Host to resolve coherency. The Host, after resolving coherency, sends a MemRdFwd on CXL.mem to complete the transaction, at which point the device can internally complete the read.

In the second flow in Figure 3-35, the device read happened to find the line in Device Bias. Because it is in Device Bias, the read can be completed entirely within the device itself and a request doesn't need to be sent to the Host.

The same device request is shown in Figure 3-36, but in this case the target is the HDM-DB address region, meaning that the BISnp channel is used to resolve coherence with the host. In this flow, the difference is that the SF Hit (similar to BIAS=host) indicates that the host could have a cached copy, so BISnpData is sent to the host to resolve coherence. After the host resolves coherence, the host responds with BIRspI indicating that the host is in I-state and that the device can proceed to access its data.

Figure 3-36. Example Device Read to Device-attached Memory (HDM-DB)

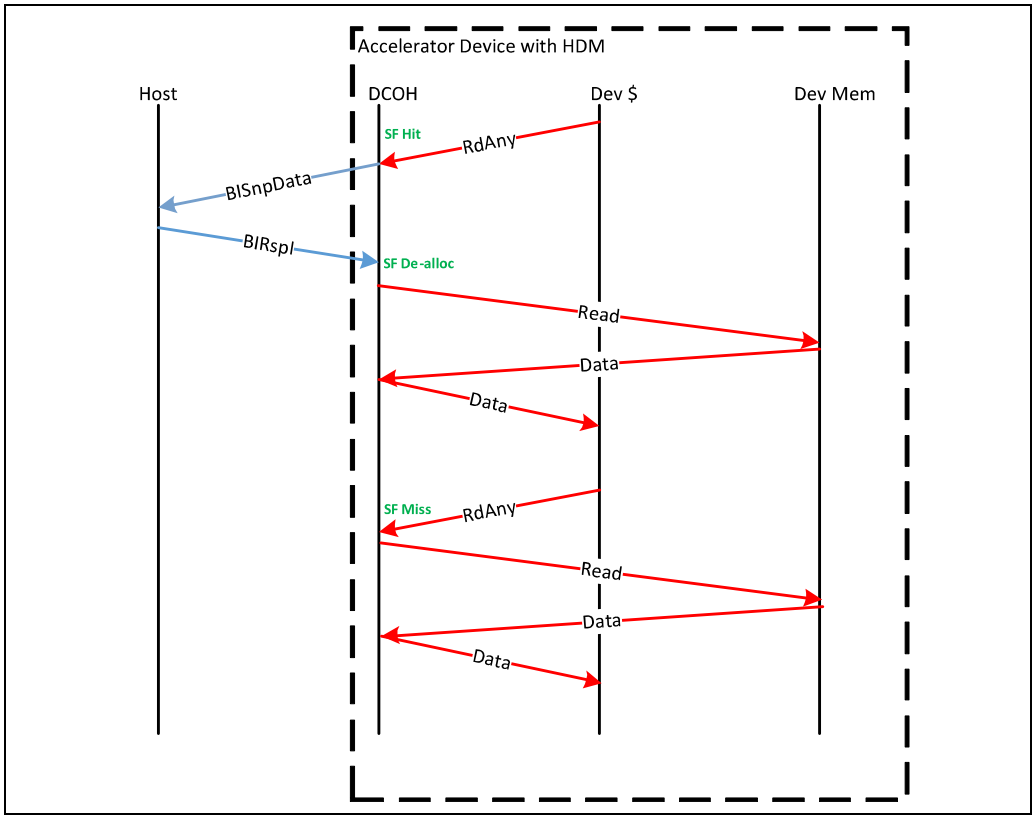
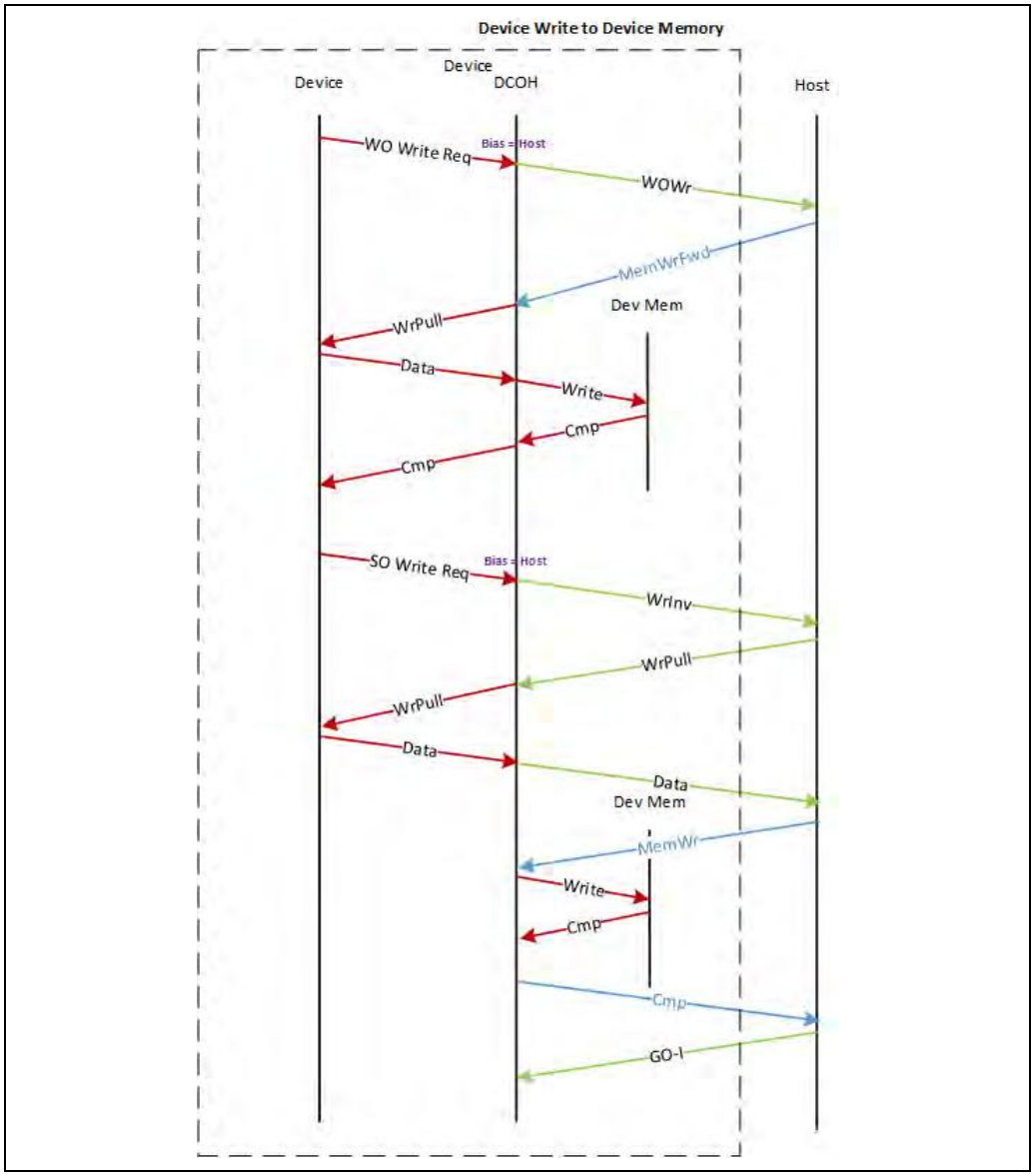


Figure 3-37. Example Device Write to Device-Attached Memory in Host Bias (HDM-D)



There are two flows shown above in Figure 3-37 for the HDM-D region. Both start with the line in Host Bias: a weakly ordered write request and a strongly ordered write request.

In the case of the weakly ordered write request, the request is issued by the device to the Host to resolve coherency. The Host resolves coherency and sends a CXL.mem MemWrFwd opcode, which carries the completion for the WOWrInv* command on CXL.cache. The CQID associated with the CXL.cache WOWrInv* command is reflected in the Tag of the CXL.mem MemWrFwd command. At this point, the device is allowed to complete the write internally. After sending the MemWrFwd, because the Host no longer prevents future accesses to the same line, this is considered a weakly ordered write.

In the second flow, the write is strongly ordered. To preserve the strongly ordered semantic, the Host can prevent future accesses to the same line while this write completes. However, as can be seen, this involves two transfers of the data across the link, which is inefficient. Unless strongly ordered writes are absolutely required, better performance can be achieved with weakly ordered writes.

Figure 3-38. Example Device Write to Device-attached Memory in Host Bias (HDM-DB)

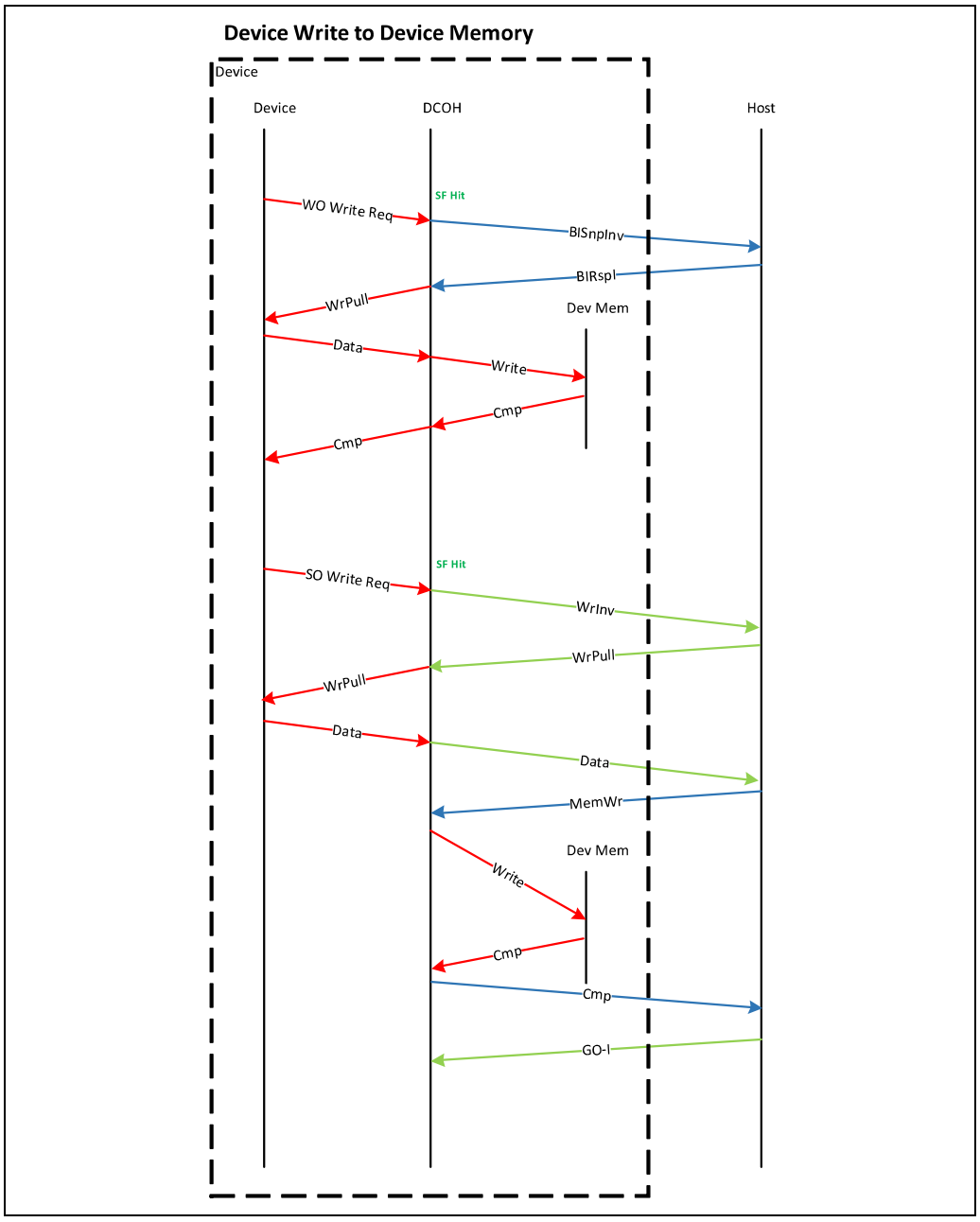
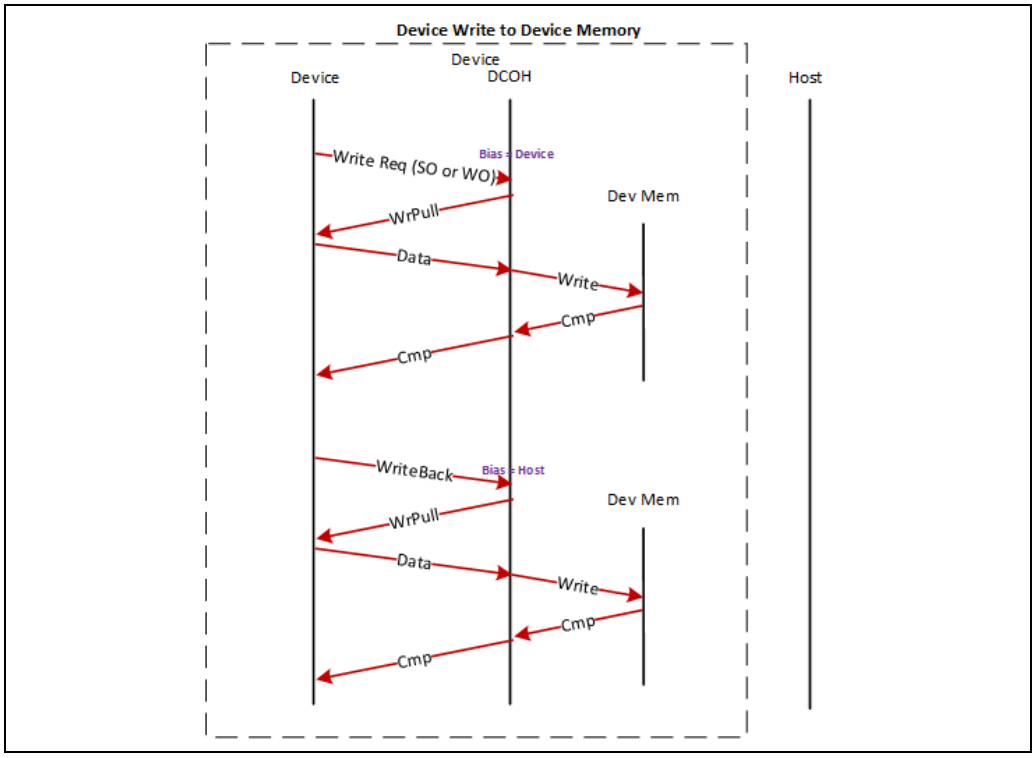


Figure 3-38 for HDM-DB is in contrast to Figure 3-37 for the HDM-D region. In the HDM-DB flow, the BISnp channel in the CXL.mem protocol is used to resolve coherence with the host for the internal weakly ordered write. The strongly ordered write follows the same flow for both HDM-DB and HDM-D.

Figure 3-39. Example Device Write to Device-attached Memory



Again, two flows are shown above in Figure 3-39. In the first case, if a weakly or strongly ordered write finds the line in Device Bias, the write can be completed entirely within the device without having to send any indication to the Host.

The second flow shows a device writeback to device-attached memory. Note that if the device is doing a writeback to device-attached memory, regardless of bias state, the request can be completed within the device without having to send a request to the Host.

The HDM-DB vs. HDM-D regions have the same basic assumption in these flows such that no interaction is required with the host.

Figure 3-40. Example Host to Device Bias Flip (HDM-D)

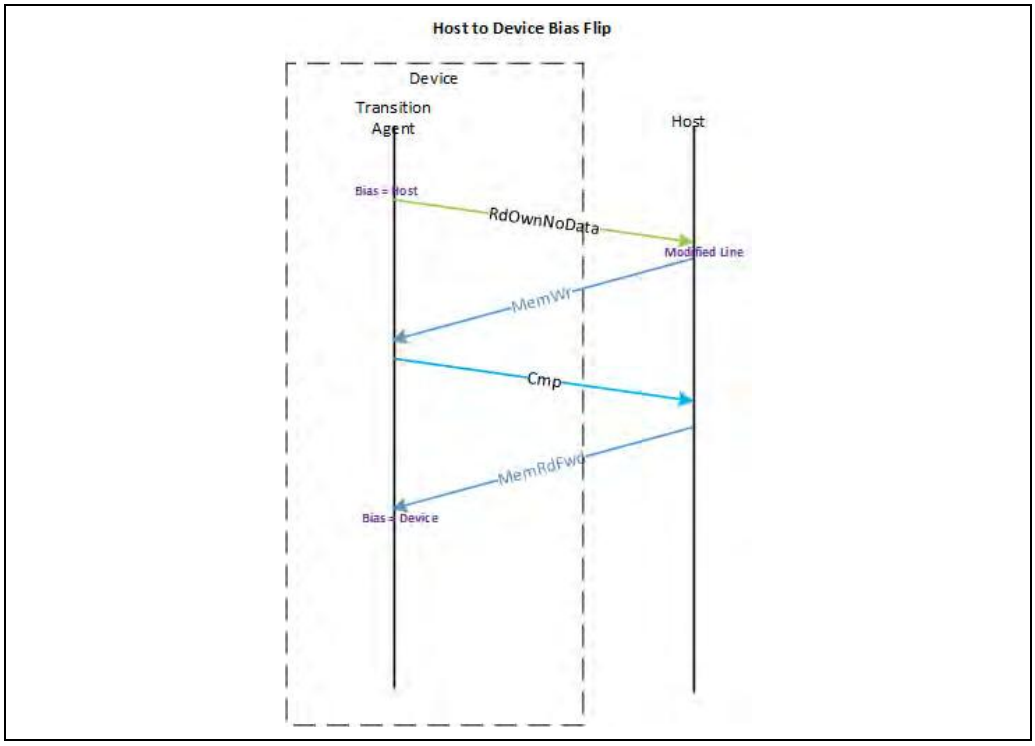


Figure 3-40 captures the "Bias Flip" flows for HDM-D memory. For the HDM-DB memory region, see Section 3.3.3 for details regarding how this case is handled. Please note that the MemRdFwd will carry the CQID of the RdOwnNoData transaction in the Tag. The reason for putting the RdOwnNoData completion (MemRdFwd) on CXL.mem is to ensure that subsequent M2S Req Channel requests from the Host to the same address are ordered behind the MemRdFwd. This allows the device to assume ownership of a line as soon as the device receives a MemRdFwd without having to monitor requests from the Host.

3.5.3 Type 2 Memory Flows and Type 3 Memory Flows

3.5.3.1 Speculative Memory Read

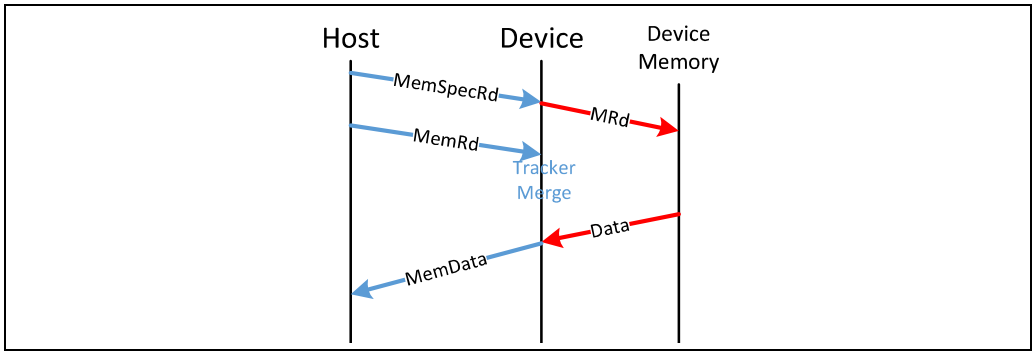
To support latency saving, CXL.mem includes a speculative memory read command (MemSpecRd) which is used to start memory access before the home agent has resolved coherence. This command does not receive a completion message and can be arbitrarily dropped. The host, after resolving coherence, may issue a demand read (i.e., MemRd or MemRdData) that the device should merge with the earlier MemSpecRd to achieve latency savings. See Figure 3-41 for an example of this type of flow.

The MemSpecRd command can be observed while another memory access is in progress in the device to the same cacheline address. In this condition, it is recommended that the device drops the MemSpecRd.

To avoid performance impact, it is recommended that MemSpecRd commands are treated as low priority to avoid adding latency to demand accesses. Under loaded conditions the MemSpecRd can hurt performance because of the extra bandwidth it

consumes and should be dropped when loading of memory or loading of the CXL link is detected. QoS Telemetry data as indicated by the DevLoad field is one way that loading of memory can be detected in the host or switch.

Figure 3-41. Example MemSpecRd

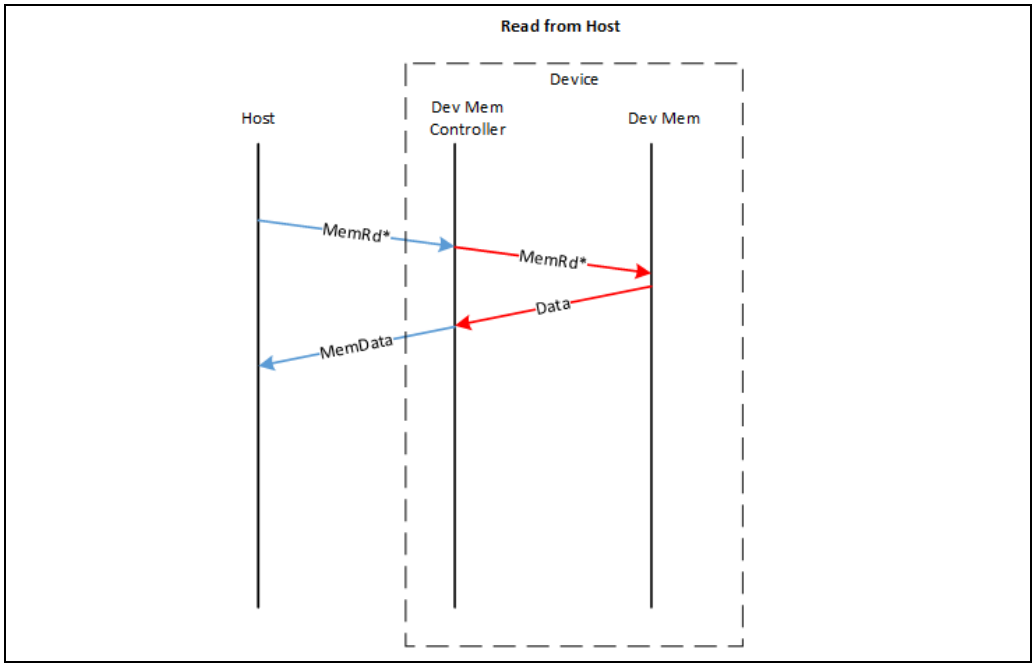


3.6 Flows to HDM-H in a Type 3 Device

The HDM-H address region in a Type 3 device is used as a memory expander or for Shared FAM device with software coherence where the device does not require active management of coherence with the Host. Thus, access to HDM-H does not use a DCOH agent. This allows the transaction flows to HDM-H to be simplified to just two classes, reads and writes, as shown below.

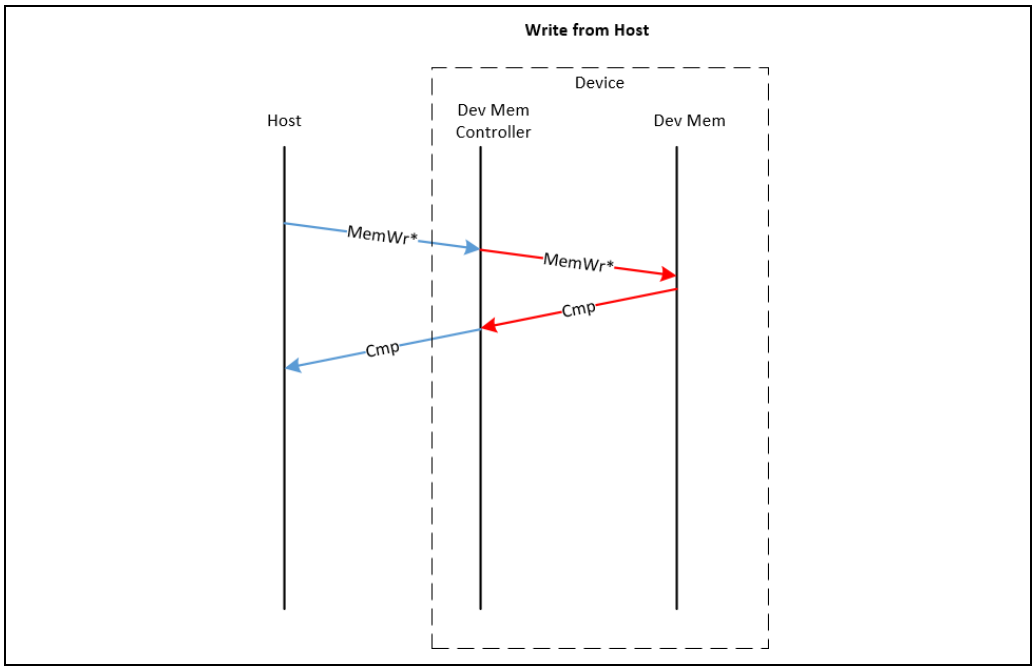
In Figure 3-42, the optimized read flow is shown for the HDM-H address region. In this flow, only a Data message is returned. In contrast, in the HDM-D/HDM-DB address region, both NDR and Data are returned. The legend shown in Figure 3-26 also applies to the transaction flows.

Figure 3-42. Read from Host to HDM-H



Unlike reads, writes to the HDM-H region use the same flow as the HDM-D/HDM-DB region and always complete with an S2M NDR Cmp message. This common write flow is shown in Figure 3-43.

Figure 3-43. Write from Host to All HDM Regions



§ §

4.0 CXL Link Layers

4.1 CXL.io Link Layer

The CXL.io link layer acts as an intermediate stage between the CXL.io transaction layer and the Flex Bus Physical layer. Its primary responsibility is to provide a reliable mechanism for exchanging transaction layer packets (TLPs) between two components on the link. The PCIe* Data Link Layer is utilized as the link layer for CXL.io Link layer. Please refer to chapter titled "Data Link Layer Specification" in PCIe Base Specification for details. In 256B Flit mode, the PCIe-defined PM and Link Management DLLPs are not applicable for CXL.io and must not be used.

Figure 4-1. Flex Bus Layers - CXL.io Link Layer Highlighted