Compute Express Link (CXL)-Cache/Mem Protocol Interface (CPI)

Specification

February 2023

Revision 1.0

Reference Number: 644330

Intel Corporation and its subsidiaries (collectively, "Intel") would like to receive input, comments, suggestions, and other feedback (collectively, "Feedback") on this specification. To be considered for incorporation into the specification, Feedback must be submitted by e-mail to: fabricspecification@intel.com. To the extent that You provide Intel with Feedback, You grant to Intel a worldwide, non exclusive, perpetual, irrevocable, royalty-free, fully paid, transferable license, with the right to sub-license, under Your Intellectual Property Rights, to make, use, sell, offer for sale, import, disclose, reproduce, make derivative works, distribute, or otherwise exploit Your Feedback without any accounting. As used in this paragraph, "Intellectual Property Rights" means, all worldwide copyrights, patents, trade secrets, and any other intellectual or industrial property rights, but excluding any trademarks or similar rights. By submitting Feedback, you represent that you are authorized to submit Feedback on behalf of your employer, if any, and that the Feedback is not confidential. Intel does not control or audit third-party data. You should review this content, consult other sources, and confirm whether referenced data are accurate.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software, or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure. Intel does not assume any liability for lost or stolen data or systems or any damages resulting from such losses.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document. The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Contents

Figures

Tables

Revision History

Revision Number	Description	Date
1.0	• Added *Compute Express Link (CXL) 3.0 Specification* support • Extended cmd parity protection to REQ/RSP Layers • Added byte enable parity protection to DATA layer • Added parameters for optional interface signals/capability • Increased DATA/RSP layer credit return protocol ID signal widths to 4 bits • Added IDE Containment Mode Latency Reduction feature support	February 2023
0.9	• Added *Compute Express Link (CXL) 2.0 Specification* support • Added parameters and clarifications for the disconnect flow • Added option for 32B transfer support	August 2021
0.7	• Initial Release	March 2020

1 Introduction

1.1 Terminology

The following acronyms and terms are used in this document:

Table 1-1. Terms and Definitions (Sheet 1 of 2)

Term	Definition
A2F	Agent-to-Fabric
Ack	Acknowledge
Agent	Agent refers to the SoC IP that connects to the fabric. The agent is on one side and the fabric is on the other of the CPI interface.
CHAN	Used to abstract the particular physical channel (REQ, DATA, RSP)
CPI	CXL-cache/mem Protocol Interface
CXL	Compute Express Link
D2H	Device to Host. Defined in the *CXL Specification*.
Device Trust Level	Field defined in the *CXL Specification*.
DRS	Data Responses. Defined in the *CXL Specification*.
EOP	End of Packet
F2A	Fabric-to-Agent
FLIT	**FL**ow Control Un**IT**. This term describes messages sent across CPI that generally express the amount of data passed on in one clock cycle on a CPI physical channel. On some physical channels, messages can consist of more than one FLIT. The term *Pump* has a similar meaning in the context of CPI and is a term more commonly used in fabrics to refer to data messages sent as multiple FLITs (or pumps).
FLOW_C	Flow Control
H2D	Host to Device. Defined in the *CXL Specification*.
Init signals	Signals used for Connect and Disconnect flows
M2S	Master to Subordinate. Defined in the *CXL Specification*.
NAck	Negatively acknowledge
QoS	Quality-of-Service
RAS	Reliability, Availability, and Serviceability
REQ	Request
RSP	Response
RTL	Register-Transfer Level—A design abstraction, which models synchronous digital circuits.
RwD	Requests with Data. Defined in the *CXL Specification*.
RX	Receiver
S2M	Subordinate to Master. Defined in the *CXL Specification*.
SFI	Streaming Fabric Interface
TX	Transmitter

Table 1-1. Terms and Definitions (Sheet 2 of 2)

Term	Definition
VC	Virtual channel
VN	Virtual network
XOR	Exclusive OR

1.2 Reference Documents

Currently, the *Compute Express Link (CXL)-Cache/Mem Protocol Interface (CPI)* supports the *CXL 3.0 Specification*.

Table 1-2. Reference Document and Locations

Document	Document Location
Streaming Fabric Interface (SFI) Specification	www.intel.com/content/www/us/en/design/resource-design-center.html
Compute Express Link™ Specification 3.0	www.computeexpresslink.org

2 Overview

This document describes the *CXL-cache/mem Protocol Interface (CPI) specification*, which has been developed to map coherent protocols between an agent and a fabric.

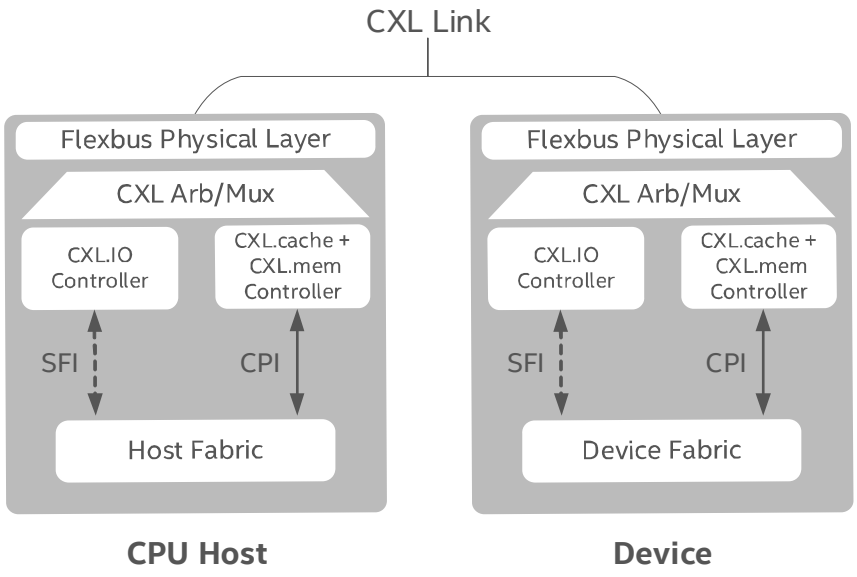
CPI allows mapping of different protocols on the same physical wires. Mapping of the CXL.cache and CXL.mem protocols to CPI is described in this specification.

Depending on whether the component is a Downstream port or an Upstream port, different channels of CXL.cache and CXL.mem become relevant for the Agent-to-Fabric (A2F) direction or the Fabric-to-Agent (F2A) direction. Figure 2-1 shows an example topology between a Host CPU and a Device.

The specific choices of channel mapping and physical wire sharing between different protocols are implementation-specific and is allowed by this specification. For example, Figure 2-2 shows an example system topology for an upstream port. CPI is used for coherent protocols such as CXL.cache and CXL.mem, whereas a Streaming Fabric Interface (SFI) is used for Load/Store protocols like PCI Express* (PCIe*) and CXL.io. For SFI details, see the *SFI Specification*.

Figure 2-2 shows example interface instantiations for CPI. The primary motivation for using CPI is to share wires at the fabric and achieve wire efficiency at the fabric and agent perimeters by allowing different protocols to share common wires. In this *CPI specification*, the channels of various protocols originating from agents are carefully mapped to a minimal set of physical channels and virtual channels, so that the bandwidth and channel isolation requirements of the agents and protocols are satisfied with the lowest total wire count.

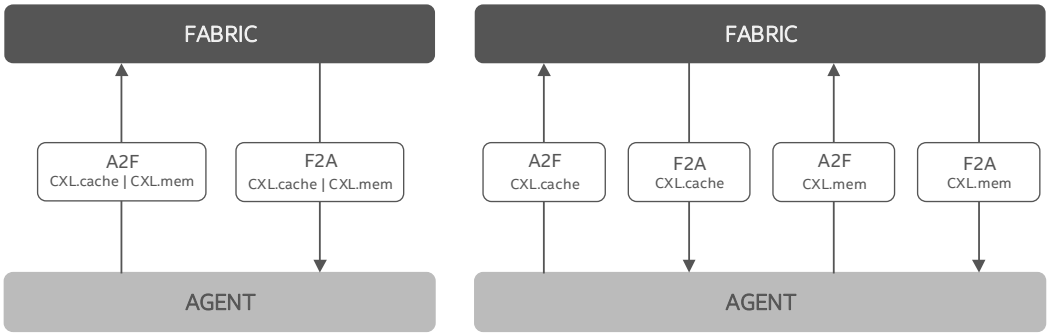
Figure 2-1. System Topology Example



CPI does not contain any new protocol definitions. It simply maps the existing protocol to a common set of channels. To maximize the wire sharing across various protocols, CPI requires protocols to use a common method of flow control and virtualization methods on their channels.

Furthermore, depending on the instantiation, certain protocols can be required to use common data widths and control signal widths. In this specification, the virtual channel concept is included for every mapped protocol. Ordering considerations within or across channels are also explained when applicable. When ordering considerations are not mentioned, assume all of the messages are unordered.

Figure 2-2. Example Topology and Interface Instantiations. Agent-to-Fabric (A2F), Fabric-to-Agent (F2A)



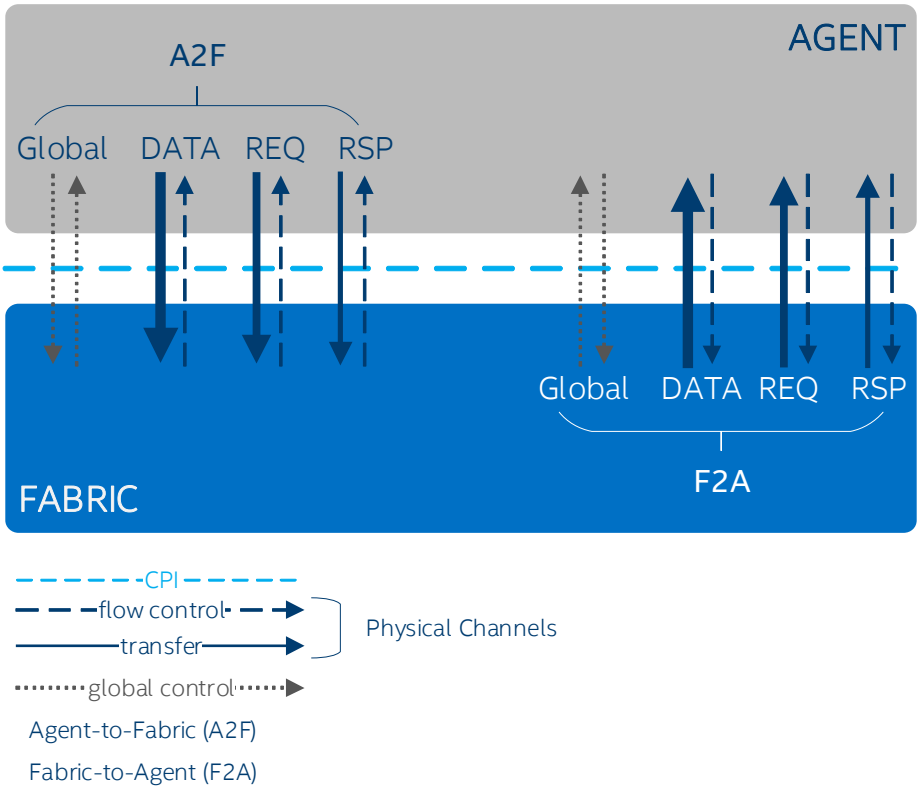
3 CPI Interface

CPI defines a standard interface between agents and the fabric. The agents can be a variety of IPs that are connected into the fabric and can have different profiles (Upstream ports or Downstream ports), as well as different protocol or bandwidth requirements. The fabric is expected to support the requirements of the agents within the bounds of CPI and the associated protocols tunneled on CPI.

3.1 CPI Physical Channels

The CPI interface defines three physical channels in each direction: Request (REQ), Response (RSP), and DATA. See Figure 3-1.

Figure 3-1. CPI Physical Channels: Agent-to-Fabric (A2F) and Fabric-to-Agent (F2A) Physical Channels



The REQ channel carries requests from agent-to-fabric and fabric-to-agent. Address and protocol-level command information for transactions are encapsulated in the **`Header`** field. This physical channel only supports transporting a single transaction per cycle, so the width is determined by the maximum width needed to transport a single request among all protocols sharing this physical channel.

The Data channel carries all messages, which have data transfer between agents. This includes write data, read response data, snoop response data, and so forth. The Data channel allows sending transactions with multiple **FL**ow Control Un**IT**s (FLITs). For example, a 64B transfer with a data channel carrying 32B of data is sent over two FLITs.

The RSP channel carries responses without data. For requests generated by agents, receiving agents from the fabric send-back responses using this physical channel. These responses can be completion, snoop responses, and so on. They do not carry address bits, so this channel has a narrower header field than REQ. This physical channel only transports single FLIT (**FL**ow Control Un**IT**) messages.

Common signals, which apply to all the physical channels, are referred to as Global. The signals in this group support initialization, disconnect, and miscellaneous error reporting.

The CPI physical channels are used by the different protocols that are being mapped to the CPI interface. To match the link bandwidth to the fabric bandwidth, multiple instantiations of the same channel for REQ, DATA, and RSP channels are allowed.

CPI does not require a protocol to use all the channels.

3.2 CPI Protocols Supported

Generally, supported protocols are not defined as part of CPI, but are defined in other documents. The mapping of CXL.cache and CXL.mem protocols are described in this document. For details about the CXL.cache protocol and the CXL.mem protocol, see *CXL Specification.*

3.3 CPI Protocol Profiles

Table 3-1 captures the profiles in terms of physical channels for CXL.cache and CXL.mem in the context of the agent-to-fabric connectivity of the Upstream port and Downstream port. The number of each of the physical channels used by an agent is an implementation choice and normally the decision is based on the bandwidth requirements of the agent.

Table 3-1. Physical Channels Per Protocol

Protocol Profile	A2F			F2A		
	REQ	RSP	DATA	REQ	RSP	DATA
CXL.CACHE (Upstream)	√	√	√	√	√	√
CXL.MEM (Upstream)	√	√	√	√	√	√
CXL.CACHE (Downstream)	√	√	√	√	√	√
CXL.MEM (Downstream)	√	√	√	√	√	√

Table 3-2 provides a mapping of the CXL.cache and CXL.mem message classes on to the CPI physical channels for these profiles.

Table 3-2. Message Channel to Physical Channel Mapping per Protocol

Protocol Profile	A2F			F2A		
	REQ	RSP	DATA	REQ	RSP	DATA
CXL.CACHE (Upstream)	H2D Req	H2D Rsp	H2D Data	D2H Req	D2H Rsp	D2H Data
CXL.MEM (Upstream)	M2S Req	M2S BIRsp	M2S RwD	S2M BISnp	S2M NDR	S2M DRS
CXL.CACHE (Downstream)	D2H Req	D2H Rsp	D2H Data	H2D Req	H2D Rsp	H2D Data
CXL.MEM (Downstream)	S2M BISnp	S2M NDR	S2M DRS	M2S Req	M2S BIRsp	M2S RwD

3.3.1 Redundant Interfaces

It is possible that some agents might expose only a single protocol per CPI interface and instead choose to replicate CPI interfaces. This choice can be made to simplify a design at the cost of using additional wires and logic.

4 CPI Physical Channel Description

CPI has one Global channel and a variable number of REQ, DATA, and RSP channels in each direction, between agent and fabric.

At a first level, the signals are grouped into the direction of the flow of data as Agent-to-Fabric (A2F) and Fabric-to-Agent (F2A).

4.1 Global Layer

The Global Layer carries signals that apply across all physical channels. Table 4-1 captures A2F signals and Table 4-2 captures F2A signals. The initialization flow that uses the Init signals is shown in Section 5.

Table 4-1. A2F Global Channel Wires

Signal Group	Signal Name	Direction	Width	Description
Init	txcon_req	Agent→Fabric	1	Connection request from transmitter (0→1 connection request, 1→0 disconnection request)
	rxcon_ack	Fabric→Agent	1	Connection acknowledge from receiver (0→1 connection acknowledge, 1→0 disconnection acknowledge)
	rxdiscon_nack	Fabric→Agent	1	Disconnection NAck (negatively acknowledge) from receiver
	rx_empty	Fabric→Agent	1	Receiver queues are empty for all channels and all credits have been returned
RAS	fatal	Agent→Fabric	1	Fatal error indication from agent (level signal)
	viral	Agent→Fabric	1	Optional signal: Used to signal viral from agent to fabric (level signal)
IDE	epoch_id	Agent→Fabric	P bits	Optional signal: Per-port MAC Epoch ID used to indicate MAC authentication status. See Section 4.6.6
	epoch_commit	Agent→Fabric	P bits	Optional signal: Per-port MAC Epoch authentication status. See Section 4.6.6
	epoch_reject	Agent→Fabric	P bits	Optional signal: Per-port MAC Epoch authentication status. See Section 4.6.6

Table 4-2. F2A Global Channel Wires

Signal Group	Signal Name	Direction	Width	Description
Init	txcon_req	Fabric→Agent	1	Connection request from transmitter (0→1 connection request, 1→0 disconnection request)
	rxcon_ack	Agent→Fabric	1	Connection acknowledge from receiver (0→1 connection acknowledge, 1→0 disconnection acknowledge)
	rxdiscon_nack	Agent→Fabric	1	Disconnection NAck from receiver
	rx_empty	Agent→Fabric	1	Receiver queues are empty for all channels and all credits have been returned
RAS	fatal	Fabric→Agent	1	Optional signal: Fatal error indication from fabric (level signal)
	viral	Fabric→Agent	1	Optional signal: Used to signal viral from fabric to agent (level signal)

4.2 REQ Layer

REQ Layer carries requests from agent-to-fabric and fabric-to-agent. Address and protocol level command information are encapsulated in the **`Header`** field of the REQ Layer.

The REQ Layer signals are symmetric in the A2F and F2A directions, even though protocols like CXL.cache and CXL.mem are not. Thus, the Upstream and Downstream versions of CXL.cache and CXL.mem map as different protocols.

Implementation only needs to support the relevant subset of the protocols required for functionality. The direction column in Table 4-3 specifies signal directions from the perspective of a Transmitter (TX) of packets and a Receiver (RX) of packets.

Table 4-3. Fields of the REQ Layer (Sheet 1 of 2)

Signal Class	Signal Name	Direction	Width	Description
VALID	req_is_valid	TX→RX	1 bit	Valid bit for the **`FLOW_C`** and **`HDR`**
	req_block	RX→TX	1 bit	Optional Signal: Transient back pressure from RX due to rate mismatch on a clock crossing at the interface boundary
	req_early_valid	TX→RX	1 bit	Optional Signal. It is an early valid indication from TX to RX indicating that TX has packets to send. See Section 4.5.1 for rules related to this signal.

Table 4-3. Fields of the REQ Layer (Sheet 2 of 2)

Signal Class	Signal Name	Direction	Width	Description
FLOW_C	req_protocol_id	TX→RX	4 bits	Identifies between protocols, if multiple protocols are using the same physical wires. 4'h1000: Upstream Port CXL.cache 4'h1001: Upstream Port CXL.mem 4'h1010: Downstream Port CXL.cache 4'h1011: Downstream Port CXL.memOther encodings are reserved. Optional for agents with single Protocol.
	req_vc_id	TX→RX	4 bits	Indicates the virtual channel used for the corresponding packet. It also identifies the flow control used when **req_shared_credit** = 0. Optional for physical channels with only a single VC per protocol.
	req_shared_credit	TX→RX	1 bit	Indicates if the HDR uses shared (a value of 1) or dedicated credits (a value of 0). Optional if only dedicated credits are used.
HDR	req_header	TX→RX	H_REQ bits	Header information. This is protocol specific, see mapping in Section 4.2.1 and Section 4.2.1.1.
	req_cmd_parity	TX→RX	1 bit	Optional signal: XOR of **req_header** fields and **req_spid** and **req_dpid** fields if supported.
	req_spid	TX→RX	12 bits	Optional signal: Source PBR ID when operating in PBR Flit mode
	req_dpid	TX→RX	12 bits	Optional signal: Destination PBR ID when operating in PBR Flit Mode
CREDIT	req_rxcrd_valid	RX→TX	1 bit	Indicates a valid dedicated credit return.
	req_rxcrd_protocol_id	RX→TX	4 bits	Identifies which protocol the credits are returned to. 4'h1000: Upstream Port CXL.cache 4'h1001: Upstream Port CXL.mem 4'h1010: Downstream Port CXL.cache 4'h1011: Downstream Port CXL.mem Other encodings are reserved. Optional for agents with a single protocol.
	req_rxcrd_vc_id	RX→TX	4 bits	Virtual channel for which credit is returned. Optional for physical channels with only a single VC per protocol.
	req_rxcrd_shared	RX→TX	1 bit	Indicates a shared credit return. Optional if only dedicated credits are used.
	req_txblock_crd_flow	TX→RX	1 bit	Optional Signal: TX requesting RX to block the credit returns due to transient back pressure. An example is a clock crossing First-In First-Out (FIFO).

The HDR size is variable and is based on the protocol that is being transported over the CPI interface. When multiple protocols are carried over the CPI interface, the HDR width is sized for the maximum size of the HDR being transported over the CPI interface. The reserved field width is primarily used to cover the unused portion of the HDR. The transmitter must drive 0 on the reserved field and the receiver must ignore this field.When supported, **req_cmd_parity** provides parity protection over **req_header** and **req_spid/req_dpid** from the HDR class if present. REQ HDR parity errors are fatal to interface operation and must be reported as fatal errors. When not supported, it must be reserved; TX must drive 0, and RX must not check REQ HDR parity.

4.2.1 Mapping CXL.cache to HDR

The widths of the different fields (except for **AddressParity**) are given as per the *CXL Specification*. **AddressParity** is computed as an XOR of all the bits of the **Address** field. For the upstream port, A2F corresponds to H2D channel on the Compute Express Link (CXL), and F2A corresponds to the D2H channel on CXL. For the downstream port, A2F corresponds to D2H channel on CXL, and F2A corresponds to H2D channel on CXL. For a Downstream port, the **Device Trust Level** field defined in the CXL Security Policy Register is also a part of D2H requests. Currently only a single virtual channel is supported on these channels for CXL.cache.

Table 4-4. Mapping CXL.Cache Protocol to req_header for an Upstream Port

Agent-to-Fabric REQ (A2F/H2D)		Fabric-to-Agent REQ (F2A/D2H)	
Field	**Position**	**Field**	**Position**
Opcode[2:0]	[2:0]	Opcode[4:0]	[4:0]
UQID[11:0]	[14:3]	CQID[11:0]	[16:5]
AddressParity	[15]	NT	[17]
Address [51:6]	[61:16]	Reserved	[19:18]
CacheID	[65:62]	AddressParity	[20]
Flit Mode	[67:66]	Address [51:6]	[66:21]
Epoch Valid	[68:68]	CacheID	[70:67]
Epoch ID	[69:69]	Flit Mode	[72:71]
Port ID	[NP+70:70]	Reserved	[H_REQ-1:73]
Reserved	[H_REQ-1:NP+71]	--	--

Table 4-5. Mapping CXL.cache Protocol to req_header for a Downstream Port

Agent-to-Fabric REQ (A2F/D2H)		Fabric-to-Agent REQ (F2A/H2D)	
Field	**Position**	**Field**	**Position**
Opcode[4:0]	[4:0]	Opcode[2:0]	[2:0]
CQID[11:0]	[16:5]	UQID[11:0]	[14:3]
NT	[17]	AddressParity	[15]
Device_Trust_Level[1:0]	[19:18]	Address [51:6]	[61:16]
AddressParity	[20]	CacheID	[65:62]
Address [51:6]	[66:21]	Flit Mode	[67:66]
CacheID	[70:67]	Reserved	[H_REQ-1:68]
Flit Mode	[72:71]	--	--
Epoch Valid	[73:73]	--	--
Epoch ID	[74:74]	--	--
Port ID	[NP+75:75]	--	--
Reserved	[H_REQ-1:NP+76]	--	--

- **CacheID** is only applicable for 256B Flit mode and is reserved for all other modes.
- **Flit Mode** indicates the CXL Flit Mode using the following encodings:

  - 2'b00: 68B Flit
  - 2'b01: 256B Flit
  - 2'b10: PBR Flit
  - 2'b11: Reserved
- **`Flit Mode`** is only applicable for the Upstream Port F2A and Downstream Port A2F directions when FM_ENC_D2H_S2M=1 and reserved otherwise.
- **`Flit Mode`** is only applicable for the Upstream Port A2F and Downstream Port F2A directions when FM_ENC_H2D_M2S=1 and reserved otherwise.

4.2.1.1 Ordering Rules

When multiple instantiations of REQ channel are implemented (to match link bandwidth to fabric bandwidth), the following ordering rules are applicable to maintain the ordering semantics outlined in *CXL Specification*:

1. Concurrent messages on the same clock cycle are unordered with respect to each other.
2. Responses received on the RSP channel(s) must be considered to be ahead of requests received on the REQ channel on the same clock cycle.

4.2.2 Mapping CXL.mem to HDR

The widths of the different fields (except for **`AddressParity`**) are given as per *CXL Specification*. **`AddressParity`** is computed as an XOR of all the bits of the **`Address`** field. For an upstream port, A2F maps to M2S REQ channeland F2A maps to S2M BISnp channel on CXL.mem. For a downstream port, F2A maps to M2S REQ channel and A2F maps to S2M BISnp channel on CXL.mem. It is permitted for implementations to support multiple VC to facilitate or enhance Quality of Service (QoS) policies for HDM accesses. QoS across the interface is enforced by either giving enough dedicated credits for a VC, and/or by having TX implement watermarks to prevent other VCs from taking significant fraction of shared credits. Both agent and fabric must be consistent with respect to the policy for VC assignment. Implementations must also make sure not to violate any ordering or QoS policy given by the *CXL Specification*.

Table 4-6. Mapping CXL.Mem Protocol to req_header for an Upstream Port (Sheet 1 of 2)

Agent-to-Fabric REQ (A2F/M2S)		Fabric-to-Agent REQ (F2A/S2M)	
Field	**Position**	**Field**	**Position**
MemOpcode[3:0]	[3:0]	Opcode[3:0]	[3:0]
Tag[15:0]	[19:4]	BI-ID[11:0]	[15:4]
TC[1:0]	[21:20]	BITag[11:0]	[27:16]
SnpType[2:0]	[24:22]	AddressParity	[28]
Address[5]	[25]	Address[51:6]	[74:29]
Metafield[1:0]	[27:26]	Flit Mode	[76:75]
Metavalue[1:0]	[29:28]	Reserved	[H_REQ-1:77]
AddressParity	[30]	--	--
Address [51:6]	[76:31]	--	--
LDID[3:0]	[80:77]	--	--
FlitMode	[82:81]	--	--

Table 4-6. Mapping CXL.Mem Protocol to req_header for an Upstream Port (Sheet 2 of 2)

Agent-to-Fabric REQ (A2F/M2S)		Fabric-to-Agent REQ (F2A/S2M)	
Field	**Position**	**Field**	**Position**
Epoch Valid	[83:83]	--	--
Epoch ID	[84:84]	--	--
Port ID	[NP+85:85]	--	--
Reserved	[H_REQ-1:NP+86]	--	--

Table 4-7. Mapping CXL.mem Protocol to req_header for a Downstream Port

Agent-to-Fabric REQ (A2F/S2M)		Fabric-to-Agent REQ (F2A/M2S)	
Field	**Position**	**Field**	**Position**
Opcode[3:0]	[3:0]	MemOpcode[3:0]	[3:0]
BI-ID[11:0]	[15:4]	Tag[15:0]	[19:4]
BITag[11:0]	[27:16]	TC[1:0]	[21:20]
AddressParity	[28]	SnpType[2:0]	[24:22]
Address[51:6]	[74:29]	Address[5]	[25]
Flit Mode	[76:75]	Metafield[1:0]	[27:26]
Epoch Valid	[77:77]	Metavalue[1:0]	[29:28]
Epoch ID	[78:78]	AddressParity	[30]
Port ID	[NP+79:79]	Address [51:6]	[76:31]
Reserved	[H_REQ-1:NP+80]	LDID[3:0]	[80:77]
--	--	FlitMode	[82:81]
--	--	Reserved	[H_REQ-1:83]

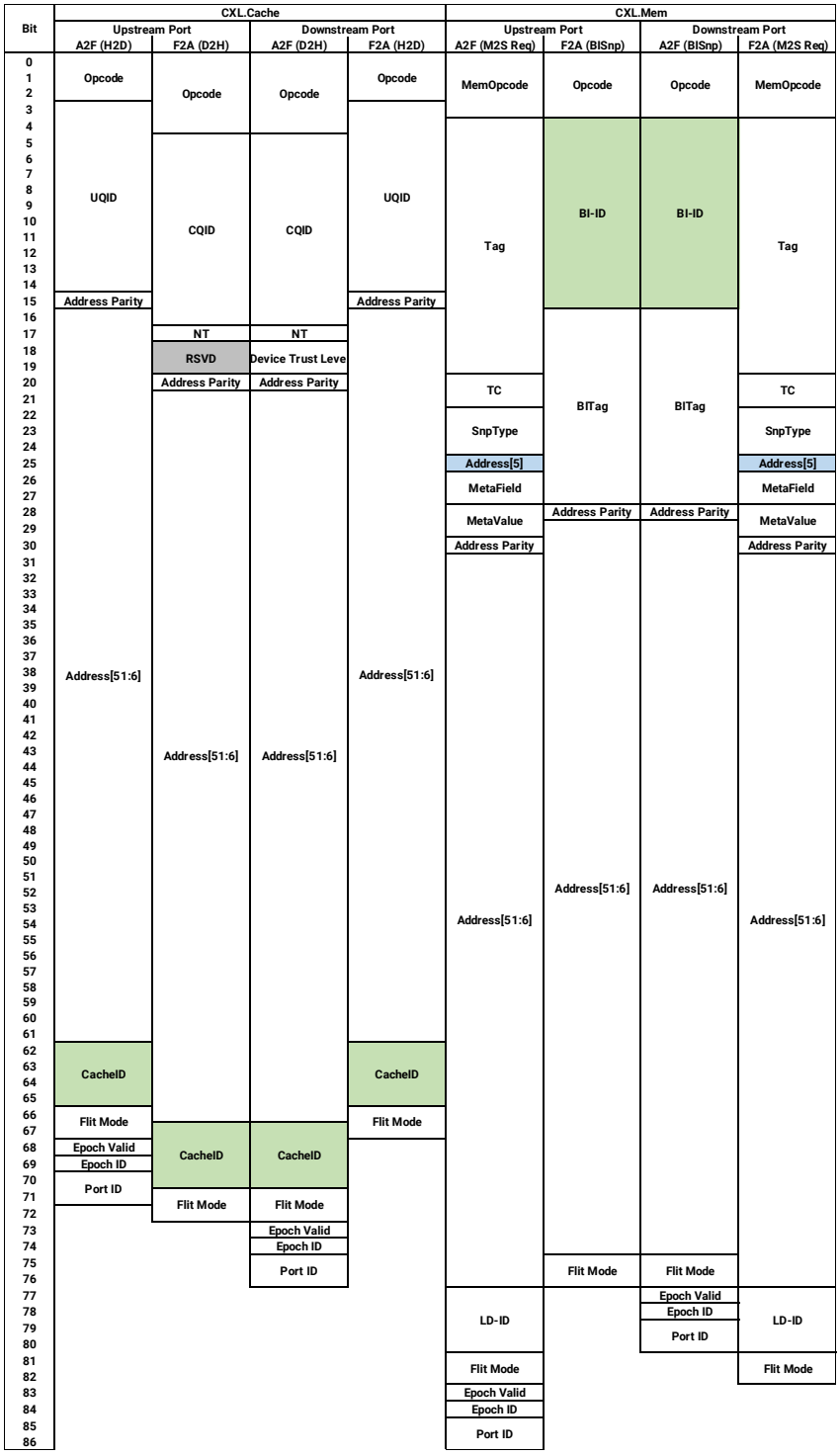
- **`Address[5]`** is only applicable for 68B Flit mode and is reserved for all other modes
- **`BI-ID`** is only applicable for 256B Flit mode and is reserved for all other modes.
- **`LD-ID`** is only applicable for 68B/256B Flit modes and is reserved for PBR Flit mode
- **`Flit Mode`** indicates the CXL Flit Mode using the following encodings:
  - 2'b00: 68B Flit
  - 2'b01: 256B Flit
  - 2'b10: PBR Flit
  - 2'b11: Reserved
- **`Flit Mode`** is only applicable for the Upstream Port F2A and Downstream Port A2F directions when FM_ENC_D2H_S2M=1 and reserved otherwise.
- **`Flit Mode`** is only applicable for the Upstream Port A2F and Downstream Port F2A directions when FM_ENC_H2D_M2S=1 and reserved otherwise.

4.2.2.1 Ordering Rules

When multiple instantiations of REQ channel are implemented (to match the link bandwidth to fabric bandwidth), it is required to map CXL.mem requests to a specific instance using a hash based on address. The specific hash is implementation specific, but every address must map to only one instance. This ensures that the M2S channel ordering outlined in *CXL Specification* is maintained.

4.2.3 Header Formats

Figure 4-1. REQ Channel Header Formats



4.3 DATA Layer

The DATA physical channel carries all messages, which have data transfer between agents. This can include write data, read response data, snoop response data, and so forth. The data physical channel messages with data can have multiple FLITs. The Data Layer signals are symmetric in the A2F and F2A directions, even though protocols such as CXL.cache and CXL.mem are not. Thus, the Upstream and Downstream versions of CXL.cache and CXL.mem map are different protocols.

Implementation only needs to support the relevant subset of the protocols required for functionality. The direction column in Table 4-8 specifies signal directions from the perspective of a Transmitter (TX) of packets and a Receiver (RX) of packets.

Table 4-8. Fields of Data Layer (Sheet 1 of 2)

Signal Class	Signal Name	Direction	Width	Description
VALID	data_is_valid	TX→RX	1 bit	Valid bit for **FLOW_C** and **HDR**. PAYLOAD and EOP have a fixed timing relationship from HDR based on *DataHdrSep* parameter. See Section 4.3.1.
	data_block	RX→TX	1 bit	Optional Signal: Transient back pressure from RX due to rate mismatch on a clock crossing at the interface boundary
	data_early_valid	TX→RX	1 bit	Optional Signal. It is an early valid indication from TX to RX indicating that TX has packets to send. See Section 4.5.1 for rules related to this signal.
FLOW_C	data_protocol_id	TX→RX	4 bits	Identifies the protocol: 4'h1000: Upstream Port CXL.cache 4'h1001: Upstream Port CXL.mem 4'h1010: Downstream Port CXL.cache 4'h1011: Downstream Port CXL.memOther encodings are reserved. Must be asserted as the same value for all pumps of a packet. Optional for physical channels with a single protocol.
	data_vc_id	TX→RX	4 bits	Indicates the virtual channel used for the corresponding packet. Must be asserted as the same value for all pumps of a packet. It also identifies the flow control used when **data_shared_credit** = 0. Optional for physical channels with only a single VC per protocol.
	data_shared_credit	TX→RX	1 bit	Indicates if the HDR uses shared (a value of 1) or dedicated (a value of 0) credits. Must be asserted as the same value for all pumps of a packet. Optional if only dedicated credits are used.

Table 4-8. Fields of Data Layer (Sheet 2 of 2)

Signal Class	Signal Name	Direction	Width	Description
HDR	data_header	TX→RX	H_DAT bits	Header information
	data_sz	TX→RX	1 bit	Optional signal: This is required if 32B transfers need to be supported (*TXFER_32B_EN*=1). When 1, it indicates 64B transfer. When 0, it indicates 32B transfer.
	data_cmd_parity	TX→RX	1 bit	Optional signal: XOR of **data_header** fields and **data_spid** and **data_dpid** fields if supported.
	data_spid	TX→RX	12 bits	Optional signal: Source PBR ID when operating in PBR Flit mode
	data_dpid	TX→RX	12 bits	Optional signal: Destination PBR ID when operating in PBR Flit mode
PAYLOAD	data_body	TX→RX	D bytes	Data associated with the header. D should be a multiple of 16 Bytes.
	data_byte_enable	TX→RX	D bits	1 bit per byte of **data_body**
	data_byte_enable_parity	TX→RX	1 bit	Optional signal: XOR of **data_byte_enable** bits
	data_poison	TX→RX	1 bit	Indication that the corresponding data chunk is corrupted and should not be used by RX.
	data_parity	TX→RX	P bits	P = (D/16)*2. Two bits of parity are used for every 16B of data. **data_parity**[n] = xor(**data_body**[63+64*n:64*n])
EOP	data_eop	TX→RX	1 bit	End of Packet indication
CREDIT	data_rxcrd_valid	RX→TX	1 bit	Indicates a valid dedicated credit return.
	data_rxcrd_protocol_id	RX→TX	4 bits	Identifies which protocol the credits are returned to. 4'h1000: Upstream Port CXL.cache 4'h1001: Upstream Port CXL.mem 4'h1010: Downstream Port CXL.cache 4'h1011: Downstream Port CXL.mem Other encodings are reserved. Optional for agents with a single protocol.
	data_rxcrd_vc_id	RX→TX	4 bits	Virtual channel for which credit is returned. Optional for physical channels with only a single VC per protocol.
	data_rxcrd_shared	RX→TX	1 bit	Shared credit return. Optional if only dedicated credits are used.
	data_txblock_crd_flow	TX→RX	1 bit	Optional Signal: TX requesting RX to block the credit returns due to transient back pressure. An example is a clock crossing FIFO.

The HDR size is variable and is based on the protocol being transported over the CPI interface. When multiple protocols are carried over the CPI interface, the HDR width is sized for the maximum size of the HDR being transported over the CPI interface. Reserved field width is used to cover the unused portion of the HDR. The transmitter must drive 0 on the reserved field, and the receiver must ignore this field. Most messages in the supported protocols carry 64B of data. Messages with 32B payloads are supported on the DATA channel. In both cases, a 64B worth credit is used.

It is strongly recommended that unused data bytes that do not have their corresponding byte enable set be driven to 0 by the transmitter to avoid data being inadvertently leaked. It is strongly recommended that receivers also ensure that the values provided in those non-enabled bytes be discarded by hardware, such that the values cannot be visible to software.

When supported, **data_cmd_parity** provides parity protection over **data_header** and **data_spid/data_dpid** from the HDR class if present. DATA HDR parity errors are fatal to interface operation and must be reported as fatal errors. When not supported, it must be reserved; TX must drive 0, and RX must not check DATA HDR parity.

4.3.1 Header and Payload Separation

As defined by the *DataHdrSep* parameter, the payload follows the sending of the header by a fixed separation of 0 to 3 cycles. This parameter is defined in each direction (A2F and F2A) to allow independent control. The value set applies to all protocols on the given CPI. The fixed separation allows the payload to be sent without a separate valid indication on the payload.

4.3.2 Stall and Blocking Example

Valid might deassert in the middle of a message, which pauses the transfer until valid is re-asserted. The example in Figure 4-2 shows the first message sent without a stall, and the second message sent with a one cycle stall after pump0.

Figure 4-2. Data Transfer Stall

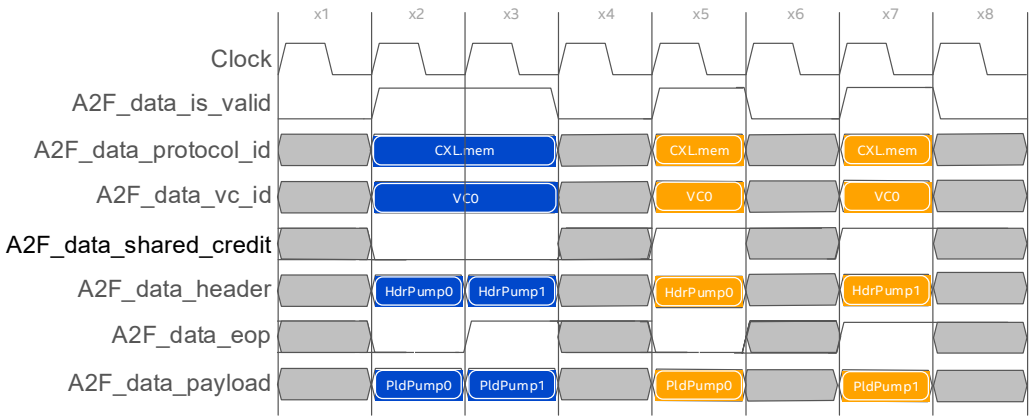
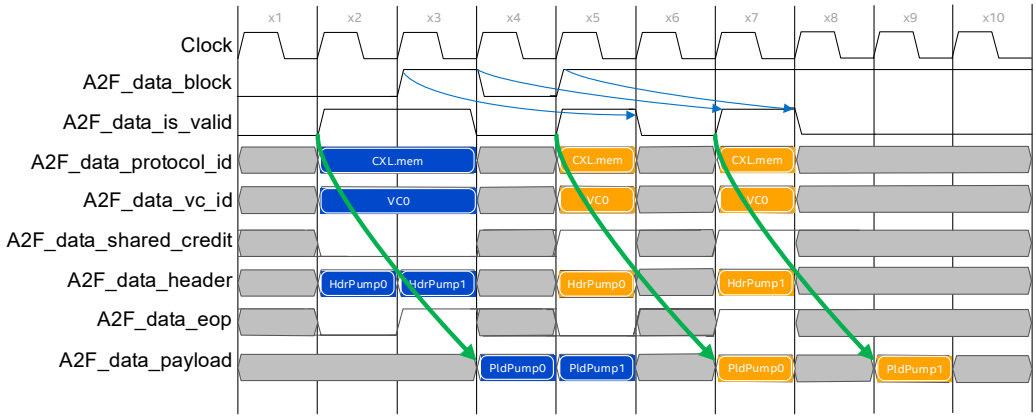


Figure 4-3 shows an example of a forced block condition from the receiver using the **A2F_data_block**, which requires the sender to deassert the **data_is_valid** signal. In this example, the payload offset value of 2 is defined by the *A2F_DataHdrSep* parameter. The arrows from **A2F_data_is_valid** to **A2F_data_payload** highlight this offset relationship. The fabric blocking has a reaction time of three cycles where the cycles are defined by the *FabricBlocking* parameter. This relationship is highlighted in the figure with arrows from the **A2F_data_block** signal to **A2F_data_is_valid**.

Figure 4-3. Data Transfer Block with Data Offset



4.3.3 Intra-Packet Interleaving

Intra-packet level interleave is not supported within a protocol or across protocols. This means that after a packet begins transmission across the interface, it must be sent across until End of Packet (EOP) is reached, before a different packet from the same or a different protocol can begin transmission. This is a simplification that allows for simplified fabric and agent design.

Note: Rule 8 in Section 3.2.5.13 of *CXL 1.1 Specification* allowed H2D data transfers to be interleaved and that is the only exception allowed in CPI. Since these packets are expected to be pre-allocated in the application layer, they will always drain and could be considered as independent 32B transfers on CPI. Starting with CXL 2.0, this rule was changed in the *CXL Specification* NOT to allow interleaving and the exception does not apply for CXL 2.0.

4.3.4 Mapping CXL.cache to HDR

For an interface carrying 64B of data, 64B transfers are sent over 1 cycle. The entire header must be sent over 1 cycle as well. For 64B wide **data_body**, *TXFER_32B_EN* must be set to 0. 32B protocol packet transfers are achieved by setting the appropriate byte enables and doing a 64B transfer across the interface.

For an interface carrying 32B of data, **data_body** is 256 bits wide, and 64B transfers are sent over 2 cycles. The **data_eop** signal should assert on the second cycle, and **data_header** is valid on the first cycle, and the second cycle is reserved.

For an interface carrying 16B of data, **data_body** is 128 bits wide, and 64B transfers are sent over 4 cycles. The **data_eop** signal should assert on cycle 4, **data_header** is valid on the first cycle, and the second, third, and fourth cycles are reserved.

Mapping of CXL.cache data header fields to **data_header** is shown in Table 4-9 and Table 4-10 for upstream and downstream ports, respectively.

- For upstream ports, A2F corresponds to H2D from CXL and F2A corresponds to D2H from CXL.
- For downstream ports, A2F corresponds to D2H and F2A corresponds to H2D.
- Currently, only a single virtual channel is supported on these channels for CXL.cache.

In the tables below, ChunkValid is only relevant for 32B transfers (*TXFER_32B_EN*=1 and **data_sz**=0). RX must ignore ChunkValid if 32B transfers are not supported (*TXFER_32B_EN*=0) or if **data_sz** = 1. Implementations must abide by the rules in the *CXL Specification* for 32B transfers.

Table 4-9. Mapping CXL.cache Protocol to data_header for an Upstream Port

Agent-to-Fabric DATA (A2F)		Fabric-to-Agent DATA (F2A)	
Field	**Position**	**Field**	**Position**
Go-Err	[0]	UQID	[11:0]
Reserved	[7:1]	Reserved	[12]
CQID	[19:8]	Bogus	[13]
ChunkValid	[20]	ChunkValid	[14]
CacheID	[24:21]	Flit Mode	[16:15]
Flit Mode	[26:25]	Reserved	[H_DAT-1:17]
Epoch Valid	[27:27]	--	--
Epoch ID	[28:28]	--	--
Port ID	[NP+29:29]	--	--
Reserved	[H_DAT-1:NP+30]	--	--

Table 4-10. Mapping CXL.cache Protocol to data_header for a Downstream Port

Agent-to-Fabric DATA (A2F)		Fabric-to-Agent DATA (F2A)	
Field	**Position**	**Field**	**Position**
UQID	[11:0]	Go-Err	[0]
Reserved	[12]	Reserved	[7:1]
Bogus	[13]	CQID	[19:8]
ChunkValid	[14]	ChunkValid	[20]
Flit Mode	[16:15]	CacheID	[24:21]
Epoch Valid	[17:17]	Flit Mode	[26:25]
Epoch ID	[18:18]	Reserved	[H_DAT-1:27]
Port ID	[NP+19:19]	--	--
Reserved	[H_DAT-1:NP+20]	--	--

- **CacheID** is only applicable for 256B Flit mode and is reserved for all other modes.
- **ChunkValid** is only applicable for 68B Flit mode and is reserved for all other modes.
- **Flit Mode** indicates the CXL Flit Mode using the following encodings:
  - 2'b00: 68B Flit
  - 2'b01: 256B Flit
  - 2'b10: PBR Flit
  - 2'b11: Reserved
- **Flit Mode** is only applicable for the Upstream Port F2A and Downstream Port A2F directions when FM_ENC_D2H_S2M=1 and reserved otherwise.

- **`Flit Mode`** is only applicable for the Upstream Port A2F and Downstream Port F2A directions when FM_ENC_H2D_M2S=1 and reserved otherwise.

4.3.5 Mapping CXL.mem to HDR

For an interface carrying 64B of data, 64B transfers are sent over one cycle. The entire header must be sent over one cycle as well.

For an interface carrying 32B of data, **`data_body`** is 256-bits wide, and 64B transfers are sent over two cycles. The **`data_eop`** signal should assert on cycle 2, and **`data_header`** is split evenly between the two cycles. If **`data_header`** is H bits wide, H must be made even by padding with a Reserved bit if required. H/2 bits ([H/2-1:0]) are sent on the first cycle and the remaining bits are sent on the second cycle. Implementations are permitted to have a wide header signal and send the entire **`data_header`** on the first cycle. *MEM_DATHDR_SPLIT* is the parameter that determines this, it is set to 1 if the header is split evenly over multiple cycles.

For an interface carrying 16B of data, **`data_body`** is 128-bits wide, and 64B transfers are sent over four cycles. The **`data_eop`** signal should assert on cycle four, and **`data_header`** is split evenly between the four cycles. If **`data_header`** is H bits wide, H must be made a multiple of four by padding with Reserved bits if required. H/4 bits ([H/4-1:0]) are sent on the first cycle, ([H/2-1:H/4]) are sent on the second cycle, ([3H/4-1:H/2]) are sent on the third cycle and the remaining bits are sent on the fourth cycle. Implementations are permitted to have a wide header signal and send the entire **`data_header`** on the first cycle. *MEM_DATHDR_SPLIT* is the parameter that determines this, it is set to 1 if the header is split evenly over multiple cycles.

Mapping of CXL.mem data header fields to **`data_header`** is shown in Table 4-11 and Table 4-12 for upstream and downstream ports, respectively.

- For upstream ports, A2F corresponds to M2S RwD from CXL and F2A corresponds to S2M DRS from CXL.
- For downstream ports, A2F corresponds to S2M DRS and F2A corresponds to M2S RwD.

Upstream ports implementing Egress Port Congestion Measurement mechanisms are permitted to override DevLoad values received from the fabric based on the algorithms described in the *CXL Specification*.

It is permitted for implementations to support multiple VC to facilitate or enhance QoS policies for HDM accesses. QoS across the interface is enforced by either giving enough dedicated credits for a VC, and/or by having TX implement watermarks to prevent other VCs from taking significant fraction of shared credits. Both agent and fabric must be consistent with respect to the policy for VC assignment. Implementations must also make sure not to violate any ordering or QoS policy given by the *CXL Specification*.

Table 4-11. Mapping CXL.mem Protocol to data_header for an Upstream Port (Sheet 1 of 2)

Agent-to-Fabric DATA (A2F)		Fabric-to-Agent DATA (F2A)	
Field	**Position**	**Field**	**Position**
MemOpcode	[3:0]	Opcode	[2:0]
MetaField[1:0]	[5:4]	Reserved	[3]
MetaValue[1:0]	[7:6]	MetaField[1:0]	[5:4]
SnpType[2:0]	[10:8]	MetaValue[1:0]	[7:6]

Table 4-11. Mapping CXL.mem Protocol to data_header for an Upstream Port (Sheet 2 of 2)

Agent-to-Fabric DATA (A2F)		Fabric-to-Agent DATA (F2A)	
Field	**Position**	**Field**	**Position**
TC[1:0]	[12:11]	Reserved	[15:8]
Reserved	[14:13]	Tag	[31:16]
AddressParity	[15]	LD-ID[3:0]	[35:32]
Address[Even Indices]	[38:16]	DevLoad[1:0]	[37:36]
Tag	[54:39]	Flit Mode	[39:38]
Address[Odd Indices]	[77:55]	Reserved	[H_DAT-1:40]
LDID[3:0]	[81:78]	--	--
Flit Mode	[83:82]	--	--
Epoch Valid	[84:84]	--	--
Epoch ID	[85:85]	--	--
Port ID	[NP+86:86]	--	--
Reserved	[H_DAT-1:NP+87]	--	--

Table 4-12. Mapping CXL.mem Protocol to data_header for a Downstream Port

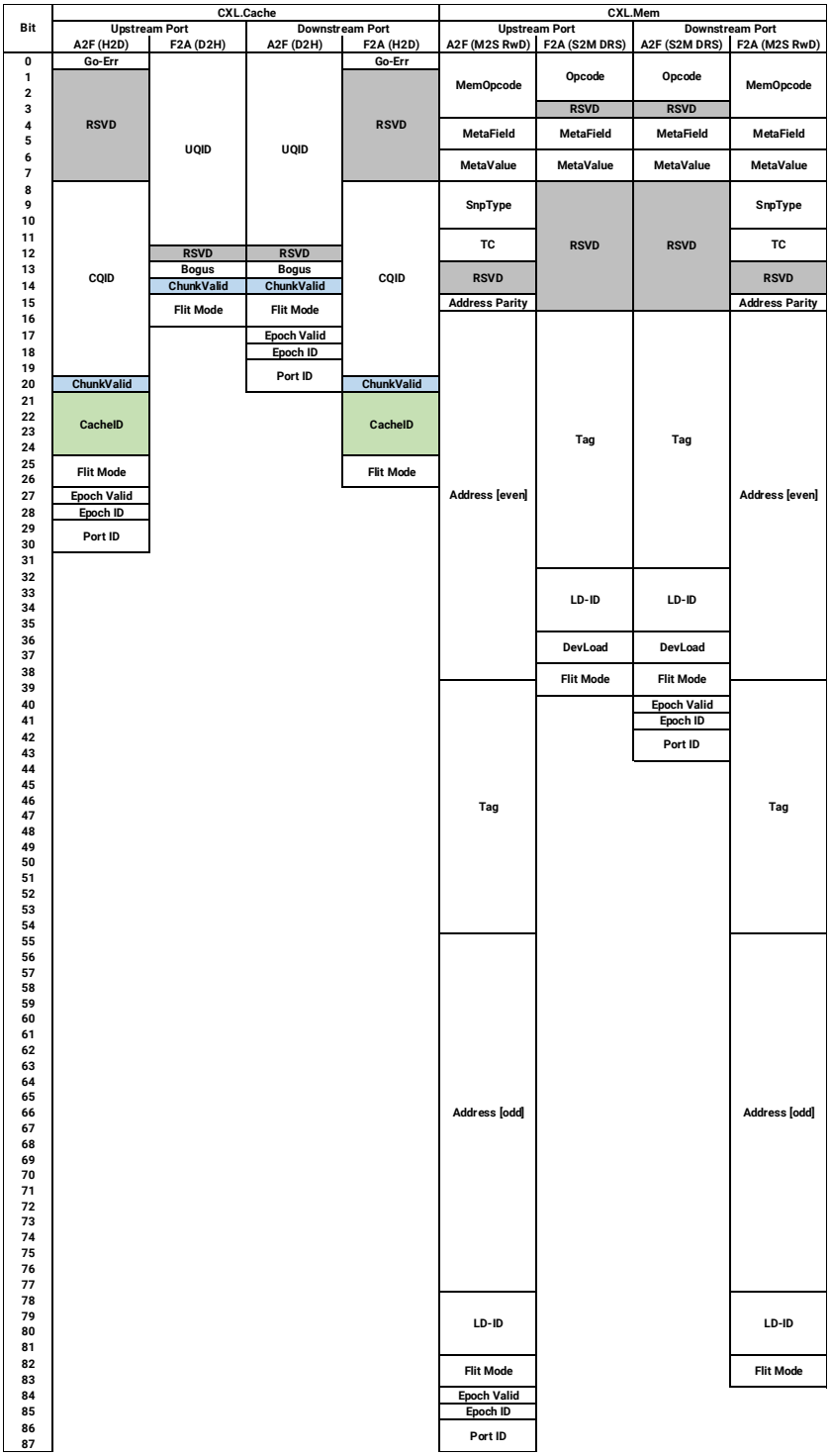
Agent-to-Fabric DATA (A2F)		Fabric-to-Agent DATA (F2A)	
Field	**Position**	**Field**	**Position**
Opcode	[2:0]	MemOpcode	[3:0]
Reserved	[3]	MetaField[1:0]	[5:4]
MetaField[1:0]	[5:4]	MetaValue[1:0]	[7:6]
MetaValue[1:0]	[7:6]	SnpType[2:0]	[10:8]
Reserved	[15:8]	TC[1:0]	[12:11]
Tag	[31:16]	Reserved	[14:13]
LD-ID[3:0]	[35:32]	AddressParity	[15]
DevLoad[1:0]	[37:36]	Address[Even Indices]	[38:16]
Flit Mode	[39:38]	Tag	[54:39]
Epoch Valid	[40:40]	Address[Odd Indices]	[77:55]
Epoch ID	[41:41]	LDID[3:0]	[81:78]
Port ID	[NP+42:42]	Flit Mode	[83:82]
Reserved	[H_DAT-1:NP+43]	Reserved	[H_DAT-1:84]

- **`LD-ID`** is only applicable for 68B/256B Flit modes and is reserved for PBR Flit mode
- **`Flit Mode`** indicates the CXL Flit Mode using the following encodings:
  - 2'b00: 68B Flit
  - 2'b01: 256B Flit
  - 2'b10: PBR Flit
  - 2'b11: Reserved

- **`Flit Mode`** is only applicable for the Upstream Port F2A and Downstream Port A2F directions when FM_ENC_D2H_S2M=1 and reserved otherwise.
- **`Flit Mode`** is only applicable for the Upstream Port A2F and Downstream Port F2A directions when FM_ENC_H2D_M2S=1 and reserved otherwise.

4.3.6 Header Formats

Figure 4-4. DATA Channel Header Formats



4.4 RSP Layer

For requests generated by agents or fabrics, receiving agents or fabrics send back responses using this physical channel. This could be completion, snoop responses, and so on.

The RSP Layer signals are symmetric in the A2F and F2A directions, even though protocols like CXL.cache and CXL.mem are not. Thus, the upstream and downstream versions of CXL.cache and CXL.mem map as different protocols.

Implementation only needs to support the relevant subset of the protocols required for functionality. The direction column in the Table 4-13 specifies signal directions from the perspective of a Transmitter (TX) of packets and a Receiver (RX) of packets.

Table 4-13. Fields of RSP Layer (Sheet 1 of 2)

Signal Class	Signal Name	Direction	Width	Description
VALID	rsp_is_valid	TX→RX	1 bit	Valid bit for the **FLOW_C** and **HDR**
	rsp_block	RX→TX	1 bit	Optional Signal: Transient back pressure from RX due to rate mismatch on a clock crossing at the interface boundary
	rsp_early_valid	TX→RX	1 bit	Optional Signal. It is an early valid indication from TX to RX indicating that TX has packets to send. See Section 4.5.1 for rules related to this signal.
FLOW_C	rsp_protocol_id	TX→RX	4 bits	Identifies the protocol: 4'h1000: Upstream Port CXL.cache 4'h1001: Upstream Port CXL.mem 4'h1010: Downstream Port CXL.cache 4'h1011: Downstream Port CXL.memOther encodings are reserved. Optional for agents with a single protocol.
	rsp_vc_id	TX→RX	4 bits	Indicates the virtual channel used for the corresponding packet. It also identifies the flow control used when **rsp_shared_credit** = 0. Currently, for both CXL.cache and CXL.mem, only 1 channel id is supported (encoding 4'h0). Optional for physical channels with only a single VC per protocol.
	rsp_shared_credit	TX→RX	1 bit	Indicates if the HDR uses shared or dedicated credits. Optional if only dedicated credits are used.
HDR	rsp_header	TX→RX	H_RSP bits	Header information
	rsp_cmd_parity	TX→RX	1 bit	Optional signal: XOR of **rsp_header** fields and **rsp_spid** and **rsp_dpid** fields if supported.
	rsp_spid	TX→RX	12 bits	Optional signal: Source PBR ID when operating in PBR Flit mode
	rsp_dpid	TX→RX	12 bits	Optional signal: Source PBR ID when operating in PBR Flit mode

Table 4-13. Fields of RSP Layer (Sheet 2 of 2)

Signal Class	Signal Name	Direction	Width	Description
CREDIT	rsp_rxcrd_valid	RX→TX	1 bit	Indicates a valid dedicated credit return.
	rsp_rxcrd_protocol_id	RX→TX	4 bits	Identifies which protocol the credits are returned to. 4'h1000: Upstream Port CXL.cache 4'h1001: Upstream Port CXL.mem 4'h1010: Downstream Port CXL.cache 4'h1011: Downstream Port CXL.mem Other encodings are reserved. Optional for agents with a single protocol.
	rsp_rxcrd_vc_id	RX→TX	4 bits	Virtual channel for which credit is returned. Optional for physical channels with only a single VC per protocol.
	rsp_rxcrd_shared	RX→TX	1 bit	Shared credit return. Optional if only dedicated credits are used.
	rsp_txblock_crd_flow	TX→RX	1 bit	Optional Signal: TX requesting RX to block the credit returns due to transient back pressure. An example is a clock crossing FIFO.

The HDR size would be variable and is based on the protocol that is being transported over the CPI interface. When multiple protocols are carried over the CPI interface, the HDR width is sized for the maximum size of the HDR being transported over the CPI interface. The reserved field width is used to cover the unused portion of the HDR. The transmitter must drive 0 on the reserved field and the receiver must ignore this field.

When supported, **`rsp_cmd_parity`** provides parity protection over **`rsp_header`** and **`rsp_spid/rsp_dpid`** from the HDR class if present. RSP HDR parity errors are fatal to interface operation and must be reported as fatal errors. When not supported, it must be reserved; TX must drive 0, and RX must not check RSP HDR parity.

4.4.1 Mapping CXL.cache to HDR

The widths of the different fields are given as per *CXL Specification*. For upstream port, A2F maps to H2D Response, and F2A maps to D2H Response. For downstream port, A2F maps to D2H Response and F2A maps to H2D Response. Currently only a single virtual channel is supported on these channels for CXL.cache.

Table 4-14. Mapping CXL.cache to rsp_header for an Upstream Port (Sheet 1 of 2)

Agent-to-Fabric RSP (A2F)		Fabric-to-Agent RSP (F2A)	
Field	**Position**	**Field**	**Position**
Opcode	[3:0]	Opcode	[4:0]
CQID[11:0]	[15:4]	Reserved	[6:5]
RSP_PRE	[17:16]	UQID[11:0]	[18:7]
Reserved	[18]	Flit Mode	[20:19]
RspData	[30:19]	Reserved	[H_RSP-1:21]
CacheID	[34:31]	--	--
Flit Mode	[36:35]	--	--

Table 4-14. Mapping CXL.cache to rsp_header for an Upstream Port (Sheet 2 of 2)

Agent-to-Fabric RSP (A2F)		Fabric-to-Agent RSP (F2A)	
Field	**Position**	**Field**	**Position**
Epoch Valid	[37:37]	--	--
Epoch ID	[38:38]	--	--
Port ID	[NP+39:39]	--	--
Reserved	[H_RSP-1:NP+40]	--	--

Table 4-15. Mapping CXL.cache to rsp_header for a Downstream Port

Agent-to-Fabric RSP (A2F)		Fabric-to-Agent RSP (F2A)	
Field	**Position**	**Field**	**Position**
Opcode	[4:0]	Opcode	[3:0]
Reserved	[6:5]	CQID[11:0]	[15:4]
UQID[11:0]	[18:7]	RSP_PRE	[17:16]
Flit Mode	[20:19]	Reserved	[18]
Epoch Valid	[21:21]	RspData	[30:19]
--Epoch ID	--[22:22]	CacheID	[34:31]
--Port ID	[NP+23:23]	Flit Mode	[36:35]
--Reserved	[H_RSP-1:NP+24]	Reserved	[H_RSP-1:37]

- **`CacheID`** is only applicable for 256B Flit mode and is reserved for all other modes.
- **`Flit Mode`** indicates the CXL Flit Mode using the following encodings:
  - 2'b00: 68B Flit
  - 2'b01: 256B Flit
  - 2'b10: PBR Flit
  - 2'b11: Reserved
- **`Flit Mode`** is only applicable for the Upstream Port F2A and Downstream Port A2F directions when FM_ENC_D2H_S2M=1 and reserved otherwise.
- **`Flit Mode`** is only applicable for the Upstream Port A2F and Downstream Port F2A directions when FM_ENC_H2D_M2S=1 and reserved otherwise.

4.4.2 Mapping CXL.mem to HDR

The widths of the different fields are given as per *CXL Specification*. For upstream port, F2A maps to S2M NDR, and A2F maps to M2S BIRsp. For downstream port, A2F maps to S2M NDR, and F2A maps to M2S BIRsp. Currently only a single virtual channel is supported on these channels for CXL.mem.

Table 4-16. Mapping CXL.mem to rsp_header for an Upstream Port

Agent-to-Fabric RSP (A2F)		Fabric-to-Agent RSP (F2A)	
Field	**Position**	**Field**	**Position**
Opcode	[3:0]	Opcode	[2:0]
BI-ID	[15:4]	MetaField	[4:3]
BITag	[27:16]	MetaValue	[6:5]
LowAddr	[29:28]	Tag	[22:7]
Flit Mode	[31:30]	LDID[3:0]	[26:23]
Epoch Valid	[32:32]	DevLoad[1:0]	[28:27]
Epoch ID	[33:33]	Flit Mode	[30:29]
Port ID	[NP+34:34]	Reserved	[H_RSP-1:31]
Reserved	[H_RSP-1:NP+35]	--	--

Table 4-17. Mapping CXL.mem to rsp_header for a Downstream Port

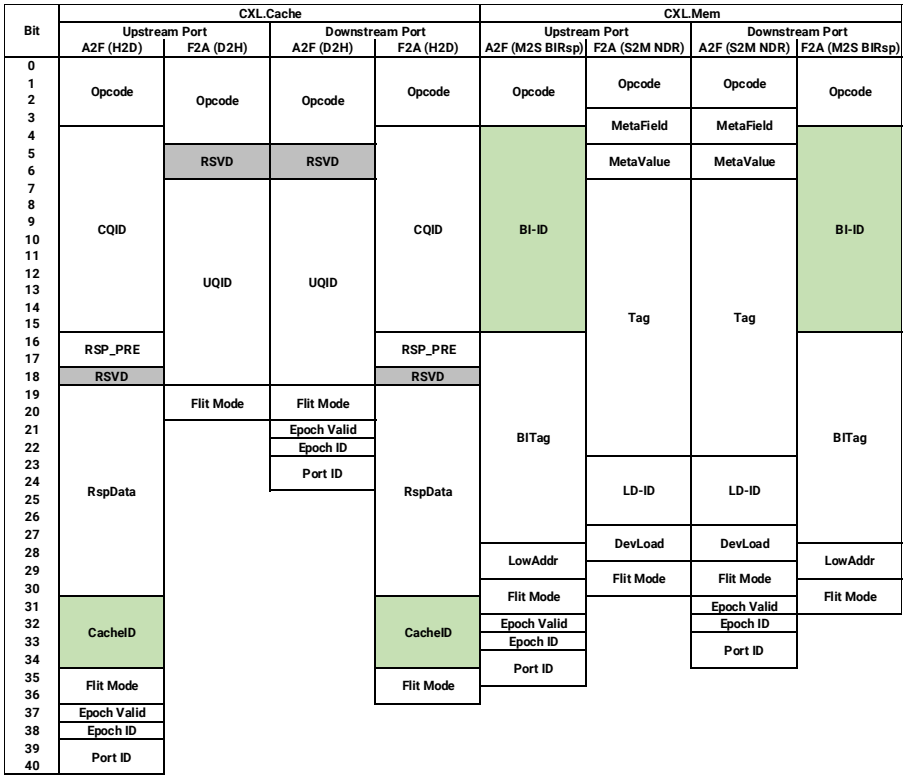
Agent-to-Fabric RSP (A2F)		Fabric-to-Agent RSP (F2A)	
Field	**Position**	**Field**	**Position**
Opcode	[2:0]	Opcode	[3:0]
MetaField	[4:3]	BI-ID	[15:4]
MetaValue	[6:5]	BITag	[27:16]
Tag	[22:7]	LowAddr	[29:28]
LDID[3:0]	[26:23]	Flit Mode	[31:30]
DevLoad[1:0]	[28:27]	Reserved	[H_RSP-1:32]
Flit Mode	[30:29]	--	--
Epoch Valid	[31:31]	--	--
Epoch ID	[32:32]		
Port ID	[NP+33:33]		
Reserved	[H_RSP-1:NP+34]		

- **`BI-ID`** is only applicable for 256B Flit mode and is reserved for all other modes.
- **`LD-ID`** is only applicable for 68B/256B Flit modes and is reserved for PBR Flit mode
- **`Flit Mode`** indicates the CXL Flit Mode using the following encodings:
  - 2'b00: 68B Flit
  - 2'b01: 256B Flit
  - 2'b10: PBR Flit
  - 2'b11: Reserved

- **`Flit Mode`** is only applicable for the Upstream Port F2A and Downstream Port A2F directions when FM_ENC_D2H_S2M=1 and reserved otherwise.
- **`Flit Mode`** is only applicable for the Upstream Port A2F and Downstream Port F2A directions when FM_ENC_H2D_M2S=1 and reserved otherwise.

4.4.3 Header Formats

Figure 4-5. RSP Channel Header Formats



4.5 Clocks and Resets

Agents and fabric can be reset independently. The initialization flow ensures a synchronization handshake to make sure both TX and RX are ready before packet transfers begin. For details on the initialization flow, see Section 5.

The interface is synchronous, but it allows the placement of clock crossing FIFOs at the receivers. Optional blocking signals (**`*_block`** and **`*_txblock_crd_flow`**) are provided for transient back pressure due to clock crossing FIFOs. It is expected that the block signal is configurable to meet timing requires at the TX to allow one to three clocks to stop injecting messages. If clock crossings or dynamic clock gating is not required, these signals are permitted to be tied off to 0.

Initialization signals support clock differences through simple synchronizers and make no assumptions about clock ratios.

4.5.1 Dynamic Clock Gating

If dynamic clock gating is desired, implementations must support all of the *block and *early_valid signals.

Rules for ***_early_valid**:

- TX must assert ***_early_valid** at least one cycle before the corresponding ***_valid**. TX is permitted to keep ***_early_valid** asserted even if it has no immediate packet(s) to transfer, if it wants to keep RX clocks running to achieve the best latency for imminent packet transfer(s).
- TX must assert ***_early_valid** if it has packet(s) to send, even if the corresponding ***_block** signal is asserted. If ***_block** is asserted, TX must keep ***_early_valid** asserted until ***_block** deasserts and TX has transferred the packet(s). It is possible that ***_block** asserts and deasserts several times while packets are being transferred (for example, in cases of clock crossing related rate mismatches); TX must follow the rules related to ***_block** at all times.

From CPI perspective, RX logic is permitted to transition to a clock gated state if the corresponding interface is in Disconnect state OR all of the following conditions are true:

- Interface is in Connected state, with no pending request from TX to Disconnect that RX has not responded to.
- ***_early_valid** is deasserted from TX on all channels.
- No packet transfers are in flight.
- RX has returned the minimum header and data credits required for a packet transfer on any of the channels (for all of the supported protocols and packet types). The minimum credits are defined by the underlying protocol. Implementations are encouraged to return all credits before moving to a clock gated state.

RX must take into account any internal requirements for clock gating in addition to the CPI requirements before transitioning to clock gated state. These internal requirements are implementation specific and outside the scope of this specification.

Once the requirements are met, RX transitions to a clock gated state *N* cycles after asserting the ***_block** signals on all channels (if no packet transfers are received during that time. If packet transfers are received during this time, RX must abort the transition to clock gated state and it is permitted to re-attempt at a later point if all the requirements for entry are met.) *N* is determined by the **Blocking* parameter (Agent or Fabric specific)

Once in a clock gated state, RX must treat any transition on any ***_early_valid**, or **txcon_req** as an asynchronous trigger to exit the clock gated state. Once it has exited clock gated state, it must revert to the regular protocol of accepting packets from TX.

From CPI perspective, TX logic is permitted to transition to a clock gated state if the corresponding interface is in Disconnect state OR all of the following conditions are true:

- Interface is in Connected state, with no pending request from TX to Disconnect that RX has not responded to.
- TX has no packet transfers pending

- TX has accumulated the minimum header and data credits required for a packet transfer on any of the supported channels (for all of the supported protocols and packet types). The minimum credits are defined by the underlying protocol.

TX must take into account any internal requirements for clock gating in addition to the CPI requirements before transitioning to clock gated state. These internal requirements are implementation specific and outside the scope of this specification.

Once the requirements are met, TX transitions to a clock gated state *N* cycles after asserting the ***block_crd_flow** signals on all channels. *N* is determined by the **Blocking* parameter (Agent or Fabric specific)

Exiting from a clock gated state for a TX are implementation specific and outside the scope of this specification.

Global clock gating must take into account the CPI clock gating conditions for all the relevant TX and RX logic associated with the design, as well any other internal or SoC level requirements.

4.6 Common to REQ, DATA, and RSP

4.6.1 Channel Flow Control

CPI channels use link-layer credits for flow control, where flow control is separate for each of the three channels (REQ, DATA, and RSP). Each virtual channel (VC ID) must use a credit to send each message and collect credit returns from the receiver. Each credit represents a message. Therefore, only a single credit is used to send a 64B payload on the DATA physical channel.

Each physical channel has dedicated credit return wires for dedicated credits (VC ID-based) and shared credits (shared across virtual channels and protocols). During operation, the receiver returns a credit whenever it has processed the message (that is, freeing the receiver buffer). If there are multiple physical channels of the same type, each includes this credit interface.

To avoid deadlock, a receiver must ensure fairness in returning each dedicated credit type and between dedicated and shared credits.

When transmitting, it is recommended to use shared credits first and only use dedicated credits (per VC credits) after shared credits have been fully consumed.

4.6.2 Credit Return

Each physical channel includes a credit return interface from the RX. In this section, we use the term *CHAN* to abstract the particular physical channel (REQ, DATA, RSP). When the ***CHAN_rxcrd_shared** signal is asserted, it indicates that a shared credit is being returned. ***CHAN_rxcrd_valid** asserted indicates that a dedicated credit is being returned. Shared credits and dedicated credits can be returned in parallel.

4.6.2.1 Requirements

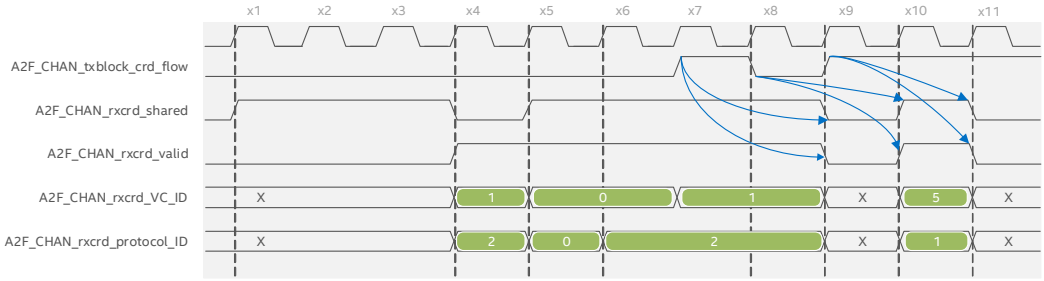
1. The TX side of a link must include an 8-bit credit counter for each supported credit type. This requirement applies to shared and dedicated credits.
2. The RX side of a link that returns credits must not return more credits than can fit into the 8-bit counter (255 maximum).

4.6.2.2 Flow

Figure 4-6 captures a credit return example, and the steps during the flow are:

1. X1 to X3 shared credits are being returned as indicated by the assertion of ***CHAN_rxcred_shared**.
2. X4 only dedicated credits are being returned as indicated by the assertion of ***CHAN_rxcrd_valid**.
3. X5 to X8 both shared and dedicated credits are being returned with both ***CHAN_rxcrd_shared** and ***CHAN_rxcrd_valid** being asserted.
4. X7 the ***CHAN_txblock_crd_flow** is asserted, which requires that credits stop being returned in 2 cycles (the value of 2 is a configurable parameter assumed in this example flow). The result is credit return stops in cycle N+8 as indicated by deassertion of both ***rxcrd_shared** and ***rxcrd_valid**.
5. X8 the **CHAN_txblock_crd_flow** is deasserted to allow credit returns to continue in X10.
6. X9 block is asserted for the second time and results in credit return de-assertion in X11.

Figure 4-6. Channel Credit Return Flow



4.6.3 Virtual Channels, Virtual Networks, and QoS

Virtual Channels (VCs) and Virtual Networks (VNs) for CPI present the following considerations:

- VC and VN both separate messages into further flow control classes beyond the baseline channel definition.
- The baseline channel flow control provides a non-blocking flow control for each class of traffic within each protocol.

In the case of CXL.cache, currently there is only 1 virtual channel per physical channel per direction.

Future revisions of this specification might include additional fields to assist with Quality-of-Service metrics and/or enforcement.

4.6.4 Credit Combining Allowances

In agents, it is acceptable to advertise only shared credits for **VC_IDs**, which are guaranteed to sink without dependence (including network layer dependence). This can be done to avoid the need for dedicated, per **VC_ID** credits. The RSP channels for protocol agents are an example where this is possible if there is a pre-allocated tracking structure that can absorb the response.

4.6.5 Error Handling

It is recommended for the CPI transmitter to filter malformed requests as defined by the protocol being transmitted, and for the CPI receiver to handle unsupported requests with appropriate error logging and reporting. The receiver must be able to detect and handle unsupported packets without compromising data or system integrity. Error logging and escalation is implementation specific.

4.6.5.1 Flow Control Error Handling

Error handling for illegal flow-control cases results in undefined behavior. It is recommended to have the agents and the fabric check for illegal cases that trigger assertions in Register-Transfer Level (RTL) and log/signal fatal errors to allow for post-Si debug.

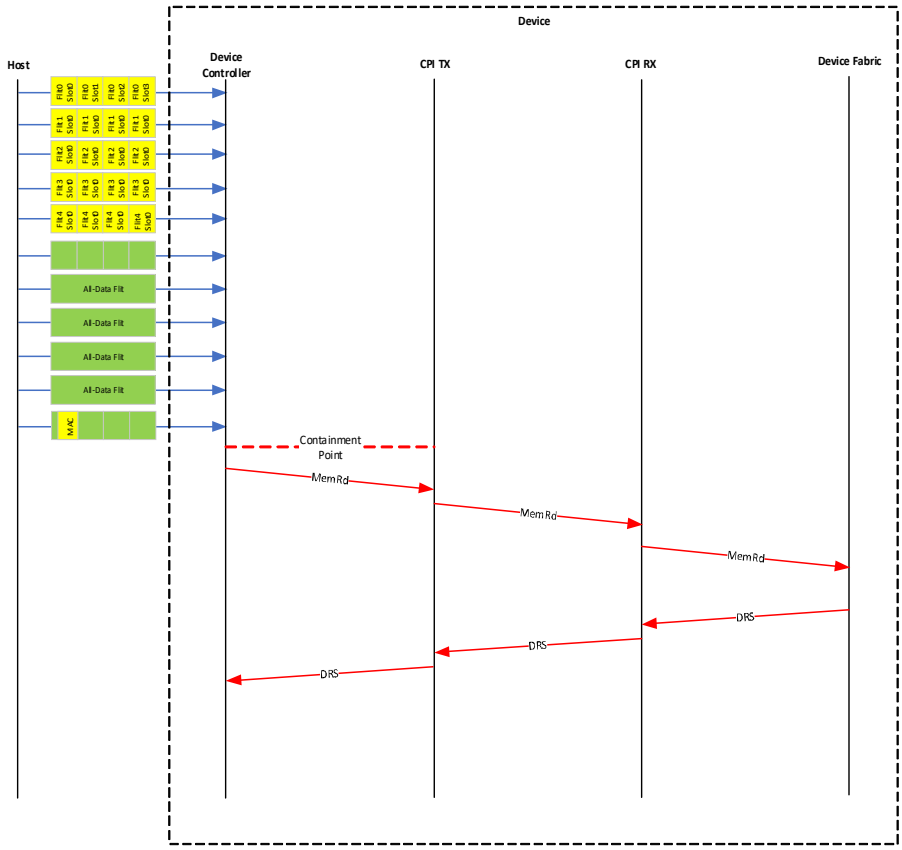
Example error conditions (not exhaustive):

- EOP signal is asserted when the packet is not completed (2-FLIT message encoding, but the EOP is set on first FLIT).
- Receive queue overflow.
- Clock Crossing FIFO overflow condition.
- Invalid/reserved protocol ID encoding
- Unsupported VC ID encoding

4.6.6 Containment Mode Latency Reduction

When a CXL link is operating in IDE Containment Mode, CXL receivers must hold on to any received protocol packets until the corresponding MAC arrives and is verified for integrity before releasing for further processing. For latency-sensitive flows such as memory reads, this consumes valuable time that could be used to pre-process the request. Figure 4-7 demonstrates an example Host to Device memory read that is held up in the device controller until the MAC epoch completes, the MAC is transmitted and is checked for integrity (Containment Point).

Figure 4-7. Example CXL.Mem M2S Read Request with Containment



This section describes a mechanism for the controller and fabric to communicate epoch integrity information over CPI to allow for pre-processing of protocol packets ahead of the containment point to reduce latencies. When this mode is enabled, the A2F header formats for all layers are augmented to include three new fields:

1. Epoch Valid: Single-bit indication that the associated Epoch ID/Port ID fields are valid
2. Epoch ID: Single-bit identifier to associate a protocol request with the epoch that sourced it over the link
3. Port ID: Variable-width field to indicate port ID that sourced the request

If this mode is not enabled, these fields are reserved and must be driven 0 by the transmitter and ignored by the receiver.

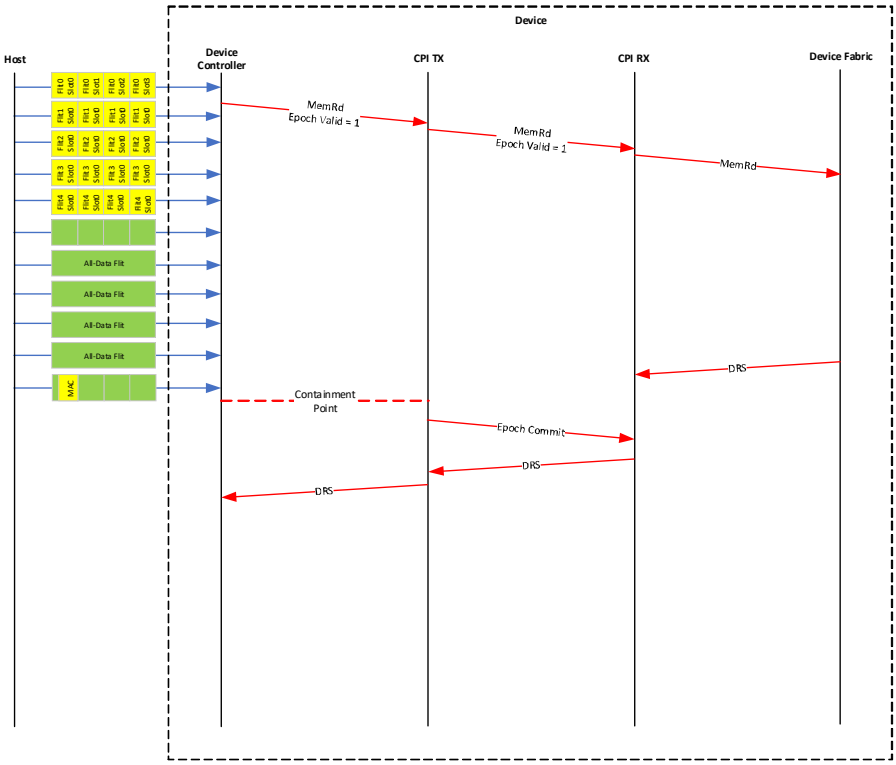
A single bit of Epoch ID is sufficient for the optimized use case due to the pipelined and ordered nature of epochs and their associated MAC transmission. In the optimized use case, the MAC for a given epoch will be delivered through the subsequent epoch, freeing up the ID of the first epoch, and so on. For the non-optimized use case where more than two MAC epochs may be in flight at the same time, the controller must not expose any packets from the additional epoch over CPI and instead wait for containment before exposing any packets from that epoch. The receiving fabric must concatenate the Port ID and Epoch ID to form a unique identifier for tracking.

These fields carry sufficient information for the CPI Receiver to begin pre-processing the request while waiting for MAC authentication. For example, it may prefetch data that is being requested. The Receiver must ensure that any requested data is not visible and no updates are committed until MAC authentication is complete. The fabric may choose to achieve this by using the CXL.mem defined MemSpecRd flow to start the memory access without directly exposing the standard Read flow to all areas of the on-die fabric.

Three new signals added to the CPI Global Layer A2F wires allow a controller to provide the MAC integrity check status to the fabric. The width of these three signals is dependent on the number of instantiated ports, with each port mapping to one of the indices with consistent mapping for all three fields. For example, for P=4 ports present, **epoch_id[0]** / **epoch_commit[0]** / **epoch_reject[0]** carry status from port0, **epoch_id[1]** / **epoch_commit[1]** / **epoch_reject[1]** carry status from port1, and so on. For each present port, once the MAC for an epoch has been received and checked for integrity, the status of the check can be communicated to the fabric by asserting either the **epoch_commit** for a successful check or the **epoch_reject** for an unsuccessful check along with the corresponding **epoch_id** for the correct port-based bit position. The CPI receiver must concatenate the port ID (by detecting which bit is being asserted) and the **epoch_id** to update the authentication status and complete processing for any previously received requests. The CPI receiver must wait for the MAC authentication status before releasing any data when in Containment Mode. Receiver behavior for the unsuccessful MAC authentication case is implementation specific, but the receiver must not expose or update any system state. If this mode is disabled for any port, the CPI TX can communicate this by asserting both **epoch_commit** and **epoch_reject** concurrently on the relevant port bits.

Figure 4-8 demonstrates a similar example Host to Device memory read that utilizes this change to pre-process the read in parallel to the authentication check. In this case, the controller immediately forwards the protocol request to the device fabric to begin processing, instead of waiting for the Containment Point. This request to the fabric is tagged with an Epoch Valid = 1 and an Epoch ID = 0. While the device is pre-processing the read, MAC arrives at the receiver and is verified, at which point the controller signals to the fabric that MAC authentication was successful. At this point, the fabric can complete the request and release the data.

Figure 4-8. Example CXL.Mem M2S Read Request with Containment Optimization



5 Connect and Disconnect Flows

This section describes the connect and the disconnect flows for CPI. Flows are invoked during boot/reset and when going into a low power mode.

Connect Flow

CPI defines an initialization phase where information about credit availability in the Receiver (RX) is communicated to the Transmitter (TX) after a connection is established. It is permitted for Reset to independently deassert between the agent and fabric sides of CPI. For independent reset, the initialization signals are driven to the disconnected condition when in reset and no traffic is sent until initialization reaches the connected state.

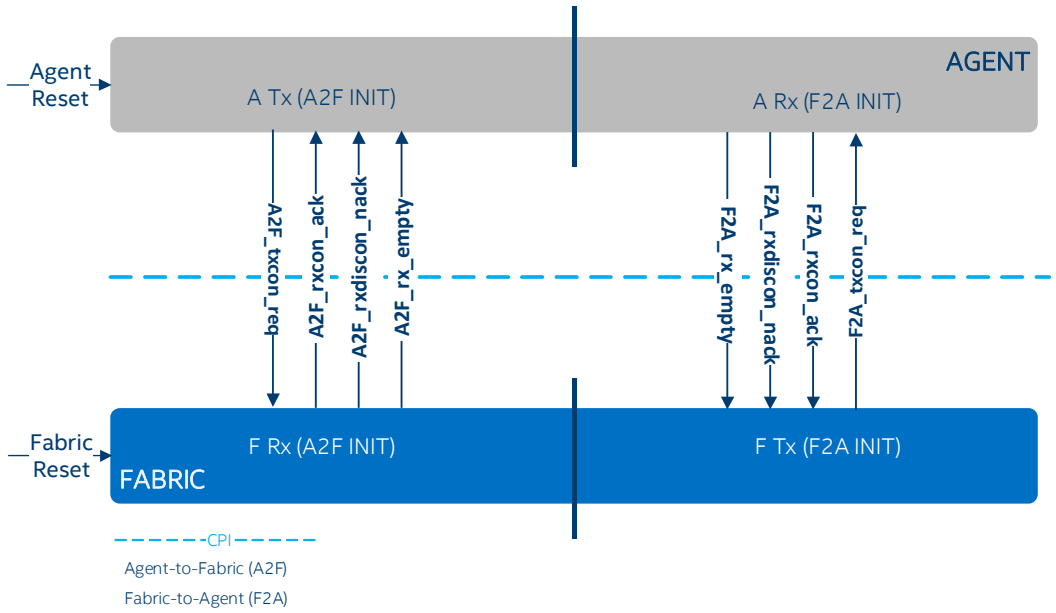
Disconnect Flow

The disconnect flow is optionally supported by agents with the following primary usage models: reconfiguring credits and power saving. Without this flow, all CPI credits must be configured to a final value before the first connection can proceed. It is recommended that the Disconnect flow is coordinated through higher level SoC or firmware involvement in order to facilitate optimal usage and avoid race conditions. As an example, one implementation could involve a SoC aggregator collect sideband idle indications from TX and RX before triggering TX to initiate the disconnect flow for power gating.

CPI connections are separated in the A2F and F2A direction. The signaling for connection is in the Init Global physical channel of CPI. See Section 4.1 for the signal list. The init wires are shown in Figure 5-1.

For interoperability, receivers that do not support the disconnect flow must respond to a disconnect request by asserting **`rxdiscon_nack`** until the transmitter re-asserts **`txcon_req`**.

Figure 5-1. Init Wires



Although not required, the agent and fabric sides of CPI are typically brought out of reset close to or at the same time. One end of CPI (after coming out of reset) does not have implicit requirements for when the other end should come out of reset. An explicit handshake mechanism during initialization ensures that both endpoints (and all pipeline stages between them) are out of reset before any credits or transactions being sent on CPI.

After connection, RX sends credits for dedicated VC buffers and shared buffers. The CPI TX supports a runtime asserted **`block`** signal for credit returns.

5.1 Initialization States

Initialization states are defined based on the following three wires:

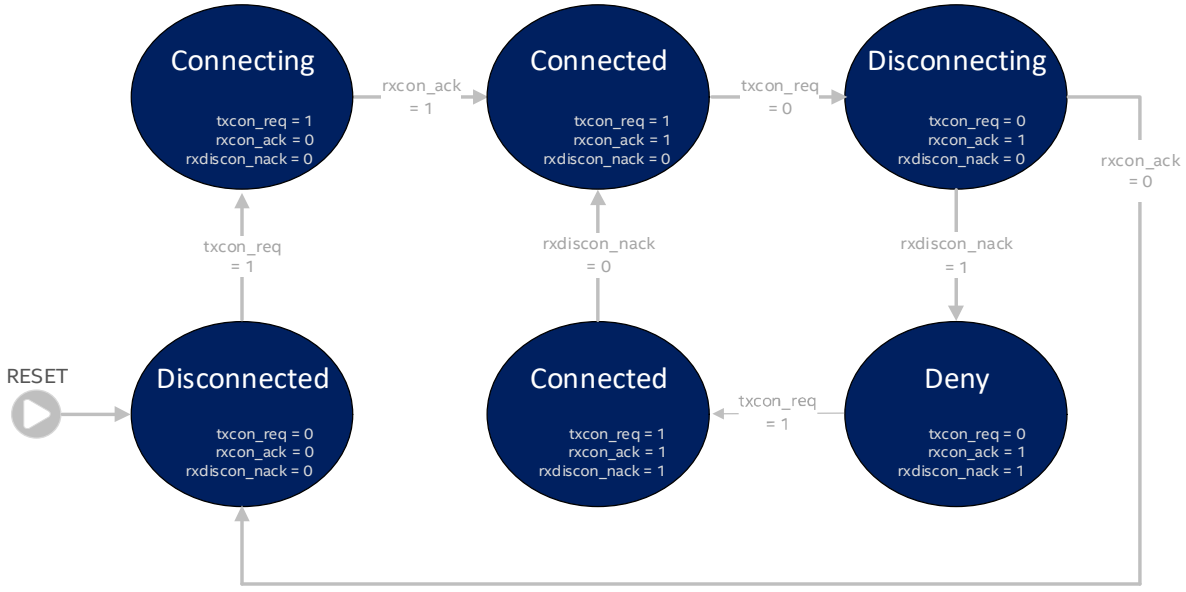
- txcon_req
- rxcon_ack
- rxdiscon_nack

The state is used to determine actions necessary on the RX and TX of CPI as defined in Table 5-1. The basic state transitions are shown in Figure 5-2 and detailed rules for connect and disconnect flows are captured in subsequent sections of this chapter.

Table 5-1. Init State Actions

txcon_req	rxcon_ack	rxdiscon_nack	State	TX Actions	RX Actions
1	0	0	Connection request (Connecting)	Sink Credits Do NOT Send Packets	Do NOT Send Credits Do Not Sink Packets
1	1	0/1	Connected	Sink Credits Send Packets	Send Credits Sink Packets
0	1	0	Disconnection request (Disconnecting)	Sink Credits Do NOT Send Packets	
0	1	1	Deny (Disconnect Rejected) and must go back to be connected		
0	0	0	Disconnected	Drop Credits Do NOT Send Packets	Do NOT Send Credits Do NOT Sink Packets
1/0	0	1	Illegal States	n/a	n/a

Figure 5-2. Initialization State Machine



5.2 Signaling Rules

1. The **txcon_req** signal use:

   0 to 1 → Connection request

   1 to 0 → Disconnection request

2. Credit return:

   a. **crd_valid** = 1 means it is releasing the dedicated message credits for a protocol ID and a virtual channel ID.

   b. **crd_shared** = 1 means it is releasing a shared credit. This can happen in parallel with a dedicated message credit return.

c. Credit return behaves in the same way during the first initialization of credits as it does during runtime return of credits.

The **rx_empty** signal indicates all channel credits returned from the RX and all RX Queues are empty. This does not account for messages that are in flight or in intermediate buffers such as clock crossing queues.

a. During initialization, if Disconnect is not supported, TX is permitted to send messages as soon as any credits are available and not depend on rx_empty assertion. if Disconnect is supported and *TX_CRD_REG*=0, TX must stall the sending of any packets after init until **rx_empty** is asserted, it must use the credits received as an indication of the total credits a receiver has advertised.

Note: 3 (a) implies a timing relationship between **rx_empty** and the credit return signals. Specifically, the assumption is that implementations guarantee **rx_empty** cannot race ahead of credit return signals. This is trivial in designs that do not support clock crossings or dynamic clock gating. When clock crossings or dynamic clock gating is supported (that is, scenarios where block_crd_flow can assert), it is required that TX implementations expose configuration registers that can be programmed and/or reset to indicate the maximum credits RX can advertise (this avoids several race conditions related to in flight credit returns, idle determination and disconnect flow). *TX_CRD_REG* is the parameter that indicates support for this. These configuration registers (also referred to as credit registers) are programmed through implementation specific sideband mechanisms. If credit reconfiguration post reset is not required, it is permitted to set the reset defaults of these registers to be the maximum credits RX will advertise and no further programming is required. However, if relying on software/ firmware programming, TX must expose a ProgrammingDone bit with default value 0, and wait for software/firmware to write 1 to it before consuming the values in the credit registers.

3. TX is permitted to send packets when it receives one credit on a given physical channel.
   a. Sending also depends on having the correct credit, where shared credit can be used by any message and dedicated credit can only be used by messages of a single VC and protocol combination.
4. RX stalls the credit release N cycles after **A_CHAN_txblock_crd_flow** is asserted.
   a. The *AgentBlocking* parameter defines the value of *N* cycles.
   b. A delay of *N* cycles occurs between **Txblock_crd_flow** state change until **crd_valid** and **crd_shared** signals reflect the corresponding block or unblock.
   c. Clock crossing of credit returns is an example use case where the **Txblock_crd_flow** is asserted if the free entries in the clock crossing FIFO are $\leq$ N.
   d. If a design does not require clock crossing or clock gating related stalls, the **Txblock_crd_flow** signal must be permanently tied to **0**.
5. Connection **Ack** always follows connection **Req**.
   a. Req is signaled by **Txcon_req** transitioning from 0 $\rightarrow$ 1. This transition is indication the TX is ready to receive credits and is in normal operation.
   b. Ack is signaled with **Rxcon_ack** transitioning from 0 $\rightarrow$ 1. Ack might be stalled until RX is ready to complete.
6. Disconnect Ack or NAck follows Disconnect Req.
   a. Disconnect Req is signaled by a **Txcon_req** transition from 1 $\rightarrow$ 0.
   b. Disconnect Ack is signaled by an **Rxcon_ack** transition from 1 $\rightarrow$ 0.
   c. Disconnect NAck is signaled by an **Rxdiscon_nack** transition from 0 $\rightarrow$ 1.

d. RX must select Ack versus NAck for each **disconnect** request. The response time is implementation specific, but must not cause system level timeouts and so on.

5.3 Reset and Connect Flow

Figure 5-3 shows the A2F Init flow and the accompanying text focuses only on the A2F init description. A mirror of this flow with opposite drivers for the signals exists in the F2A direction. An interface is considered IDLE if either of the following conditions is true:

1. Interface is in Disconnect state with no outstanding connection request from TX. Credit avail counters of TX are 0. This is the same as Reset state.
2. Interface is in Connected state, and there are no transactions outstanding from TX. Credit avail counters would have the maximum advertised credits from RX.

Steps for connection:

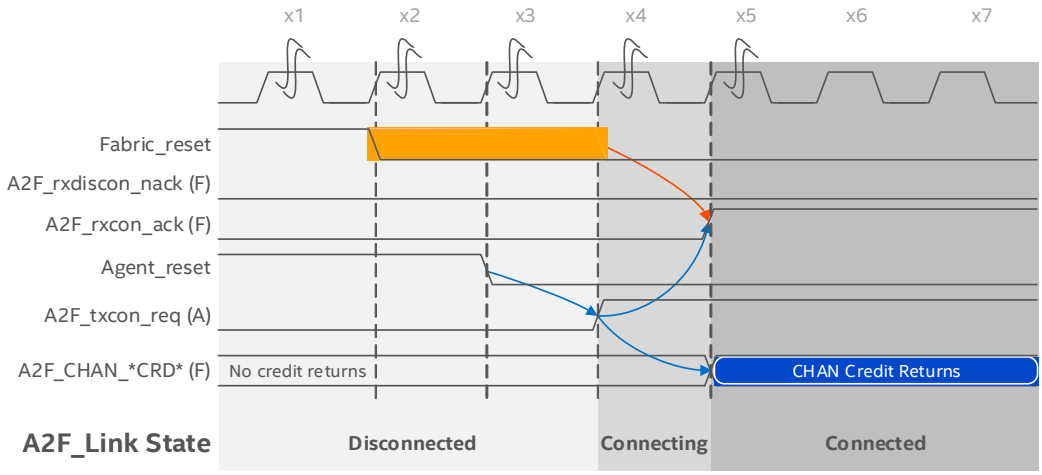
1. After TX is out of reset, it asserts **txcon_req** to RX.
2. After RX is out of reset, it waits for **txcon_req** from TX.
3. After RX receives **txcon_req**, it asserts **rxcon_ack** to TX. **rxcon_ack** must assert at least one cycle after **txcon_req** asserts.
4. After RX asserts **rxcon_ack** to TX, it is permitted to immediately start returning credits.
5. After the minimum credits are received to send packets, TX is permitted to start sending packets (provided conditions outlined in rule 3(a) in Section 5.2 are met).

In Figure 5-3 and Figure 5-4, **F** after the signal name represents the fabric as the driver of the signal and **A** denotes the agent as the driver of the signal. Table 5-2 lists each major time marker.

Table 5-2. Reset Time Marker Description

Time Marker	Description
X1	Agent and fabric are in reset.
X2	**Fabric_reset** might deassert.
X3	**Agent_reset** deasserts, allowing the agent to start initialization.
X4	Agent asserts **A2F_txcon_req**. This can be an arbitrary number of cycles after the **Agent_reset** asserts. Until the connection is complete, this signal must remain asserted and can only deassert as part of the disconnect flow. After asserting **A2F_txcon_req**, the agent must be able to accept credit returns because credit returns might be observed before observation of **A2F_rxcon_ack** due to intermediate buffering or clock crossings.
X5	**A2F_rxcon_ack** asserts after both **Fabric_reset** and **A2F_txcon_req** are asserted. A fixed number of cycles are not required between X4 and X5. **A2F_rxcon_ack** can only deassert as part of the disconnect flow.
X5	A2F can start to return credits simultaneously with the assertion of **A2F_rxcon_ack**. There are no strict expectations on flight time or observation of when A observes the Ack relative to the credits.

Figure 5-3. Reset to Connected Timing Diagram



5.4 Disconnect and Reconnect flow

As mentioned previously, it is recommended that implementations coordinate idle conditions through higher level SoC or firmware assistance for triggering a disconnect flow. This helps simplify hardware implementations. In addition, TX must make sure it has received all the expected credits from RX and have no transactions in flight before initiating a Disconnect flow. If *TX_CRD_REG* is 1, TX knows the maximum expected credit returns from RX via the credit registers. If *TX_CRD_REG* is 0, implementations must be able to guarantee the timing relationships between **`rx_empty`** and credit return signals as outlined in Section 5.2 rule 3 (a).

The following steps see Figure 5-4 and reference the time stamps in that diagram.

1. TX deasserts **`Txcon_req`** to disconnect at time x3.
   a. **`rxdiscon_nack`** must be deasserted before **`txcon_req`** de-assertion.
   b. TX must not be sending messages on any channel (indicated by **`CHAN_is_valid`** bit assertion).
2. RX must decide to acknowledge (Ack) or negatively acknowledge (NAck or reject) the disconnect.
   a. To Ack, RX deasserts **`rxcon_ack`** after ensuring all pipelines are empty at time x4 in Figure 5-4, which marks the entry into a disconnected state. Optionally, it can also ensure that all credits have been returned.
   b. To NAck, the RX asserts the **`rxdiscon_nack`** at time x4 in Figure 5-5. It might choose to do this if it is unable to drain its pipelines without risking deadlock.
      i. After NAck, the **`txcon_req`** must re-assert as shown at time x5 in Figure 5-5.
      ii. After that is observed, the **`rsdixcon_nack`** can deassert at time x6 in Figure 5-5.
3. The Reconnect flow is the same as the Reset flow with the following exceptions:
   a. To start a new credit init, RX must reset its credit counters to reset values.
   b. TX resets its **`credit_avail`** counters to zero. If *TX_CRD_REG* = 1, TX must reset the credit registers and if applicable, the ProgrammingDone bit must be reset to 0 as well.

Note: The connect and disconnect flows are expected to complete within a few microseconds after initiation, but no timeout is explicitly defined. To meet this expectation for disconnect, the RX should reply Ack or NAck within this time window. The agent, fabric, or SoC can define a timeout requirement to ensure this.

Figure 5-4. Connected to Disconnected to Connected Flow

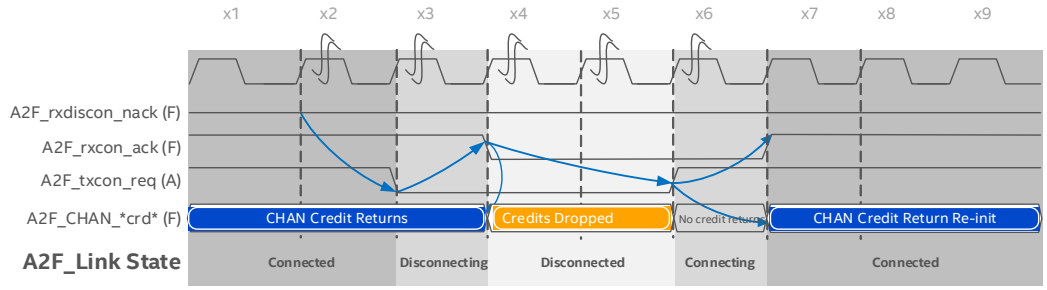
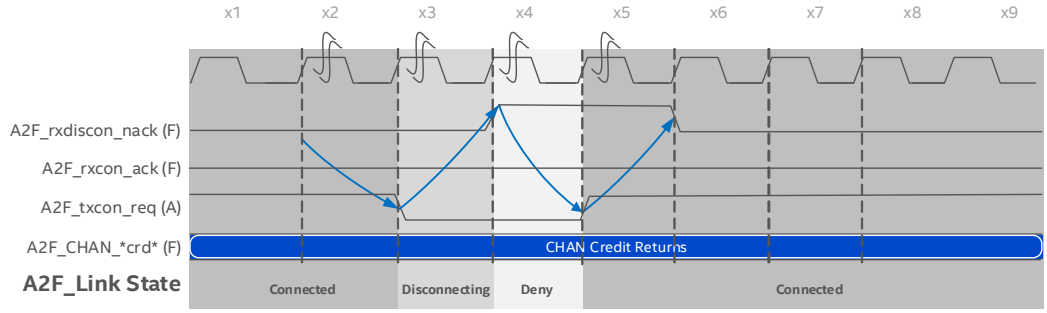


Figure 5-5. Deny for Disconnect (Disconnect NACK)



5.5 Surprise Reset

A surprise reset occurs when the agent or fabric resets while CPI is connected. The recommended flow is to use disconnect before reset. This section captures expected behavior if a surprise reset occurs.

There are two cases to consider for a surprise reset:

1. **`rxcon_ack`** 1 → 0 occurs because of a surprise reset on RX side of the link while TX **`txcon_req`** is 1:
   a. TX forces itself to a disconnected state and restarts initialization. If this happens when TX is in an idle state, it can recover without loss of messages.
2. **`txcon_req`** 1 → 0 occurs because of a surprise reset on the TX side of the link while **`rxcon_ack`** is 1:
   Follow the regular disconnect flow.
   a. If this happens when RX is in an idle state, disconnect should receive Ack and cleanly reach a disconnected state provided TX stays in reset.
   b. If the disconnect is Denied (NAck) by RX, it results in a fatal or illegal link state that does not recover.

For both cases, if traffic is active (that is, not idle), a loss of protocol messages can result and will likely be fatal to continued normal operation.

6 Mode Configuration

For a given SoC use case, CPI expects the configuration of an interface to be static. The exact solution for defining the configuration can be done in different ways and is not mandated by CPI. One way to configure the interface is through static *RTL* parameters.

The protocol supported through the parameters results in changes to the header field size in each physical channel. For details of the header and physical channel widths for different combinations, see Section 4. Additional supported parameters are shown in Table 6-1. Parameters supported are per physical channel, except when a particular channel is specified in the Description column.

Table 6-1. Parameters Supported (Sheet 1 of 2)

Parameter	Description
Prot_UP_CXL_CACHE	If 1, Upstream Port CXL.Cache protocol supported
Prot_DP_CXL_CACHE	If 1, Downstream Port CXL.Cache protocol supported
Prot_UP_CXL_MEM	If 1, Upstream Port CXL.Mem protocol supported
Prot_DP_CXL_MEM	If 1, Downstream Port CXL.Mem protocol supported
A2F_DataHdrSep[2:0]	Defines the separation between header and payload fields on the DATA physical channel. Applies to all protocols on an interface. Modes 0x0 to 0x3 = Fixed 0 to 3 cycle separation, respectively, Modes 0x4 to 0x7 = Reserved and not used. This value is independent for A2F and F2A directions.
F2A_DataHdrSep[2:0]	
AgentBlocking[1:0]	Agents that require support for the **block** signal to stop traffic for the channel or credit return use this parameter to define the block behavior. The reaction time of the **block** signal is dependent on the fabric's reaction time, meaning that time from the **block** asserted until **Valid** is guaranteed to be deasserted. See description of blocking in Section 4.3.2. 0x0 – No Blocking 0x1 – Blocking is enabled with a response time of 1 cycle 0x2 – Blocking is enabled with a response time of 2 cycles 0x3 – Blocking is enabled with a response time of 3 cycles
FabricBlocking[1:0]	Same definition as *AgentBlocking*, but applies to traffic going to the fabric, where fabric can assert a block signal. When enabled, this parameter defines the reaction time of the agent to a fabric block.
ReqCmdParity	If 1, **req_cmd_parity** field is supported on the REQ channel.
DataCmdParity	If 1, **data_cmd_parity** field is supported on the DATA channel.
RspCmdParity	If 1, **rsp_cmd_parity** field is supported on the RSP channel.
ByteEnableParity	If 1, **data_byte_enable_parity** field is supported on the DATA channel.
SHARED_CRD_EN	If 1, shared credits are supported on all credited channels. If 0, only dedicated credits are used on all credited channels.
MEM_DATHDR_SPLIT	If 1, **data_header** is split over multiple cycles for CXL.mem protocol sent on the DATA channel.
TXFER_32B_EN	If 1, 32B transfers are supported and **data_sz** signal must be implemented. If 0, 32B transfers are not supported and **data_sz** signal must not be implemented.
TX_CRD_REG	If 1, TX exposes configuration registers that can be programmed and/or reset to values that indicate the maximum credits advertised by RX for each credit pool.
VIRAL_EN	If 1, the Viral signal is supported on the Global channel
FATAL_EN	If 1, the Fatal signal is supported on the Global channel

Table 6-1. Parameters Supported (Sheet 2 of 2)

Parameter	Description
PBR_FLIT_MODE_EN	If 1, instantiate applicable DPID/SPID signals for REQ/DATA/RSP physical channels. If 0, signals are not instantiated and only 68B/256B Flit Mode packet formats are supported.
FM_ENC_D2H_S2M	If 1, Flit Mode field is present in CXL.Cache D2H and CXL.Mem S2M packets using the appropriate header formats described in this document. If 0, the bits are Reserved.
FM_ENC_H2D_M2S	If 1, Flit Mode field is present in CXL.Cache H2D and CXL.Mem M2S packets using the appropriate header formats described in this document. If 0, the bits are Reserved.
H_REQ	Total width of **`req_header`** in bits
H_DAT	Total width of **`data_header`** in bits
H_RSP	Total width of **`rsp_header`** in bits
IDE_Epoch_Support	If 1, IDE Containment Mode latency reduction mechanism is supported. Epoch signals for GLOBAL physical channel are instantiated, and header formats carry relevant signals. See Section 4.6.6
P	Number of active CXL ports. Used only if IDE_Epoch_Support = 1.
NP	Number of bits needed to encode active CXL ports in binary.