AMBA® CHI Chip-to-Chip (C2C)

Architecture Specification

Document number	IHI0098
Document quality	Release
Document version	A
Document confidentiality	Non-confidential
Date of issue	Feb 2024

AMBA® CHI Chip-to-Chip (C2C) Architecture Specification

Release information

Date	Version	Changes
2024/Feb/07	A	• First public release

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

http://www.arm.com

Contents

AMBA® CHI Chip-to-Chip (C2C) Architecture Specification

Part A
Preface

About this specification

This specification describes the AMBA® CHI *Chip-to-Chip* (C2C) architecture.

Intended audience

This specification is written for engineers who want to become familiar with the C2C architecture compatible with the CHI architecture.

Using this specification

The information in this specification is organized into parts, as described in this section:

Chapter B1 *Introduction*

Read this for an introduction to the CHI C2C architecture and the terminology in this specification.

Chapter B2 *Interface structures*

Read this for an overview on the functional layers between on-chip CHI and CHI C2C.

Chapter B3 *Packetization*

Read this for a description of how messages are packed into containers.

Chapter B4 *Message structures*

Read this for a description of the size and fields in each message type.

Chapter B5 *Flow control*

Read this for a description of the credit exchange used to flow control messages.

Chapter B6 *DVM transactions*

Read this for a description of the DVM transaction format and how they are propagated.

Chapter B7 *RME-DA and RME-CDA support*

Read this for an overview of RME-*Device Assignment* (RME-DA) and RME-*Coherent Device Assignment* (RME-CDA).

Chapter B8 *Interface state management*

Read this for a description of how to manage the connectivity state of the interface.

Chapter B9 *Interface initialization*

Read this for an overview of how to initialize C2C interface.

Chapter B10 *Protocol layer property exchange*

Read this for a description of interface properties and how these properties are communicated.

Conventions

Typographical conventions

The typographical conventions are:

italic	Highlights important notes, introduces special terminology, and denotes internal cross-references and citations.
bold	Denotes signal names, and is used for terms in descriptive lists, where appropriate.
`monospace`	Used for assembler syntax descriptions, pseudocode, and source code examples. Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.
SMALL CAPITALS	Used for a few terms that have specific technical meanings.

Signals

The signal conventions are:

Signal level The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- HIGH for active-HIGH signals.
- LOW for active-LOW signals.

Lowercase n At the start or end of a signal name denotes an active-LOW signal.

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`. Both are written in a `monospace` font.

Additional reading

This section lists publications by Arm and by third parties.

See Arm Developer, http://developer.arm.com for access to Arm documentation.

[1] *AMBA® CHI Architecture Specification*. (ARM IHI 0051 G) Arm Ltd.

[2] *Universal Compute Interconnect Express (UCIe)*. Universal Compute Interconnect Express Consortium.

[3] *Compute Express Link (CXL)*. Compute Express Link Consortium.

[4] *Arm® Realm Management Extension (RME) System Architecture*. (ARM AES 0053 Issue B 00eac5) Arm Ltd.

[5] *Arm® Architecture Reference Manual Supplement, The Realm Management Extension (RME), for Armv9-A*. (ARM DDI 0615 Issue A.d) Arm Ltd.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this specification

If you have any comments or queries about our documentation, create a ticket at https://support.developer.arm.com.

As part of the ticket, please include:

- The title (AMBA® CHI Chip-to-Chip (C2C) Architecture Specification).
- The number (IHI0098 A).
- The section name to which your comments refer.
- The page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Inclusive terminology commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive.

Arm strives to lead the industry and create change.

This specification does not contain such terms. If you find offensive terms in this document, please contact terms@arm.com.

Part B
Specification

Chapter B1
Introduction

This specification is a Chip-to-Chip (C2C) extension of the *AMBA® CHI Architecture Specification* [1]. It contains the following sections:

B1.1 Overview

The C2C extension enables building a system with multiple CPUs, accelerators, or other devices. CHI C2C focuses on the packetization of CHI messages, making them suitable to be transported over a chip-to-chip link. The packetization format is optimized for link utilization and latency, while avoiding complex packing and unpacking schemes.

The two primary use cases are:

- Multi-chip Symmetric Multi-Processor (SMP)
- Multi-chip coherent accelerator attach

The C2C extension can also be referred to as Chip(let)-to-Chip(let), as it covers both the board-level connection of multiple chips and the package level connection of multiple dies or chiplets.

B1.1.1 Multi-chip Symmetric Multi-Processor (SMP)

The SMP configurations of interest are a small number of chips that are similar in their functionality and tightly connected together. Each chip includes *System-on-Chip* (SoC) components with a large number of processing cores. Each chip is expected to have attached memory that is coherently shared by processing cores on all the chips.

Figure B1.1 shows an example of a two-chip topology.

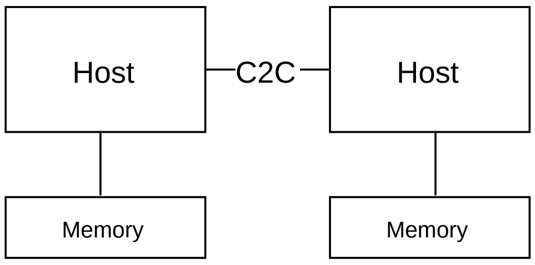

Figure B1.1: Example of a two-chip topology

Figure B1.2 shows an example of four chips in a fully connected topology. The attached memories will exist but are not shown in the figure.

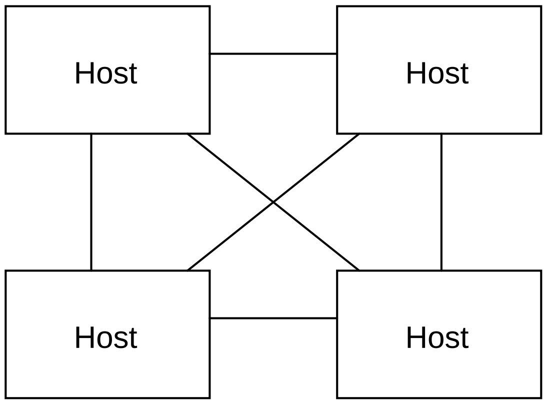

Figure B1.2: Example of four chips in a fully connected topology

B1.1.2 Multi-chip coherent accelerator attach

The accelerator attach configurations of interest are one or more coherent accelerators connected to a host chip.

The accelerators can be fully coherent or IO coherent.

Figure B1.3 shows an example of a topology of coherent accelerators.

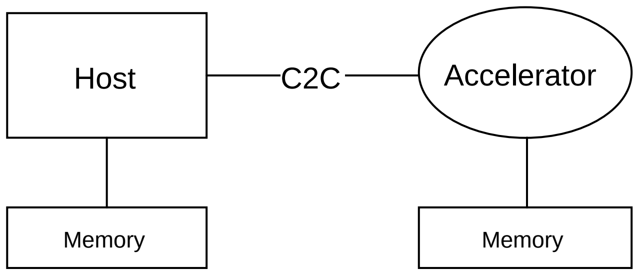

Figure B1.3: Example of a topology with coherent accelerators

B1.2 Terminology

The following terms have a specific meaning within this specification:

DN

DVM Node (DN) is a node located within the chip that processes DVM operations to and from remote chips.

RP

Resource Planes are link buffer resources enabling groups of messages within the same Message Class to be independently transported over the link.

Transmitter

An interface component that forwards messages to the Link layer.

Receiver

An interface component that receives messages from the Link layer.

Container

A container is the smallest fixed size unit in which messages are transferred across an interface.

Chapter B2
Interface structures

This chapter describes the structure of the C2C interface. It contains the following sections:

B2.1 Interface layers

The functional logic between on-chip CHI interface and the chip pins that a message traverses through is divided into multiple functional layers. These layers are referred to as:

- Protocol layer
- Packetization layer
- Link layer
- Physical layer

Link layer and Physical layer presence is expected, but not required, for connecting two chips together.

Figure B2.1 shows the C2C interface structure.

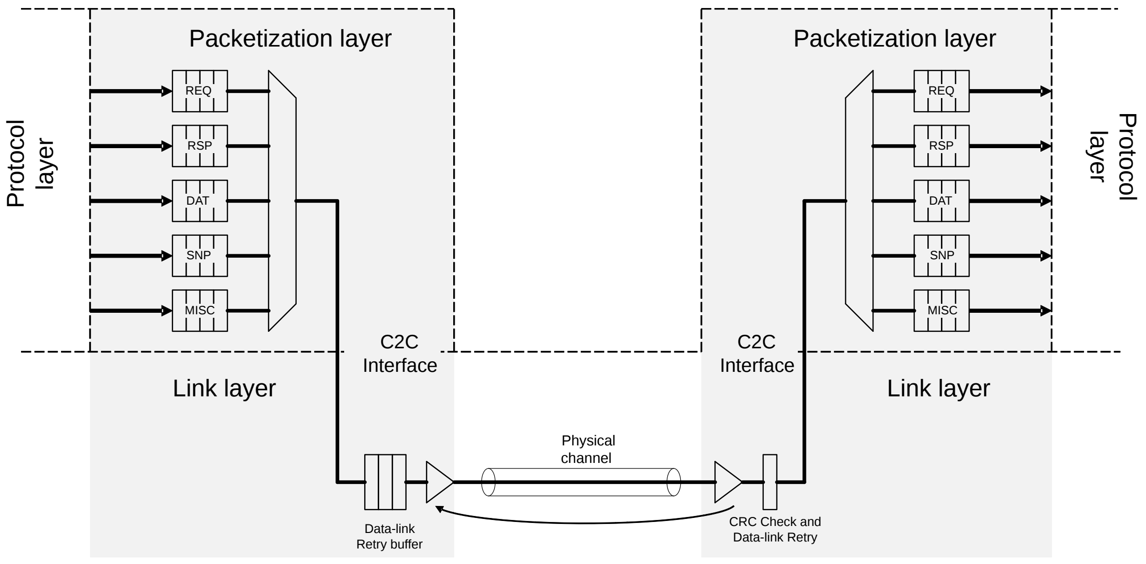

Figure B2.1: C2C interface structure

B2.1.1 Protocol layer

The Protocol layer handles protocol conversion between on-chip CHI and CHI C2C.

The logic required for the conversion is minimal.

B2.1.2 Packetization layer

When the Link layer is present, the Packetization layer is between the Protocol layer and the Link layer.

If the Link layer is not present, the Packetization layer connects directly to the external signals. If required, the Packetization layer packs outgoing messages into a fixed size container and unpacks incoming containers into individual messages. The incoming messages are forwarded to the Protocol layer or other functional blocks within the Packetization layer.

B2.1.3 Link layer

The Link layer is responsible for handling message transport at a *FLow control unIT* (Flit) granularity.

The Link layer is also responsible for optional data integrity and transport error detection and recovery. Typically, this is done by adding a generated *Cyclic Redundancy Check* (CRC) checksum to a flit being transmitted and checking the CRC checksum on the received flit.

When an error occurs, the Link layer can use a flit retry mechanism to recover from the error.

B2.1.4 Physical layer

The Physical layer is responsible for providing reliable electrical connectivity between the two chips.

Some of the capabilities of the Physical layer include the logic to overcome latency skew among different signals and link retraining to maintain signal integrity.

There must be a space in a container for the Link layer to fill the following transport information:

- Flit header
- Link reliability

The detail of the Link layer and Physical layer is outside the scope of this specification.

B2.2 Message classes

This specification defines a message class as the group of messages of a single CHI message type. Message classes are similar to on-chip CHI physical channels.

Messages are grouped into the following five message classes:

- Request, REQ
- Response, RSP. Does not include data.
- Snoop, SNP
- Data, DAT
- Miscellaneous, MISC

MISC, a C2C-specific message class, is defined to group all messages that do not directly relate to on-chip CHI protocol messages. Example use of MISC messages are interface initialization and message credit grant.

For further MISC message details, see B4.2.7 *Uncredited Miscellaneous message fields*.

B2.3 Channel connection

There are two different approaches to connect the different message class channels between chips:

1. Shared channel connect
2. Independent channel connect

The common usage for a C2C interface is expected to be a shared channel connection. All on-chip channels share the same connection between chips.

A fixed size container is used as the unit of exchange across the C2C interface. The messages to be sent are placed in a container. The container can include multiple messages from the same message class as well as messages from different message classes.

The use of a container with messages included from any message class instead of using independent physical channels for each message class typically results in a better link efficiency.

Figure B2.2 shows a shared channel connection.

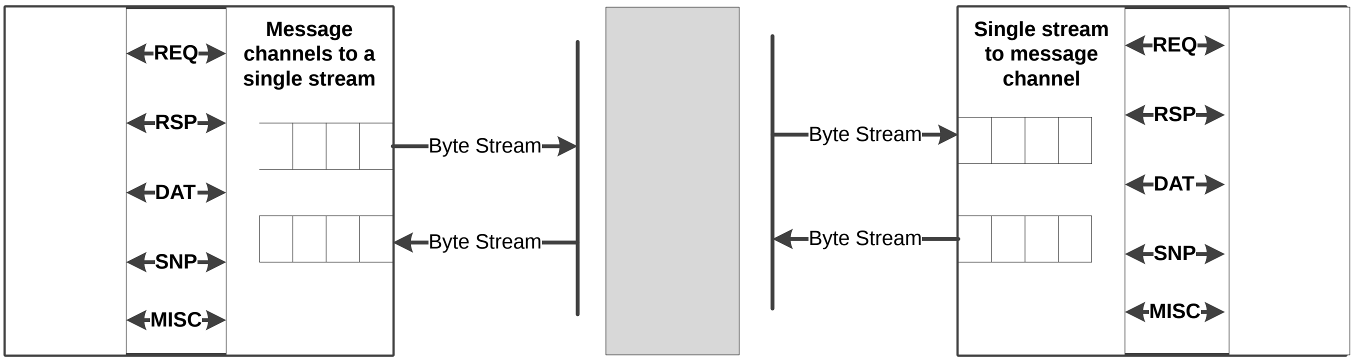

Figure B2.2: Shared channel connect

Alternatively, an independent channel connect can be used, where a separate physical set of wires is used for each message class. This type of connection provides implementation ease due to not requiring packetization logic. However, this is at the cost of potential underutilization of the physical channels due to lack of messages to fully utilize all available channels.

B2.4 Multiple interfaces

It is expected that a single C2C interface is used to connect two chips.

It is permitted that multiple C2C interfaces are used between two chips to meet greater throughput requirements.

When using multiple interfaces, both the chips must follow message association requirements.

Within transaction	For a Request or Snoop, all associated Data or Response messages must use the same interface.
Between transaction	All requests and snoops to the same address must use the same interface. All endpoint ordered requests must use the same interface. DVMReq and DVMSync transactions from a given source must use the same interface. See B6.3 *DVM transaction flows*. MISC messages to the same target must use the same interface.

A Completer can ensure that a response is sent on the same interface as the corresponding memory or snoop request by tracking the interface used by the request.

A Home can ensure snoop requests are sent on the same interface as requests to that address by either tracking the interface the request was received on or by using a common address hash.

When using an address hash:

- A chip is required to use a single address hash for requests to a given target chip.
- Requests to different target chips can use different address hash functions.
- It is permitted to use an approach where both the request and snoop sides are able to be programmed with a hash function. An example hash function is provided in B2.4.1 *Hash function example*.

The number of interfaces permitted for aggregating traffic between two components is 1 to 16.

B2.4.1 Hash function example

Figure B2.3 shows how the interface number is determined from an address hash.

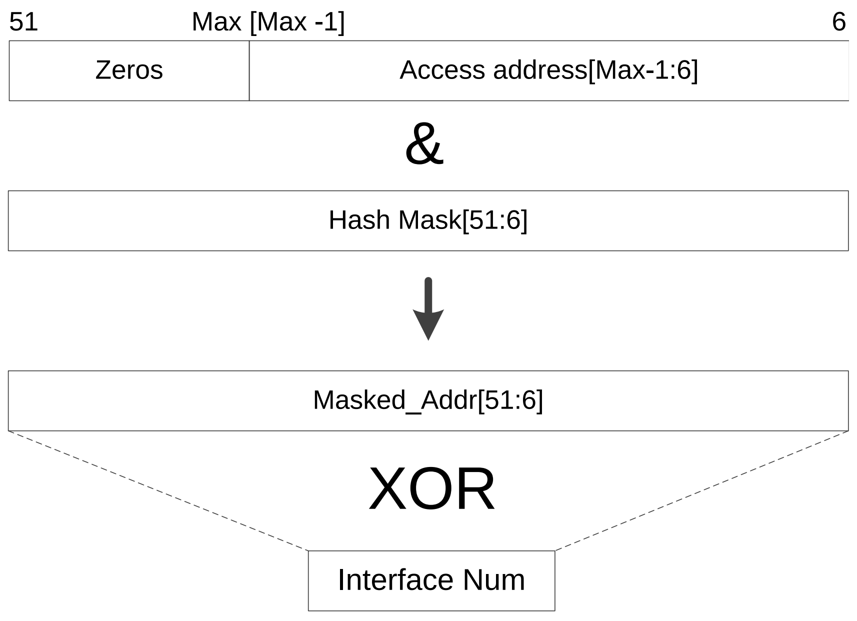

Figure B2.3: Interface determination from address hash

The following steps are used to determine the interface on which to send a request or a snoop:

1. The cache line aligned input address is first filtered through a Hash Mask. The most significant address bits that are not used must be 0 before filtering the address. In this example, the result of the filtering is labeled as Masked_Addr.
2. The individual bits of the Masked_Addr are XORed to obtain the target interface in a manner defined by the number of interfaces between the two chips.

The following are examples of determining the interface number for a given address when the number of interfaces is:

- A power-of-two

  For 2 interfaces Interface Num[0] = Masked_Addr[n-1] XOR Masked_Addr[n-2]...Masked_Addr[7] XOR Masked_Addr[6]

  For 4 interfaces Interface Num[1:0] = Masked_Addr[n-1:n-2] XOR Masked_Addr[n-3:n-4]...Masked_Addr[9:8] XOR Masked_Addr[7:6]

  For 8 interfaces Interface Num[2:0] = Masked_Addr[n-1:n-3] XOR Masked_Addr[n-4:n-6]...Masked_Addr[11:9] XOR Masked_Addr[8:6]

- Non power-of-two.
  The first step is to perform the hash using a power-of-two number of interfaces which is larger than the available number of interfaces. The obtained intermediate interface number is mapped to the interface number by using a table or by some other implementation means. The table is heuristically set up to distribute, as evenly as possible, the hashed numbers across the available number of interfaces. The larger the intermediate interface number relative to the actual number of interfaces, the greater than the uniformity of message distribution across the interfaces.

Chapter B3
Packetization

This chapter describes the process of packetization within the C2C interface. It contains the following sections:

B3.1 Containers

A container is 256 bytes and comprises Link header bytes, Protocol header bytes and message bytes organized in granules.

The following two container formats are defined:

1. Format X, comprising of:
    - 240 bytes of payload, split into twelve 20-byte granules
    - 6 bytes of Link header
    - 10 bytes of Protocol header
    - This format is compatible with UCIe [2] 256-byte Latency Optimized Mode with optional bytes.
2. Format Y, comprising of:
    - 226 bytes of payload, split into:
        - Ten 20-byte granules
        - One 16-byte granule
        - One 10-byte granule
    - 20 bytes of Link header
    - 10 bytes of Protocol header
    - This format is compatible with CXL [3] 256-byte latency optimized flit format.

Figure B3.1 shows the container byte layouts for Format X and Format Y. The granules are distributed so they are aligned across 64-byte and 128-byte parts of the container.

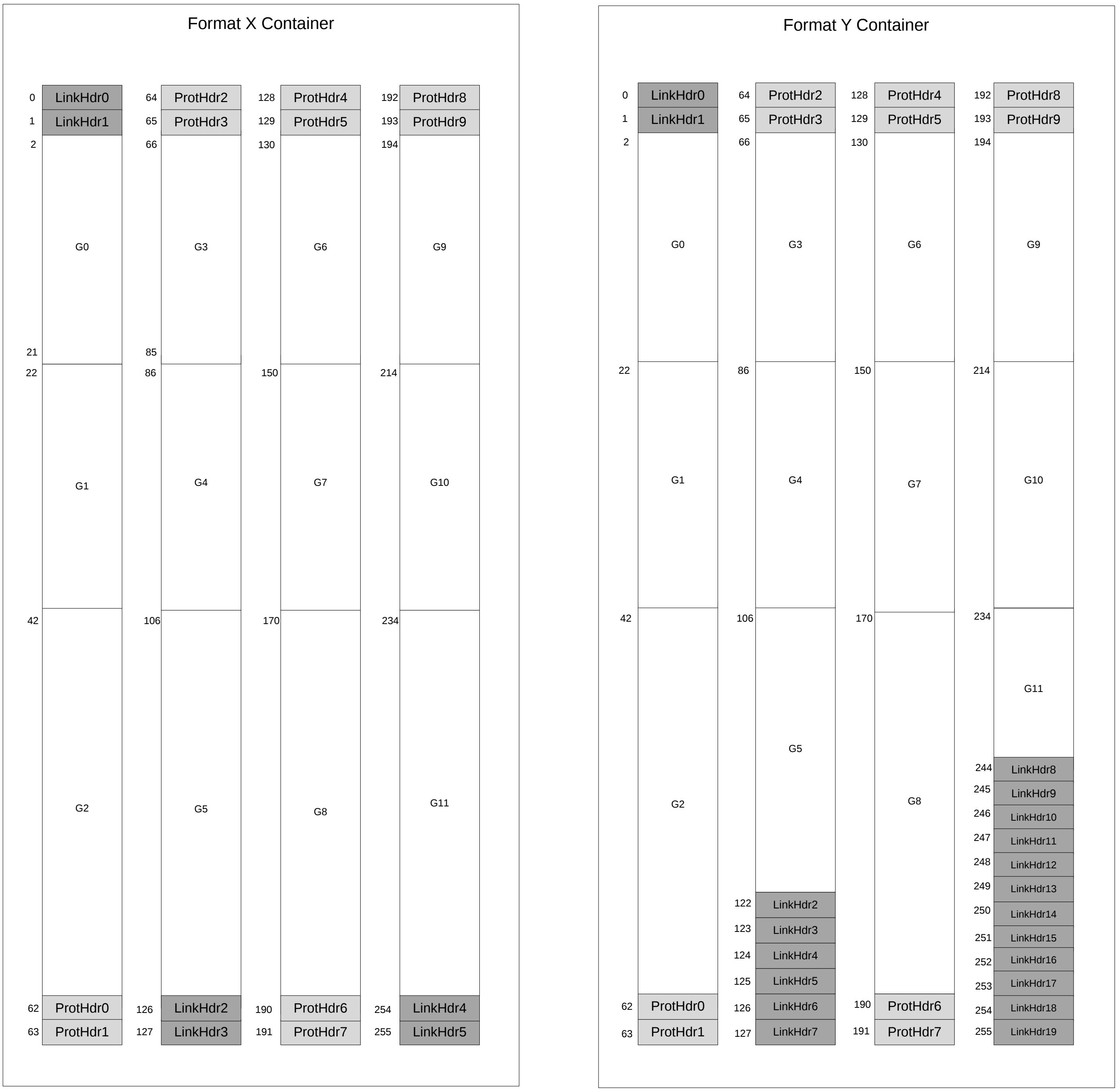

Figure B3.1: Container layout format

For information on ProtHdr bits, see B3.2 *Protocol header*.

B3.2 Protocol header

The bytes which are not used for message granules are either the Link header or the Protocol header.

The description of Link header bytes are outside the scope of this specification. Typical usage for these bits are for adding a flit header, a CRC, and an FEC.

The Protocol layer header bits are allocated as shown in Figure B3.2.

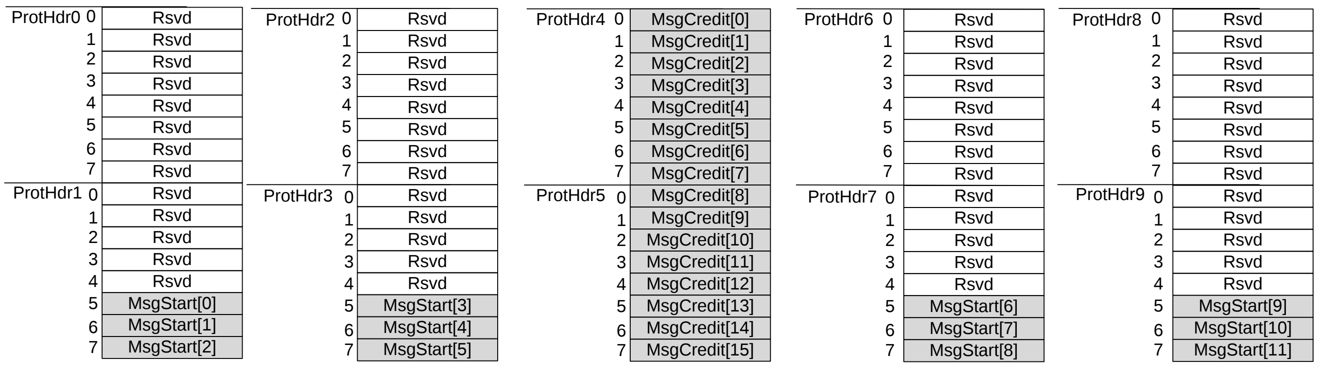

Figure B3.2: Protocol header field layout

Table B3.1 shows the encoding of the Protocol Header fields present in Figure B3.2.

Table B3.1: Protocol header fields

Field name	Width	Description
MsgCredit	16	Message credits granted by the Receiver. See B5.2 *Message credits*.
MsgStart	12	MsgStart[i] indicates if a message starts in granule i. There is a MsgStart bit for each of the granules. **MsgStart[i] = 0** A message does not start in the granule i. The granule might be empty or be part of a message that started in an earlier granule. **MsgStart[i] = 1** A message starts in the granule i. That is, there is a MsgType field in the first bits of that granule.

B3.3 Message packing

Messages are packed into a container using granules. Messages are only permitted to begin at the start of a granule. This limits the number of positions each message type can occupy within a container to minimize decode logic.

The MsgStart vector in the Protocol Header determines if a new message starts at a particular granule. Each message begins with a MsgType field that defines the type of the message.

B3.3.1 Rules and restrictions

The restrictions on message placement within a container are:

- In Format X container, a message can be placed in any granule.
- In Format Y container:
  - A message, except Resp and Misc type, can be placed in any granule except G5 and G11.
  - A Resp can be placed in any granule.
  - A Misc message of:
    - A total message size of 10 bytes or less can be placed in any granule.
    - A total message size of 16 bytes or less can be placed in any granule except G11.
    - A total message size greater than 16 bytes can be placed in any granule except G5 and G11.
- Multi-granule messages:
  - Can start in any granule, except G5 and G11 in Format Y.
  - Must be contiguous, except when spanning G5 or G11 in Format Y. Multi-granule messages in Format Y skip over G5 and G11.
  - Can span across consecutive containers. When spanning multiple containers, the message must continue in G0 of the next container.

Additionally, the container packing rules are:

- Inclusion of a message in a granule is optional. A container with no messages is permitted.
- The LSB of a message is placed in the LSB of the granule.
- Unused bits must be 0, including:
  - Granules that do not include a message.
  - Any bits between the end of a message and the end of the granule.
- 32-byte data is placed in the half of the 64-byte Data field, corresponding to the request address bit[5] value.
- Granules are organized into groups: G0-G2, G3-G5, G6-G8, G9-G11.
  - Each group can have:
    - No granules populated.
    - The lowest numbered granule populated.
    - The two lowest numbered granules populated.
    - All granules populated.
    - A maximum of one MiscU message.
    - A maximum of four response messages, where Resp2 counts as two response messages.

  For example, if G6 is not populated, then G7 and G8 must also be empty.
  - A MiscU.LinkStatus message can only use G0.

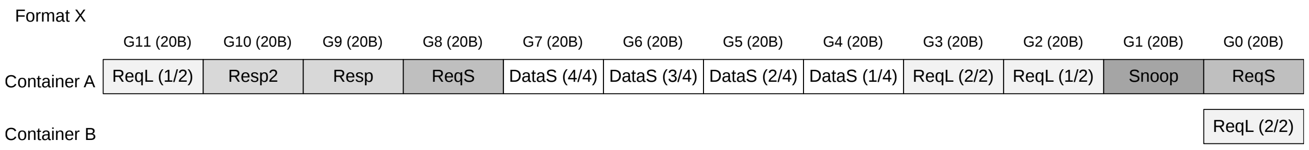

Figure B3.3: Example of Format X container packing

Figure B3.3 illustrates an example of Format X container packing. The two containers in Figure B3.3 include:

- Two ReqS and two ReqL messages, where one ReqL message spans over two containers.
- One Resp and one Resp2 message.
- One Snoop message.
- One DataS message.
- Empty granules in Container B.

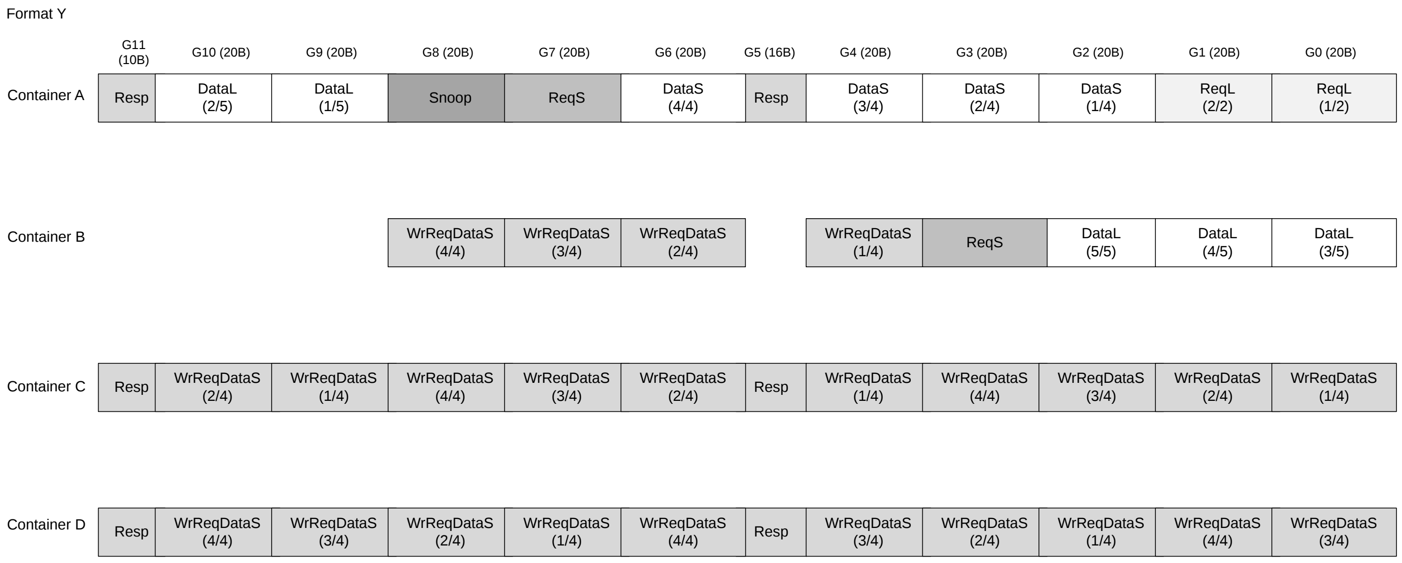

Figure B3.4: Example of Format Y container packing

Figure B3.4 illustrates an example of Format Y container packing. The four containers in Figure B3.4 include:

- Two ReqS messages and one ReqL.
- Six Resp messages.
- One Snoop message.
- One DataS message and one DataL message.
  - The DataS message in Container A spans over G5.
  - The DataL message starts in Container A, skips over G11, and rolls over to the beginning of Container B.
- Six WrReqDataS messages.
  - One WrReqDataS message in Container B, spans over G5.
  - One WrReqDataS message starts in Container C G9, skips over G11, and rolls over to the beginning of Container B.
- Empty granules in Container B.

Chapter B4
Message structures

This chapter describes the extent of support for on-chip CHI transaction flows, the on-chip CHI fields in messages that are optimized out as they are not required and resulting message formats with list of fields in each message class. It contains the following sections:

B4.1 Transactions and transaction flows

This section describes the mapping of fields in a request received by a Subordinate Node to on-chip CHI fields, and also lists the CHI transactions that are not supported.

B4.1.1 Field mapping in requests to Subordinate Nodes

An interface receiving requests targeting Subordinate Node must map the fields in the received request to the on-chip CHI request message in the following manner:

- Duplicate SrcID onto ReturnNID
- Duplicate TxnID onto ReturnTxnID
- Set DoDWT to 0

B4.1.2 Unsupported transactions

The following transactions are not supported:

- Forwarding snoops
- Exclusive access to Non-cacheable and Device memory
- Request Retry flows
- *Direct Memory Transfer* (DMT). The ReturnNID and ReturnTxnID fields used for DMT are not included in a Request message.
- *Direct Cache Transfer* (DCT). The FwdNID and FwdTxnID fields used exclusively for DCT are not included in a Snoop message.
- *Direct Write Transfer* (DWT). The ReturnNID field used for DWT is not included in a Request message.
- When Tag Match is enabled, direct TagMatch response to Request Node from Subordinate Node is not supported.
- In Persist *Cache Maintenance Operation* (CMO) transaction flow, direct Persist response to Request Node from Subordinate Node is not supported.

Note

For a Read transaction, the chip which includes the Home Node and Subordinate Node can use DMT internally to that chip. This allows data to be forwarded directly to the C2C port from the Subordinate Node.

However, because DMT is not supported across the C2C interface, the data response only has a single SrcID field, rather than having both SrcID and HomeNID fields. At the C2C port, the SrcID field in the data response must be populated with the Node ID of the Home. This ensures that any subsequent message in the transaction, such as CompAck, is correctly sent to the Home.

For Write transactions and Snoop transactions, the SrcID field of the data response will contain the correct Node ID and does not need to be replaced with the Home ID.

At the Requester chip interface, the SrcID and HomeNID in the responses to the Requester must be populated by duplicating the common field value.

DCT support can be partially supported at the Home chip egress by keeping track of the snoop type before converting the Forwarding snoop into a Non-forwarding snoop. When the snoop response with data is received, the interface logic can split the response where one is sent to the Requester and the other is sent to the Home.

B4.2 Message fields

This section describes the fields in each message class.

Fields that are present for C2C, but not in on-chip CHI messages, are described in B4.2.1 *C2C specific message fields*. On-chip CHI messages that are removed in C2C messages are described in B4.2.8 *Redundant fields*.

B4.2.1 C2C specific message fields

The following fields are added to C2C messages and are not present in on-chip CHI messages:

- MsgType
- ResPlane
- SharedCrdt
- ChunkValid
- RsvdZero

B4.2.1.1 Message type, MsgType

This field identifies the message type in terms of message class and size, where:

- Request, Data, WrReqData messages can be one of two types, dependent upon message length. See B4.2.2 *Request message fields*, B4.2.4 *Data message fields*, and B4.2.5 *WritePush message* for more details.
- Response messages can be one or two per granule of 20 bytes. See B4.2.3 *Response message fields*.
- Miscellaneous messages can be credited or uncredited.
  - See B4.2.7 *Uncredited Miscellaneous message fields* for details on uncredited MISC messages.
  - This specification does not yet define any credited MISC messages.

Table B4.1 shows the encoding of the MsgType field.

Table B4.1: MsgType encoding

MsgType	Type	CHI channel
`0b0000`	MiscU	-
`0b0001`	MiscC	-
`0b0010`	ReqS	REQ
`0b0011`	ReqL	REQ
`0b0100`	Resp	RSP
`0b0101`	Resp2	RSP
`0b0110`	Snoop	SNP
`0b0111`	DataS	DAT
`0b1000`	DataL	DAT
`0b1001`	WrReqDataS	REQ and DAT
`0b1010`	WrReqDataL	REQ and DAT
`0b1011` - `0b1111`	Reserved	

B4.2.1.2 Resource plane, ResPlane

This field is used to support multiple resource planes in Request and WritePush messages. See B5.3 *Resource Planes* and B5.4 *Data credit pools*.

B4.2.1.3 Shared credit, SharedCrdt

This field is used to support multiple resource planes and data credit pools in Request and Data message class messages. See B5.3 *Resource Planes* and B5.4 *Data credit pools*.

B4.2.1.4 Chunk valid, ChunkValid

This field is present in DataS, DataL, WrReqDataS, and WrReqDataL messages. Its value indicates if one or both of the 32 byte data chunks have valid data.

Table B4.2 shows the ChunkValid field encodings.

Table B4.2: ChunkValid field encoding

ChunkValid[1:0]	Valid chunk
`0b00`	Reserved
`0b01`	Lower 32 bytes
`0b10`	Upper 32 bytes
`0b11`	All 64 bytes

B4.2.1.5 RsvdZero

This field indicates that the use of the bits is Reserved and a Transmitter must set the field value to zero, a Receiver must ignore the field and an intermediate agent must forward them unchanged. The inclusion of RsvdZero field in a message may be for starting the subsequent field at a byte or granule boundary or to fill the bits beyond useful information in the last granule.

B4.2.2 Request message fields

The fields in the Request message types are described in this section:

- ReqS, for short-sized formats
- ReqL, for long-sized formats

A Request message uses a ReqL message format when any of the following fields are present in the request and have a non-zero value:

- Addr[3:0] in any request. Addr[5:0] bits in Dataless requests of 64 bytes size to normal memory are permitted to be zeroed.
- LPID in an exclusive transaction.
- GroupID in a Persist CMO, a StashOnceSep, or a Write with TagOp value of *Match*.
- StashNID[10:1]
- StashLPIDValid
- PBHA

- LikelyShared
- DataTarget[6:1]
- RSVDC[N-1:16]

When the the use of ReqS is allowed, a Transmitter is permitted to use ReqL messages in place of ReqS. The use of ReqL instead of ReqS may lead to reduced container packing efficiency.

Table B4.3 lists Request message fields and their placement into ReqS and ReqL.

The columns in the table are defined as follows:

Common Fields marked with the same 'x' in 'Cx' share a common position in the message.

ReqS The request includes all mandatory fields and most commonly used optional fields.

ReqL Request size is extended to include all supported Request fields.

Table B4.3: Request message fields

Common	Field name	Width (bits)	
		ReqS	ReqL
	MsgType	4	4
	SharedCrdt	1	1
	ResPlane	3	3
	QoS	4	4
	SrcID	11	11
	TxnID	12	12
	NS	1	1
	NSE	1	1
	SecSID1	1	1
	Order	2	2
	MemAttr	4	4
	ExpCompAck	1	1
	TraceTag	1	1
	Addr[51:6]	46	46
	Addr[5:4]	2	2
	SnpAttr	1	1
	MPAM	15	15
C0	MECID StreamID	16	16
	RSVDC[15:0]	16	16
	Size	3	3
	Opcode	7	7

Continued on next page

Table B4.3 – *Continued from previous page*

Common	Field name	Width (bits)	
		ReqS	ReqL
	TagOp	2	2
C1	StashNIDValid Endian Deep PrefetchTgtHint	1	1
C2	Excl SnoopMe CAH	1	1
C3	DataTarget[0] StashNID[0]	1	1
	RsvdZero	3	3
	PBHA	0	4
	Addr[3:0]	0	4
	StashLPIDValid	0	1
	StashLPID	0	5
C4	{4'b0, DataTarget[6:1]} StashNID[10:1]	0	10
C5	{3'b0, LPID} PGroupID StashGroupID TagGroupID	0	8
	RSVDC[23:16]	0	8
	RSVDC[31:24]	0	8
	LikelyShared	0	1
	RsvdZero	0	111
Total (bits)[a]		157	206

[a] The total bit count does not include the RsvdZero bits.

B4.2.3 Response message fields

The fields in the Response message types are described in this section:

- Resp
- Resp2

B4.2.3.1 Resp message type

Table B4.4 shows the fields in a Resp message type, where one response is packed into a single granule.

Table B4.4: Resp response message fields

Common	Field name	Width (bits)
	MsgType	4
	QoS	4
	TgtID	11
	SrcID	11
	TxnID	12
	Opcode	5
	RespErr	2
	Resp	3
	DataPull	1
	CBusy	3
	TagOp	2
	TraceTag	1

C6	DBID
	{4'`b0`, PGroupID}
	{4'`b0`, StashGroupID}
	{4'`b0`, TagGroupID}
		12

	RsvdZero	9
Total (bits)[a]		71

[a] The total bit count does not include the RsvdZero bits.

B4.2.3.2 Resp2 message type

Table B4.5 shows the fields in a Resp2 message type, where two responses are packed into a single granule.

Table B4.5: Resp2 response message fields

Common	Field name	Width (bits)
	MsgType	4
	QoS	4
	TgtID	11
	SrcID	11
	TxnID	12
	Opcode	5
	RespErr	2

Continued on next page

Table B4.5 – *Continued from previous page*

Common	Field name	Width (bits)
	Resp	3
	DataPull	1
	CBusy	3
	TagOp	2
	TraceTag	1

C6	DBID
	{4'b0, PGroupID}
	{4'b0, StashGroupID}
	{4'b0, TagGroupID}
		12

	RsvdZero	9
	RsvdZero	4
	QoS	4
	TgtID	11
	SrcID	11
	TxnID	12
	Opcode	5
	RespErr	2
	Resp	3
	DataPull	1
	CBusy	3
	TagOp	2
	TraceTag	1

C6	DBID
	{4'b0, PGroupID}
	{4'b0, StashGroupID}
	{4'b0, TagGroupID}
		12

	RsvdZero	9
Total (bits)[a]		138

[a] The total bit count does not include the RsvdZero bits.

B4.2.4 Data message fields

DataS and DataL message formats are used for read data, snoop response data, and non-WritePush data messages.

The characteristics of DataS and DataL message format include:

- SrcID and HomeNID share a single 11-bit field. This field is used by:
  - HomeNID in CompData and DataSepResp

  - SrcID in SnpRespData and WriteData

- Data poison is a single bit per data transfer and is indicated by a RespErr value of DERR.
- The ChunkValid field is used to indicate if the corresponding 32 bytes includes valid data. See Table B4.2.
- The Data field starts at a byte boundary. This requires a RsvdZero bit padding before the Data field.
- Fields exclusively in DataL message start at a granule boundary.
- Invalid bytes in the Data field in any data message, indicated by ChunkValid, and BE in write data and snoop response data, must be zeroed.

The DataL message format must be used instead of DataS when any of the following conditions are true:

- The BE field in a data response to a partial write request has one or more bits with zero value.
- The following fields are present in the message and have a non-zero value:
  - QoS
  - PBHA
  - RSVDC[N-1:16]

When the use of DataS is allowed, a Transmitter is permitted to use DataL message in place of DataS. The use of DataL instead of DataS may lead to reduced container packing efficiency.

Table B4.6 shows the message fields for DataS and DataL.

Table B4.6: DataS and DataL message fields

Common	Field name	DataS	DataL
	MsgType	4	4
	SharedCrdt	1	1
	RsvdZero	1	1
	ChunkValid	2	2
	TgtID	11	11
C8	SrcID HomeNID	11	11
	TxnID	12	12
	Opcode	4	4
	RespErr	2	2
	Resp	3	3
	DataSource	8	8
	DataPull	1	1
	CBusy	3	3
	CCID	2	2
	TagOp	2	2
	Tag	16	16
	TU	4	4
	TraceTag	1	1

Continued on next page

Table B4.6 – *Continued from previous page*

Common	Field name	DataS	DataL
	CAH	1	1
C9	{4'b0, DBID} MECID	16	16
	RSVDC[15:0]	16	16
	RsvdZero	7	7
	Data	512	0
	RsvdZero	0	32
	RSVDC[31:16]	0	16
	QoS	0	4
	PBHA	0	4
	RsvdZero	0	40
	BE	0	64
	Data	0	512
Total Non-data (bits)[a]		120	208
Total (bits)[a]		632	720

[a] The total bit count does not include the RsvdZero bits.

B4.2.5 WritePush message

Immediate Writes, Atomics and DVMReq transactions with NonCopyBackWriteData data, are permitted, but not required, to optimize their transaction flow by using write push semantics. A Write push flow combines the request and data sent from the Transmitter to Receiver into a single message, thus eliminating the need for a DBIDResp response.

This two-step write push over the three-step write pull transaction allows the following advantages:

- Reduced number of messages, therefore improving link utilization.
- Fields that are present in both Request and Data only need to be sent once.
- Reduced request buffer occupancy time.

The following interface properties are defined to support WritePush transactions:

WritePush_Support This property indicates if the interface supports the use of WritePush for Immediate Writes and Atomic transactions. See B10.4.2 *Receiver and Transmitter properties*.

DVMPush_Support This property indicates if the interface supports the use of WritePush for DVM transactions. See B10.4.2 *Receiver and Transmitter properties*.

See B4.2.5.2 *Message fields* for message format details.

B4.2.5.1 Transaction flow

Figure B4.1 shows the transaction flow for an Immediate Write transaction using write push semantics.

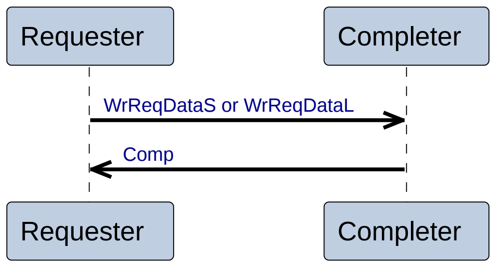

Figure B4.1: WritePush transaction flow

The sequence for the WritePush transaction is:

- The transaction starts with the Requester issuing a WrReqDataS or WrReqDataL message.
- The Completer sends a Comp response.

B4.2.5.2 Message fields

Table B4.7 lists the request opcodes that are permitted to use a WrReqDataS message. In these requests, the BE field values must all be 1. All other push transactions, including those that require fields which are not included in a WrReqDataS message, must use a WrReqDataL message format.

In WrReqDataS and WrReqDataL, invalid bytes in the Data field must be zeroed.

Note

The interface can use ChunkValid to determine which bytes to be zeroed.

The OWO field in Table B4.7 indicates if the write is part of a group of writes that require *Ordered Write Observation* (OWO) guarantees.

When OWO = 0	OWO guarantees are not required.
When OWO = 1	OWO guarantees are required.

When the rules permit the use of WrReqDataS, a write push message is permitted to use WrReqDataL instead of WrReqDataS. The use of WrReqDataL may lead to reduced container packing efficiency.

Table B4.7: WrReqDataS and WrReqDataL message fields

Common	**Field name**	**Width (bits)**	
		WrReqDataS	**WrReqDataL**
	MsgType	4	4
	SharedCrdt	1	1
	ResPlane	3	3
	QoS	4	4
	SrcID	11	11
	TxnID	12	12
	NS	1	1
	NSE	1	1

Continued on next page

Table B4.7 – *Continued from previous page*

Common	Field name	Width (bits)	
		WrReqDataS	WrReqDataL
	SecSID1	1	1
	Order	2	2
	MemAttr	4	4
C10	ExpCompAck OWO	1	1
	TraceTag	1	1
	Addr[51:6]	46	46
	Opcode2	2	0
	Union1[a]	34	0
	Data	512	0
	Addr[5:4]	0	2
	SnpAttr[b]	0	1
	MPAM	0	15
	MECID	0	16
	RSVDC[15:0]	0	16
	Size[c]	0	3
	Opcode	0	7
	TagOp[d]	0	2
C1	StashNIDValid Endian Deep PrefetchTgtHint	0	1
C2	Excl SnoopMe CAH	0	1
C3	DataTarget[0] StashNID[0]	0	1
	RsvdZero	0	3
	PBHA	0	4
	Addr[3:0]	0	4
	StashLPIDValid	0	1
	StashLPID	0	5
C4	{4'`b0`, DataTarget[6:1]} StashNID[10:1]	0	10

Continued on next page

Table B4.7 – *Continued from previous page*

Common	Field name	Width (bits)	
		WrReqDataS	WrReqDataL
C5	{3'`b0`, LPID} PGroupID StashGroupID TagGroupID	0	8
	RSVDC[23:16][e]	0	8
	Tag	0	16
	TU	0	4
	ChunkValid	0	2
	RespErr	0	2
	BE	0	64
	Data	0	512
Total Non-data (bits)[f]		128	285
Total (bits)[f]		640	797

[a] This field is shared among MECID, MPAM, and RSVDC. See Figure B4.2.

[b] For WrReqDataS, this field is derived from Opcode2.

[c] For WrReqDataS, this field has 64 bytes implicitly.

[d] The TagOp value in the Request is the same as the TagOp value in the Data message.

[e] A maximum of 24 bits for the RSVDC is permitted for C2C interface.

[f] The total bit count does not include the RsvdZero bits.

Table B4.8 shows the Opcode2 field in a WrReqDataS message.

Table B4.8: Opcode2 field encoding in WrReqDataS message

Opcode2[1:0]	WrReqDataS command	Comments
`00`	WriteNoSnpFull	
`01`	WriteUniqueFull	
`10`	WriteNoSnpPtl	All 64 BE bits set
`11`	WriteUniquePtl	All 64 BE bits set

Figure B4.2 illustrates the sharing of the Union1 field among MPAM, MECID, and RSVDC fields.

The lower order bits of the Union1 field determine how the other bits of the field are used for MPAM, MECID, and

RSVDC values.

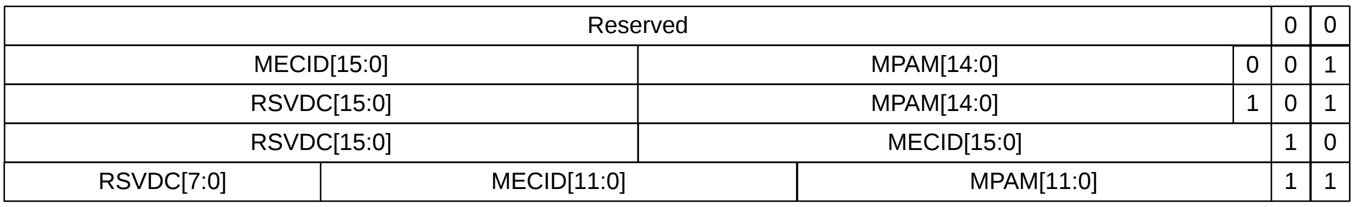
Reserved | 0 | 0
MECID[15:0] | MPAM[14:0] | 0 | 0 | 1
RSVDC[15:0] | MPAM[14:0] | 1 | 0 | 1
RSVDC[15:0] | MECID[15:0] | 1 | 0
RSVDC[7:0] | MECID[11:0] | MPAM[11:0] | 1 | 1

Figure B4.2: Union1 field use for MPAM, MECID, and RSVDC

B4.2.6 Snoop message fields

Table B4.9 shows the fields in a Snoop message.

Table B4.9: Snoop message fields

Common	Field name	Width (bits)
	MsgType	4
	QoS	4
	TgtID	11
	SrcID	11
	TxnID	12
	PBHA	4
	StashLPIDValid	1
	StashLPID	5
	Opcode	5
	Addr	48
	NS	1
	NSE	1
	DoNotGoToSD	1
	RetToSrc	1
	TraceTag	1
	MPAM	15
	MECID	16
	RsvdZero	19
Total (bits)[a]		141

[a] The total bit count does not include the RsvdZero bits.

B4.2.7 Uncredited Miscellaneous message fields

The fields in a MiscU message are shown in Table B4.10.

The number of Payload bits used by a MiscU message is dependent on the MISC message type.

The following key is used:

X Variable field width, where the precise value is dependent on the MiscOp field value.

Table B4.10: MiscU message fields

Field name	Width (bits)
MsgType	4
MiscOp	4
Payload	X
Total (bits)	8 + X

MiscOp describes the message subtypes supported by Miscellaneous message class.

The encoding of MiscOp field in a MiscU message is shown in Table B4.11.

Table B4.11: Encoding of MiscOp field in MiscU message

MiscOp[3:0]	MiscU message
`0000`	Reserved
`0001`	Reserved
`0010`	Activation
`0011`	Connect
`0100`	CrdtGrant
`0101`	Properties
`0110`	LinkStatus
Others	Reserved

B4.2.8 Redundant fields

Certain CHI fields are not required in C2C messages because of one or more of the following reasons:

- The transaction flow that uses the field is not supported.
- The field is used in messages in the Home Node to Subordinate Node interface only to support flow optimizations not supported in C2C.
- Routing of messages of that message class is based on address decode rather than ID.

Table B4.12 shows the CHI fields not required in C2C messages.

Table B4.12: CHI message fields not inlcuded in C2C messages

Message Class	Field name	Bits	Description
Request	TgtID	11	This field is removed as requests are routed using address decode rather than using TgtID.
	ReturnNID ReturnTxnID DoDWT	11 12 1	These fields are removed as they are only used in requests from Home Node to Subordinate Node to provide an optimized direct path response communication between a Request Node and a Subordinate Node. Direct responses between these nodes are not supported.
	AllowRetry PCrdType	1 4	These fields are removed as they are only used in a Request Retry transaction flow.
Response	FwdState	3	This field is removed as it is only used in DCT transaction flows.
	PCrdType	4	This field is removed as it is only used in Request Retry transaction flows.
Data	FwdState	3	This field is removed as it is only used in DCT transaction flows.
	DataID	2	This field is removed because data is always sent across a C2C inteface in a single 64-byte block.
	Poison	8	This field is removed as the poison indication is per the cache line and is encoded in RespErr field. The RespErr encoding of `0b10` is used to indicate Posion or Data Error.
	DataCheck	64	This field is removed as the Link layer may use its own proprietary mechanism, such as CRC and *Forward Error Correction* (FEC), to support transport reliability. This makes the DataCheck field redundant.
	NumDat Replicate	2 1	These fields are removed as they are used to support limited data elision. A cache line data is always transported as a 64B granule.
Snoop	FwdNID FwdTxnID	11 12	These fields are removed as they are only used in Forwarding snoops.
	VMIDExt	8	This field is removed as it is only used in DVM snoops. DVM snoops are sent across the C2C interface as DVM requests, therefore this field is not required in snoops.

Chapter B5
Flow control

This chapter describes the message flow control for a C2C interface. It contains the following sections:

- B5.1 *Introduction*
- B5.2 *Message credits*
- B5.3 *Resource Planes*
- B5.5 *Ordered interface*

B5.1 Introduction

Messages are required to adhere to the following conditions:

- A message must not be dropped or lost in transport, that is, every message that is sent by a Transmitter must be received and processed by the Receiver.
- Incoming packets from the Link layer or Physical layer must be accepted without applying backpressure, in accordance with industry wide physical design standards. See UCIe [2] for more information.
- Response and Data channel messages must be accepted without dependency on forward progress of any Request or Snoop messages.
- Snoop channel message must progress without dependency on forward progress of Request channel messages.
- Write push requests must not block forward progress of any data that is not associated with a write push.

The following mechanisms are used to support the flow control requirements:

- Separate credits per message class are provided to ensure messages from an individual message class can progress, while other Message Class messages are back pressured.
- Multiple Resource Planes and DAT credit pools are supported within REQ and DAT Message Classes to support forward progress guarantees for different message groups within each message class.
- Ordering of messages within a message class and an RP is also supported.

B5.2 Message credits

To support flow control requirements, it is required that:

- Messages from each message class must be flow controlled independently.
- For each message class, the Receiver must provide message credits to the Transmitter. This ensures a Transmitter has the appropriate credits before sending a message.
  - Uncredited MISC messages do not require a message credit and the Receiver must guarantee to accept such a message.
  - To send a WrReqDataS or WrReqDataL message, the Transmitter needs to have one REQ and one DAT message credit. See B5.4 *Data credit pools*.

See Table B4.1 to determine which credit is required for each message type.

Credits can be granted using either the Protocol header or using a MiscU message.

B5.2.1 Credit grant using a Protocol header

Credits can be granted to the Transmitter using the message credit grant field, MsgCredit, in the Protocol Header. The subfields of the MsgCredit field are shown in Figure B5.1. The MsgCredit field can be used to grant credits for all message classes, except MISC.

A container with Activation messages cannot be used to grant credits using the Protocol header. See B8.2 *Interface Activation and Deactivation* for details on Activation messages.

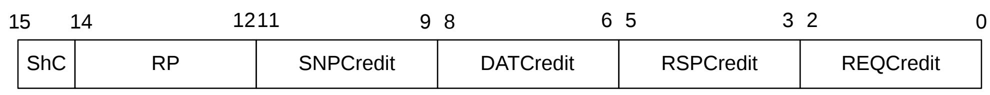

Figure B5.1: Message credit fields in Protocol Header

SharedCrdt (ShC) and RP fields are included to be used when multiple RP are supported in either REQ or DAT message classes. ShC and RP field values are applicable in both REQCredit and DATCredit fields, but are not applicable to RSPCredit or SNPCredit fields.

When ShC = 0 The REQCredit and DATCredit both correspond to credit pools indicated by the RP field value and:

- When WritePush_Support and DVMPush_Support are both False, DATCredit must be 0.
- When either WritePush_Support or DVMPush_Support, or both, are True, DATCredit must be 0 when the RP value is greater than 1.
- An RP value of greater than or equal value to the maximum REQ RP is not permitted.

When ShC = 1 The REQCredit and DATCredit both correspond to shared credit pool. The RP field is inapplicable and must be zero.

See B5.3 *Resource Planes* and B5.4 *Data credit pools* for details of the shared credit and dedicated credit pool usage.

B5.2.2 Credit grant using a MiscU message

Credits for all message channels can be granted using a MISC channel message by sending an explicit credit grant message on the MISC channel. See B5.2.3 *Credit grant message format*.

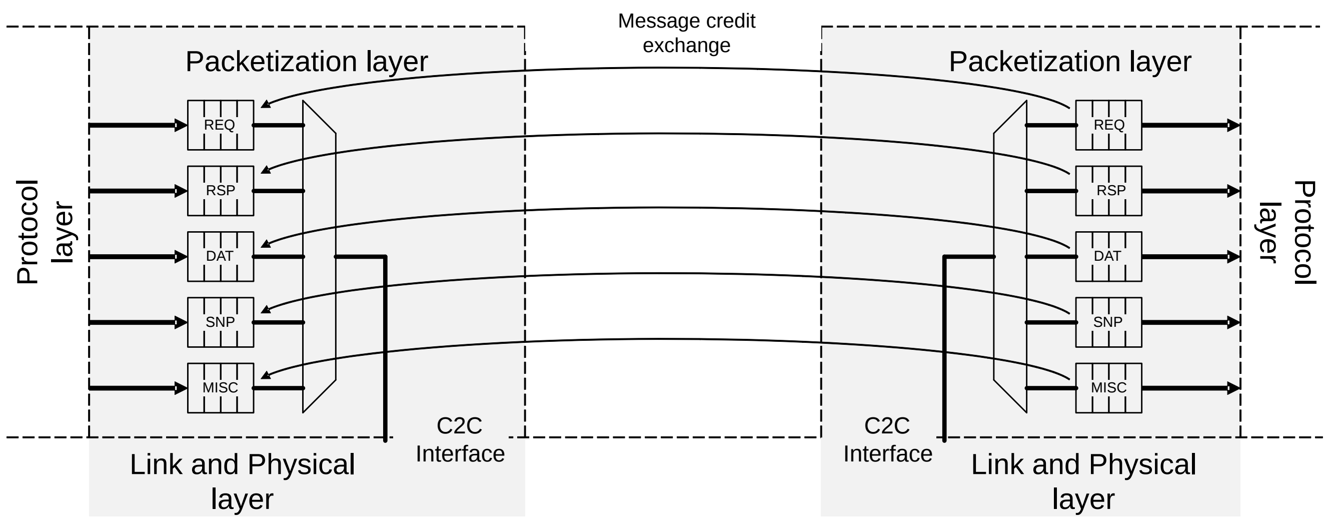

Figure B5.2: Message credits

Granted credits are used when a message in the associated message class is sent.

Table B5.1 shows the encoding of the message credit field.

Table B5.1: Message credit field encoding

Message credit[2:0]	Num credit
`000`	0
`001`	1
`010`	2
`011`	4
`100`	8
`101`	16
`110`	Reserved
`111`	Reserved

B5.2.3 Credit grant message format

The MiscU.CrdtGrant is used for granting credits for messages.

Table B5.2 shows the CrdtGrant MiscU message fields.

Table B5.2: CrdtGrant message fields

Field name	Width (bits)
MsgType	4
MiscOp	4
REQShCredit	3
RSPCredit	3
DATShCredit	3
SNPCredit	3
MISCCredit	3
REQ0Credit	3
REQ1Credit	3
REQ2Credit	3
REQ3Credit	3
REQ4Credit	3
REQ5Credit	3
REQ6Credit	3
REQ7Credit	3
DAT0Credit[a]	3
DAT1Credit[a]	3
RsvdZero	27
Total (bits)	80

[a] When WritePush_Support and DVMPush_Support are both False, this field is inapplicable and must be zero.

B5.3 Resource Planes

Multiple resource planes are typically used to enable:

- Forward progress guarantees against different requests. For example, to ensure the forward progress of PCIe posted writes is not dependent on the progress of other write transactions.
- Service time guarantees. For example, messages that are part of real-time traffic which need to be serviced within required latency bounds.
- Simplification of the protocol. For example, eliminating Request retry without creating a transport deadlock, even when two phases of a transaction use the same transport.

This specification supports request *Resource Planes* (RP) to provide transport independence between requests. When using Resource Planes:

- Messages using different Resource Planes must not block one another from the local Protocol layer to the remote Protocol layer.
- The following types of credits can be issued:
  - Dedicated credits are for a specific RP and can only be used for messages belonging to that RP.
  - Shared credits do not specify an RP and can be used for messages assigned to any RP.
- Messages using the same RP must remain in-order, even if they are using shared credits.
- Any RP for which the Transmitter has credits can be used to send messages.
- It is recommended, but not required, that a Transmitter uses shared credits before dedicated credits.

There can be up to eight Resource Planes for Request messages.

A Receiver can divide its buffers between shared and dedicated credit pools. The number of message credits for each dedicated credit pool is permitted to be different.

The Receiver must have available at least one dedicated credit per request RP supported and at least one credit for shared credit pool.

A Request message must:

- Indicate if it is using shared pool credit.
- Include the RP to which it belongs to, even when using shared pool credit.

SharedCrdt and ResPlane fields are included in request messages to support Resource Planes:

- SharedCrdt. Indicates if a request is using a shared credit.

When SharedCrdt = 0	The request is using dedicated RP credit. The RP number is indicated by the ResPlane field value.
When SharedCrdt = 1	The request is using shared pool credit. The RP to which the message belongs must be indicated by the ResPlane field value, even though shared credits are being used.

- ResPlane[2:0]. Indicates the RP to which the request belongs.

  The field value can be 0 to Num_RP_REQ. The Num_RP_REQ property defines the maximum number of dedicated RP supported by the Request message class. See Table B10.7 and Table B10.8 for the Num_RP_REQ encoding.

See B4.2.2 *Request message fields* for the ReqS and ReqL message format.

B5.4 Data credit pools

Credits that are used to transfer messages with data, DAT credits, are split into the following credit pools:

- DAT0Credit, a dedicated credit used to transfer Data messages that need progress to be guaranteed over any blocked WritePush message. The sending of a message using DAT0Credit must not be dependent on the progress of messages using credits from the other data credit pools.
- DAT1Credit, a dedicated credit used to transfer WrReqData messages that need guaranteed forward progress. The sending of a message using DAT1Credit must not be dependent on progress of messages using credits from the other data credit pools
- DATShCredit, a shared credit that can be used to transfer any data message. The use of DATShCredit by messages relating to one request RP must not block or be blocked by messages relating to another request RP.

Table B5.3 shows which message types are permitted to use which credit types.

Table B5.3: Data credit options for each message type

Message type	Messages	Credit type
Data	Read response with data, Snoop response with data, Write pull data response.	DATShCredit, DAT0Credit
WrReqData	WritePush that require forward progress guarantees.	DATShCredit, DAT1Credit
	WritePush that do not require forward progress guarantees.	DATShCredit

The choice of using a dedicated or shared credit can depend on the required progress of the transfer. For example:

- A PCIe write which must make forward progress, and is using the WrReqData message, is expected to use a dedicated credit when no shared credits are available.
- A DVM message which can be stalled by other messages can wait for a shared credit to be available and does not need dedicated credit support.

At least one credit of DATShCredit type must be available.

If either WritePush_Support or DVMPush_Support are True, at least one credit of each dedicated DAT credit type must be made available.

If both WritePush_Support and DVMPush_Support are False, DAT0Credit and DAT1Credit are not required and credits for them must not be sent.

B5.4.1 Credits for DataS and DataL messages

The DataS and DataL fields include a SharedCrdt field with the following encoding:

When SharedCrdt = 0	The data message is using a DAT0Credit.
When SharedCrdt = 1	The data message is using a DATShCredit.

See B4.2.4 *Data message fields* for the DataS and DataL message format.

B5.4.2 Credits for WrReqData messages

A WrReqData message includes a request and write data, so needs one REQ message credit and one DAT message credit.

The request part of a WrReqData message can be allocated an RP in the same way as other request messages.

Any WrReqData message that:

- Requires forward progress guarantees is permitted to use either DATShCredit or DAT1Credit. It is recommended, but not required, to use DATShCredit before DAT1Credit.
- Does not require forward progress guarantees must use DATShCredit.

The credit and RP options for WrReqData messages are encoded in SharedCrdt and ResPlane fields as follows:

- SharedCrdt

When SharedCrdt = 0	The request is using a REQxCredit. The data is using a DAT1Credit.
When SharedCrdt = 1	The request is using a REQShCredit. The data is using a DATShCredit.

- ResPlane[2:0]

  Indicates the RP to which the request is assigned.

See B4.2.5.2 *Message fields* for the WrReqData message format.

Note

A write message that needs both Request and Data shared credits to progress using WritePush semantics may change to using Write pull to progress on the transaction, when simultaneous availability of both shared credits is not provided in a timely manner.

B5.5 Ordered interface

It is required that the C2C interface must be ordered.

Containers must be received at the C2C interface in the same order that they were sent by the C2C interface on the far side of the link.

It is required that:

- At the Transmitter:
  - Messages from the same message class and same RP where that applies, must be placed in the container in the same order that they are received from the Protocol layer, that is in granules starting from the lowest number first to higher numbers.
  - Messages from different message class do not have such ordered placement restrictions.
- At the Receiver:
  - The Packetization layer must maintain the order of placement of messages of a given message class within the received container as well as order in which containers are received.
  - The messages within a message class of the same RP value must be forwarded to the Protocol layer in the same order as they are received.

See B2.4 *Multiple interfaces* for ordering requirements when using multiple interfaces between two components.

The ordered interface is a required feature and hence an interface property to control its presence is not included.

Chapter B6
DVM transactions

This chapter contains the following sections:

B6.1 Introduction

All CHI DVM transactions are supported over the C2C interface. A new node type, *DVM Node* (DN), is defined which is responsible for propagating local DVM transactions to remote chips and performing the required local invalidations as requested by remote DVM requests.

DVM Nodes also manage flow control to guarantee DVM transaction forward progress.

DVM message flows are optimized by introducing new response types and reducing the number of messages.

B6.2 DVM Node, DN

A DN processes DVM operations to and from remote chips. A DN performs the following functions:

- Sends DVM requests to remote DVM Nodes for local DVM transactions. When sending the request to multiple nodes, it may use unicasting DVM requests to each remote DN or a hierarchical tree communication pattern to send to a subset of remote DN which further forward the request to another subset and thus spanning the DVM request to all DVM nodes in the system.
- Collects responses for DVM requests it sent to remote DN.
- Performs locally required invalidations for remote DVM requests.
- Coordinates interaction between local and remote DVM transactions.

Each chip is permitted to have one DN only.

The number of DVM Nodes in the system is limited to 64.

B6.2.1 DVM Node assignment

A DN is permitted to share ID name space with Request Nodes and Home Nodes on the same chip.

This requires DVM responses which are targeting a DN to be identified separately from responses targeting Request Nodes.

Note

The ability to share Node ID name space minimizes the total unique ID required in the system.

B6.3 DVM transaction flows

The two DVM transactions over C2C are:

- DVMReq
- DVMSync

In this specification, DVMReq refers to DVMOp(NonSync) and DVMSync refers to DVMOp(Sync).

B6.3.1 DVMReq

The DVMReq transaction flow is illustrated in Figure B6.1. The flow is similar to a 32-byte write partial transaction flow. The DVMReq transaction can use write push or write pull semantics.

The DVMReq is permitted to use write push semantics only when DVMPush_Support property is True.

The Completer responses are:

- When a write push flow is used: CompDVM.
- When a write pull flow is used: CompDVM, DBIDRespDVM, or a combined CompDBIDRespDVM.

See B6.4 *DVM transaction responses* for response message details.

DVM transaction payload is distributed across the Addr and Data fields of the request, like on-chip CHI. Comp and DVMData are differentiated from Comp and Data responses for non-DVM transactions by using different opcodes.

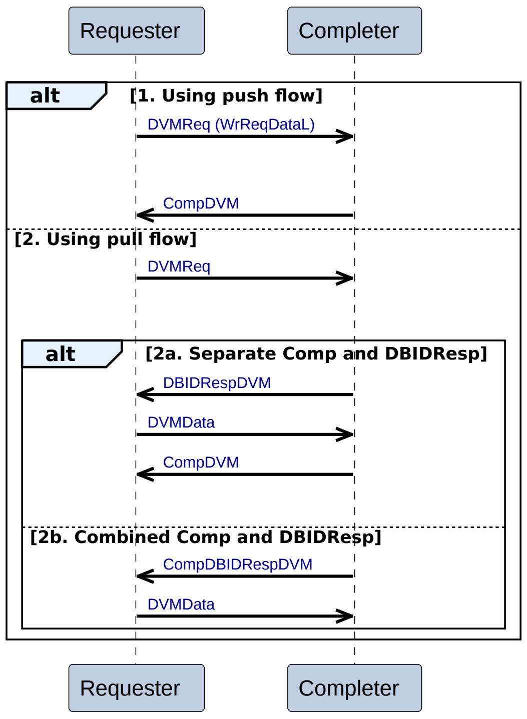

Figure B6.1: DVMReq transaction flow

Note

On-chip CHI is used to pass DVM transactions from a local DN or ingress port to the C2C Requester.

After being sent across the C2C interface, the C2C Completer uses on-chip CHI to pass the transaction to the local DN or an egress port.

B6.3.2 DVMSync

The DVMSync transaction flow is illustrated in Figure B6.2. The flow is similar to a Dataless request transaction flow with a single response.

The complete response is CompSyncDVM.

All required payload is in the request address fields and thus the DVM transaction data response is eliminated.

DVMSync is used when the on-chip Opcode field indicates DVMOp and the DVMType subfield indicates Synchronization.

Request is tracked by SrcID alone. There can be only one DVMSync transaction outstanding from each source. Thus, the TgtID in the response is used to match the response to its corresponding request.

In the request, the TxnID is inapplicable and must be zero.

In the response, the TxnID field is inapplicable and must be zero.

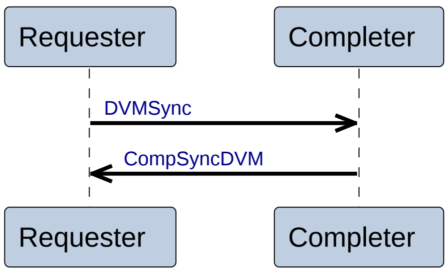

Figure B6.2: DVMSync transaction flow

B6.4 DVM transaction responses

The response messages for DVMReq and DVMSync messages are:

- For DVMReq: CompDVM, DBIDRespDVM, CompDBIDRespDVM, and DVMData
- For DVMSync: CompSyncDVM.

See Table B6.1 and Table B6.2 for opcodes for DVM transaction responses, both data and non-data responses.

Table B6.1: DVM transaction response opcode encodings

Opcode[4:0]	Resp
`0x1C`	CompDVM
`0x1D`	DBIDRespDVM
`0x1E`	CompDBIDRespDVM
`0x1F`	CompSyncDVM

Table B6.2: DVMReq data response opcode encodings

Opcode[4:0]	Resp
`0xD`	DVMData

B6.5 Flow control

The following rules must be followed by the Requester and the Completer of DVM transactions to guarantee transaction forward progress is made.

A DN:

- Must have only one DVMSync request outstanding to remote DVM nodes.
- Permitted to have any number of DVMReq requests outstanding to remote DVM nodes.
- Must accept one DVMSync from each remote DN.

This requirement of all DVMSync requests must be sunk at the destination DN is placed to avoid request path being blocked at the C2C.

A DVMSync at DN must not block forward progress of DVMReq transactions. That is, must guarantee forward progress of DVMReq transactions received from remote nodes. This guarantee can be obtained by reserving a buffer for received DVMReq transactions.

DVMReq transactions forward progress at the C2C interface must not depend on forward progress of DVMSync transactions.

Chapter B7
RME-DA and RME-CDA support

This chapter describes the C2C support for RME *Device Assignment* (DA) and RME *Coherent Device Assignment* (CDA). which are part of Arm *Confidential Compute Architecture*. It contains the following sections:

B7.1 Introduction

RME-DA is part of the *Realm Management Extension* architecture that enables the secure assignment of assignable device interfaces. A *Device Permission Table* (DPT) contains the permission attributes associated with physical addresses and specifies a corresponding set of DPT checks that apply to incoming accesses from devices.

In this section, the term RME-DA refers to a device that is IO coherent and the term RME-CDA refers to a device that has the capability of being fully coherent.

This specification supports RME-DA and RME-CDA in multiple accelerator device topologies, where the accelerators are connected to a host. See Figure B7.1.

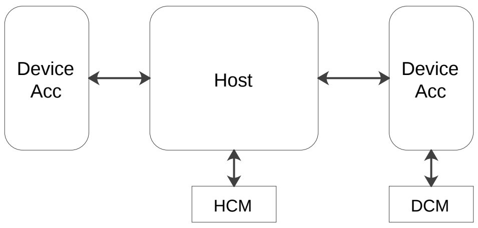

Figure B7.1: A host connected to two Accelerator devices

A host, made from a single or multiple SoC dies, forms a *Trusted Computing Base* (TCB): A host:

- Is responsible for enforcing the security policies required by the system.
- Can be connected to multiple devices.
- Can have memory termed as *Host Coherent Memory* (HCM) directly attached to it. The HCM can be coherently accessed by the host, or by any fully coherent or IO coherent devices.
- Must not send a Stash Snoop to a device, unless the line is already legally permitted to be cached by the device.
- Is permitted to send DVMOp requests or SnpDVMOp snoops towards a device.

A device is a single physical chip(let) and is treated as a single unit from a trust perspective and individually attested. A device:

- Is trusted in its entirety for a set of Realms only and trusted to maintain Realm separation where required.
- Can have private memory-mapped resources.
- Can have one or more interfaces.
- Can have memory termed as *device coherent memory* (DCM) directly attached to it. The DCM can be coherently accessed by the host, the device, and by other coherent devices.
- Can access the DCM of another coherent device only where that access is channeled through a host compute node.
- Only operates in the Realm and Non-secure PAS.
- Is expected, but not required, to include the following fields in all requests to the host:
  - *Security State* (SecSID1) field
  - *Stream ID* (StreamID) field

- Only receives snoop requests for regions of memory owned by Realms that trust the device.
- Cannot be directly connected to another device.
- Is not permitted to send DVMReq or DVMSync requests to a host.

The host and the devices can have multiple Request Nodes and Home Nodes.

It is expected that the host will perform additional checks on certain messages from or to a device. These checks can ensure that:

- Requests coming from the device to HCM are permitted to access that memory. This means the device cannot issue a request with a random PA and expect to get access to that location.
- Snoops coming from the device are only for DCM. This means the device cannot issue a request with arbitrary PA to gain access to a location.
- Snoops going to the device for cached HCM locations only expose accesses that the device is permitted to observe. This means the device cannot build up a picture of all trusted system activity for more targeted attacks.

For more information on RME-DA and RME-CDA, see [1].

B7.2 Fields supporting RME-DA and RME-CDA

The fields used to enable support for RME-DA and RME-CDA include:

StreamID Stream Identifier. StreamID is a unique identifier of request streams originating from one or a set of Requesters associated with the same System MMU context. The StreamID field in messages at an interface is applicable as defined by the DevAssign_Support properties.

> **Note**
>
> When interfacing to PCIe, StreamID is the Requester ID associated with a single PCIe endpoint.

SecSID1 Secure Stream Identifier. SecSID1 qualifies the Security state of StreamID. The SecSID1 field in messages at an interface is applicable as defined by the DevAssign_Support properties.

MECID Memory Encryption Context Identifier. MECID is an identifier assigned to memory accesses, associating each access with a memory encryption context. The MECID value is unique to each Realm. The MECID field in messages at an interface is applicable as defined by the MEC_Support properties.

For more details on these fields and their applicability, see [1].

B7.3 Host and device responsibilities

For details on the responsibilities of a host or device in a system that supports RME-DA or RME-CDA, see [4] and [5].

B7.4 Request Nodes per device

In an accelerator-attached configuration when the DevAssign_Support_Tx property indicates an interface is represented as a device:

- At the interface, the maximum number of Request Nodes permitted is 16. From this, a maximum of 8 nodes can be RN-F. The remaining can be IO coherent Request Nodes. The RN-F nodes must be assigned node ID values within the range 0 to 7.
- The interface must precisely track device caching to determine if an RN-F on that device has cached a copy of the line.

Note

The manner in which memory caching is tracked at a Home determines the efficiency of snoop filter implementation.

Precisely knowing where a line is cached results in a Home sending targeted snoops to that cache.

With coarse grain tracking, a Home needs to resort to multi-casting the snoop to a subset of caches or broadcasting the snoops to all the caches except the Requester. Both multicasting and broadcasting of snoops involves a Home sending multiple unicast snoops.

A fine grain tracker at Home, typically a snoop filter or a directory, increases in size with respect to the increase in the number of caches to be tracked precisely.

To manage the increase in the size of a fine grain tracking snoop filter, an architectural upper limit on the number of caching agents per interface is placed. A host that supports RME-CDA can preemptively size its cache tracking structure to that limit.

Chapter B8
Interface state management

This chapter describes the C2C interface state management. It contains the following sections:

B8.1 Introduction

Hardware flows manage the connectivity state of the interface and thus the link.

The three different connect and disconnect set of flows affect either the fully coherent traffic, DVM messages or all protocol messages.

B8.2 Interface Activation and Deactivation

Interface activation and deactivation is used to enable or disable transport of all protocol traffic across the interface. Activation messages within MiscU message type support interface activation and deactivation. See B8.2.2 *Messages* for details of these messages.

B8.2.1 Activity states

The interface can be in one of four activity states.

STOP	The interface is disconnected from all protocol activity.
ACTIVATE	The interface is preparing to enable protocol activity.
RUN	The interface is enabled for protocol activity.
DEACTIVATE	The interface is preparing to disconnect from all protocol activity.

See B8.2.4 *Activation and Deactivation flow* for details of the interface behavior in each state.

The permitted state transitions are:

- STOP to ACTIVATE
- ACTIVATE to RUN
- RUN to DEACTIVATE
- DEACTIVATE to STOP

Figure B8.1 shows the permitted state transitions.

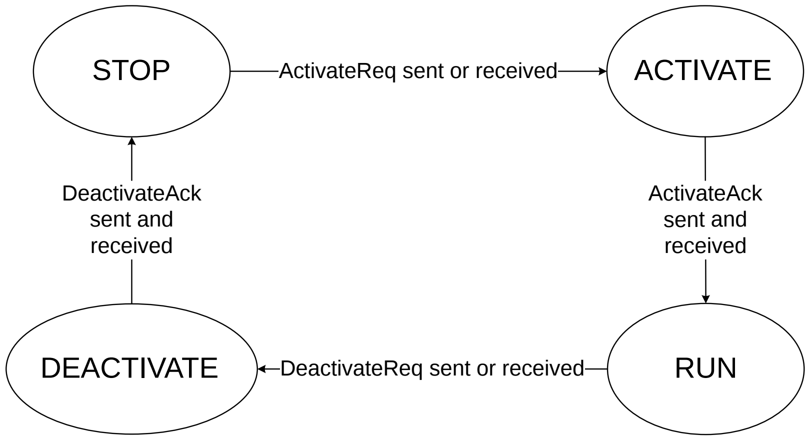
STOP
ActivateReq sent or received
ACTIVATE
DeactivateAck sent and received
ActivateAck sent and received
DEACTIVATE
DeactivateReq sent or received
RUN

Figure B8.1: Activation deactivation state transition

The remote credits are implicitly returned when the interface moves into the STOP state. That is, all Transmitter message credit counts are zeroed and Receiver message credits are reset to the values they were at the interface initialization.

B8.2.2 Messages

The following five messages are defined to support interface activation flow:

- ActivateReq
- ActivateAck
- DeactivateReq

- DeactivateAck
- DeactivateHint

See Table B8.2 for their message format and Table B8.1 for ActivationOp encoding.

B8.2.2.1 ActivateReq and ActivateAck

ActivateReq and ActivateAck message pair is used to move the interface from a STOP state to the RUN state.

See STOP and ACTIVATE state entries in B8.2.4 *Activation and Deactivation flow* for when ActivateReq and ActivateAck messages are sent and the interface response on receiving them.

B8.2.2.2 DeactivateReq and DeactivateAck

DeactivateReq and DeactivateAck message pair is used to bring the interface from the RUN state to the STOP state.

B8.2.2.3 DeactivateHint

DeactivateHint is a message informing the remote interface about the readiness of the Transmitter to deactivate the interface. The characteristics of this message include:

- It is a MiscU.Activation message. See Table B8.1.
- An interface is only permitted to send this message in the RUN activation state.
- An interface is permitted to send a DeactivateHint and then resume activity on the link.
- An interface is permitted to send multiple DeactivateHint messages, either with or without link activity between them.
- An interface is permitted to send a DeactivateReq without first sending a DeactivateHint message.
- If an interface has sent a DeactivateHint it is permitted to send a DeactivateReq without waiting for any response to the DeactivateHint.

> **Note**
>
> The sender may use heuristics to decide on when to send the DeactivateHint message. For example, the heuristics might use the length of time the link has been inactive, that is, the time period for which no message is sent or received across the link.

- The Receiver is permitted to respond to the DeactivateHint message in the following manner:
  - Can drop the hint without any state change.
  - Accept the hint and respond with DeactivateReq when the conditions for sending the message are met.

B8.2.3 Activation message format

Table B8.1 shows the encoding of ActivationOp field in the Activation message.

Table B8.1: Encoding of ActivationOp field in MiscU.Activation messages

ActivationOp[3:0]	Message
`0000`	ActivateReq

Continued on next page

Table B8.1 – *Continued from previous page*

ActivationOp[3:0]	Message
`0001`	ActivateAck
`0010`	DeactivateReq
`0011`	DeactivateAck
`0100`	DeactivateHint
Others	Reserved

See B8.2.2 *Messages* for details on activation messages.

Table B8.2 shows the Activation message fields.

Table B8.2: Activation message fields

Field name	Width (bits)	Value
MsgType	4	`0b0000`
MiscOp	4	`0b0010`
ActivationOp	4	
PropertyReq	1	`0` or `1`[a]
RsvdZero	19	
Total (bits)	32	

[a] When ActivationOp == ActivationReq, else inapplicable and must be 0.

B8.2.3.1 PropertyReq

This field value determines if a property exchange is required at the completion of interface activation.

When PropertyReq = 0	Properties must not be exchanged.
When PropertyReq = 1	Properties messages must be sent after ActivateAck is sent and received.

This field is applicable when ActivationOp is ActivateReq.

This field is inapplicable and must be zero for other Activate message types.

See B10.1 *Introduction*.

B8.2.4 Activation and Deactivation flow

This section describes the behavior of the interface in each activation state.

There are two options for conditions under which the interface deactivation can be initiated. The two deactivation conditions, defined by Deactivation_Support property, are:

1. Protocol agnostic: DeactivateReq is permitted to be sent from the RUN state at any time and does not need to check protocol message flows. Additionally, the protocol messages must not be sent after DeactivateReq is sent until the interface comes back to RUN state.
2. Protocol aware: DeactivateReq is permitted to be sent only when interfaces are in the RUN state and are completely quiesced in both directions. A reason for quiescing the interface may be for powering down the interface or the chip. Quiesce condition requires both sides must not have any protocol traffic to send. That is, all outstanding requests must have completed, no response messages must be sent, and no new request or snoop messages must be sent across the link.

The interface behavior for each activation and deactivation state includes:

STOP

- The credits are reset in this state.
- The Receiver has all the credits but must not send any credits.
- The Transmitter has no credits and is only permitted to send LinkStatus or ActivateReq messages.
- If an interface wants to start sending other messages, it must send an ActivateReq message.
- When an interface has sent or received an ActivateReq message, it must move into the ACTIVATE state.

ACTIVATE

- If not already sent, send an ActivateReq message.
- If not already received, wait for an ActivateReq message.
- After receiving the ActivateReq message, ActivateAck must be sent in a timely manner in response to the request.
- After ActivateAck is sent, it is permitted, but not required, to send credits.
- After ActivateAck is received, credits may be received.
- When an interface has sent and received an ActivateAck message, it must move into the RUN state.

RUN

- Interface properties are sent, if requested.
- Credits, appropriate to the negotiated interface properties, must be sent.
- Credited messages can be sent.
- If an interface might want to stop, it can send a DeactivateHint message. This message can be ignored by the remote or can cause it to send a DeactivateReq message.
- If an interface wants to move to the STOP state, it can send a DeactivateReq message.
- When an interface has sent or received a DeactivateReq message, it must move into the DEACTIVATE state.

DEACTIVATE

- If not already sent, when required activity is completed, send a DeactivateReq message.
- Credits must be sent, until DeactivateReq is received.

- Credited messages must not be sent, after the DeactivateReq message is sent.
- DeactivateAck must be sent in a timely manner after a DeactivateReq is received.
- Credits must not be sent after the DeactivateAck message is sent.
- When an interface has sent and received a DeactivateAck message, it must move into the STOP state.

B8.2.4.1 Activation trigger, ActTrigger

A software visible 2-bit activation control field, ActTrigger, is defined to enable the hardware to initiate an Activate or Deactivate flow.

Table B8.3 shows the encoding of the ActTrigger field.

Table B8.3: ActTrigger bit encoding

ActTrigger[1:0]	Description
`00`	No action
`01`	Send ActivateReq
`10`	Send DeactivateReq
`11`	Reserved

It is sufficient to set the trigger bits on one side to start the activation or deactivation sequence.

For example, if an Accelerator is attached to a host using the C2C link, the triggers can be implemented only on the host side.

B8.3 Connect Miscellaneous messages

The MiscU.Connect messages are used in connecting and disconnecting an interface from the Snoop and DVM domains. See B8.4 *Coherency connect and disconnect* and B8.5 *DVM domain connect and disconnect* on how these messages are used.

Table B8.4 shows the Connect message fields.

Table B8.4: MiscU.Connect message fields

Field name	Width (bits)	Value
MsgType	4	`0b0000`
MiscOp	4	`0b0011`
ConnectOp	4	
RsvdZero	20	
Total (bits)	32	

Table B8.5 shows the encoding of ConnectOp field in Connect messages.

Table B8.5: Encoding of ConnectOp field in MiscU.Connect messages

ConnectOp[3:0]	Message
`0000`	CohConnectReq
`0001`	CohConnectAck
`0010`	CohDisconnectReq
`0011`	CohDisconnectAck
`0100`	DVMConnectReq
`0101`	DVMConnectAck
`0110`	DVMDisconnectReq
`0111`	DVMDisconnectAck
Others	Reserved

B8.4 Coherency connect and disconnect

Interface coherency connect and disconnect messages are used to connect or disconnect the interface from coherency management requirements.

B8.4.1 Coherency connect states

Each direction of C2C interface can be in one of the four coherency states. See Table B8.6 for details of each state.

CohDisabled	The Requester caches on the interface do not include coherent data.
CohConnect	An interface is requesting to be permitted to cache coherent data.
CohEnabled	The Requesters on the interface participate in coherent data exchange.
CohDisconnect	The Requesters on the interface are requesting to be removed from participating in coherent data exchange.

The Requesters on a chip participating in coherency domain are independent of the Home Nodes on that interface sending snoops to the remote nodes. Thus, the coherency connect and disconnect is per direction. The coherency state of an interconnect effects all Requesters on that interface.

The permitted state transitions are:

- CohDisabled to CohConnect
- CohConnect to CohEnabled
- CohEnabled to CohDisconnect
- CohDisconnect to CohDisabled

The interface moving from CohDisconnect to CohDisabled state must retain and not zero out the SNP credits held from the remote side.

Figure B8.2 shows the permitted state transitions.

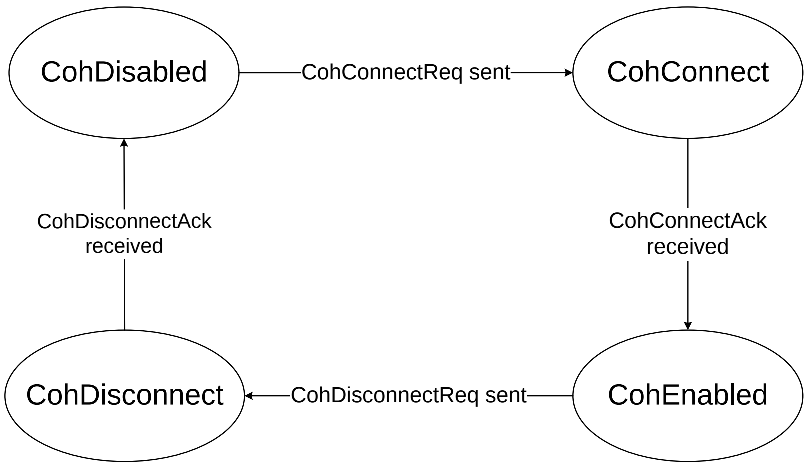

Figure B8.2: Coherency domain connect disconnect state transitions

B8.4.2 Messages

As part of MiscU.Connect message type, there are four messages defined to support interface Coherency connect flow:

1. CohConnectReq

2. CohConnectAck
3. CohDisconnectReq
4. CohDisconnectAck

See Table B8.4 for their message format and Table B8.5 for ConnectOp encoding.

Table B8.6 describes the behavior of the two interfaces during connecting and disconnecting to a coherency domain.

Table B8.6: C2C coherency connect state description

State	Tx C2C interface behavior	Rx C2C interface behavior
CohDisabled	Requesters at this interface must not contain coherent data. Must not send Snoopable Allocating requests. Does not: – Receive snoop requests from the remote. – Influence coherency connect state in the reverse direction. Permitted to grant SNP message credits to remote. Must remain in CohDisabled state until CohConnectReq is sent and then must move to CohConnect state.	Must not send snoop requests to remote. Must be able to handle receiving SNP message credits. Does not influence coherency connect state in the reverse direction. Must remain in CohDisabled state until CohConnectReq is received and then must move to CohConnect state.
CohConnect	Permitted to send SNP message credits. Must be prepared to respond to snoop requests. Must remain in CohConnect state until CohConnectAck is received and then must move to CohEnabled state.	Must be able to handle receiving SNP message credits. Must send CohConnectAck in a timely manner and then move to CohEnabled state.
CohEnabled	Permitted to send Snoopable Allocating requests. – Requesters at this interface are permitted to cache coherent data. Must respond in a coherence compliant manner to any received snoop requests. When desired to disconnect from coherency domain and the Requester caches on the interface do not include coherent data then send CohDisconnectReq and move to CohDisconnect state.	Permitted to send snoop requests as required to maintain coherency. Must remain in CohEnabled state until CohDisconnectReq message is received and then must move to CohDisconnect state.

Continued on next page

Table B8.6 – *Continued from previous page*

State	Tx C2C interface behavior	Rx C2C interface behavior
CohDisconnect	Must not send Snoopable Allocating requests. Must continue granting SNP credits. The Remote side might have snoop requests queued to be sent. Must continue to respond to snoop requests. Must remain in CohDisconnect state until it receives CohDisconnectAck and then must move to CohDisabled state.	Complete outstanding snoop requests. Must send CohDisconnectAck message in a timely manner and move to CohDisabled state.

Note

Coherence connect states are independent in each direction. An interface Transmitter may continue to send Snoop requests while the Receiver on the same side of the interface is in coherency disabled state.

B8.5 DVM domain connect and disconnect

The DVM domain connect and disconnect messages are used to connect and disconnect the interface from address translation tables management.

B8.5.1 DVM connect states

A C2C interface can be in one of four DVM domain connect states. See B8.5.3 *DVM domain connect and disconnect flow* for details of each state.

DVMDisabled	The interface is removed from DVM domain and does not send or receive DVM requests.
DVMConnect	An interface is requesting to join the DVM domain.
DVMEnabled	An interface is participating in DVM domain activities.
DVMDisconnect	An interface is requesting to be taken out of DVM domain.

When an interface removes itself from DVM domain, it neither sends or accepts DVM requests.

The permitted state transitions are:

- DVMDisabled to DVMConnect
- DVMConnect to DVMEnabled
- DVMEnabled to DVMDisconnect
- DVMDisconnect to DVMDisabled

Figure B8.3 shows the permitted state transitions.

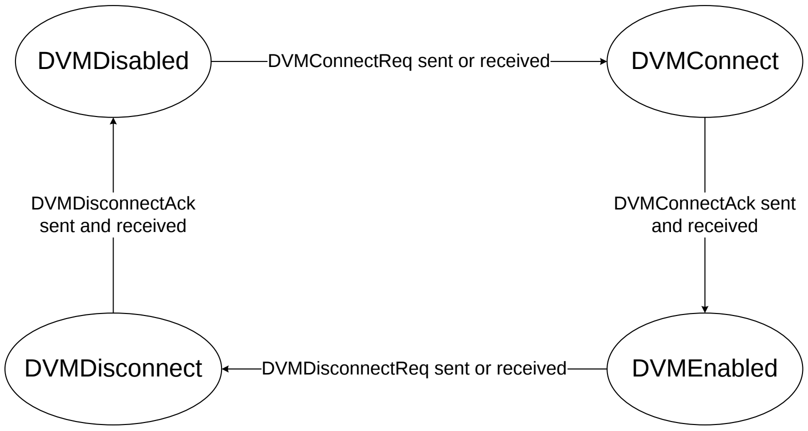
DVMDisabled
DVMConnectReq sent or received
DVMConnect
DVMDisconnectAck sent and received
DVMConnectAck sent and received
DVMDisconnect
DVMDisconnectReq sent or received
DVMEnabled

Figure B8.3: DVM domain connect disconnect state transitions

B8.5.2 Messages

The following four messages, grouped together as part of MiscU.Connect message type, are defined to support interface DVM connect flow:

- DVMConnectReq
- DVMConnectAck
- DVMDisconnectReq
- DVMDisconnectAck

See Table B8.4 for their message format and Table B8.5 for ConnectOp encoding.

B8.5.3 DVM domain connect and disconnect flow

The interface behavior in each DVM connect state includes:

DVMDisabled

- The Requesters on the interface must not cache address translations.
- The interface must not send or receive DVM transactions.
- The interface is permitted to receive a DVMConnectReq message.
- A DVMConnectReq message can be sent if the interface wants to participate in DVM domain activities.
- When an interface has sent or received a DVMConnectReq message, it must move into the DVMConnect state.

DVMConnect

- If not already sent, then send a DVMConnectReq.
- If not already received, wait for DVMConnectReq message.
- Once DVMConnectReq is received, send DVMConnectAck in a timely manner.
- Must remain in DVMConnect state until DVMConnectAck is sent and received and then move to DVMEnabled state.

DVMEnabled

- Permitted to send DVM transactions.
- Can receive DVM transactions.
- If the interface wants to be removed from the DVM domain, it must first complete all outstanding DVM transactions and then send DVMDisconnectReq.
- When an interface has sent or received a DVMDisconnectReq message, it must move into the DVMDisconnect state.

DVMDisconnect

- If not already sent, wait for all outstanding DVM transactions to complete then send a DVMDisconnectReq message.
- After sending a DVMDisconnectReq, must not send any DVM requests.
- Must continue to respond to DVM requests until a DVMDisconnectReq is received.
- Must send DVMDisconnectAck in a timely manner after DVMDisconnectReq is received.
- When an interface has sent and received DVMDisconnectAck message, it must move into the DVMDisabled state.

Chapter B9
Interface initialization

This chapter describes the C2C interface initialization. It contains the following sections:

B9.1 Introduction

C2C interface initialization establishes the interface to enable the sending and receiving of protocol messages.

The initialization flow has the following steps:

1. Coordinate with the Link layer to check if the remote Protocol layer is up within the interface.
2. Exchange interface properties using MiscU.Properties messages. This sets up the interface to start exchanging message credits followed by protocol messages.

Note

Discovery, enumeration, and negotiation of global or common capabilities, such as Req_Addr_Width, Node ID assignment, address maps, routing table set up, is not managed by the hardware property exchange flows.

The exchange of capabilities that are not part of the hardware property exchange flow is expected to be handled by software.

B9.2 Initialization flow

The key features of the initialization flow include:

- The Protocol layers on the two sides of the link are guaranteed to be able to receive Miscellaneous messages before link initialization is completed. This enables the Protocol layers to exchange MiscU.Activation messages using the default container format.

Figure B9.1 shows the possible transaction flows for interface initialization.

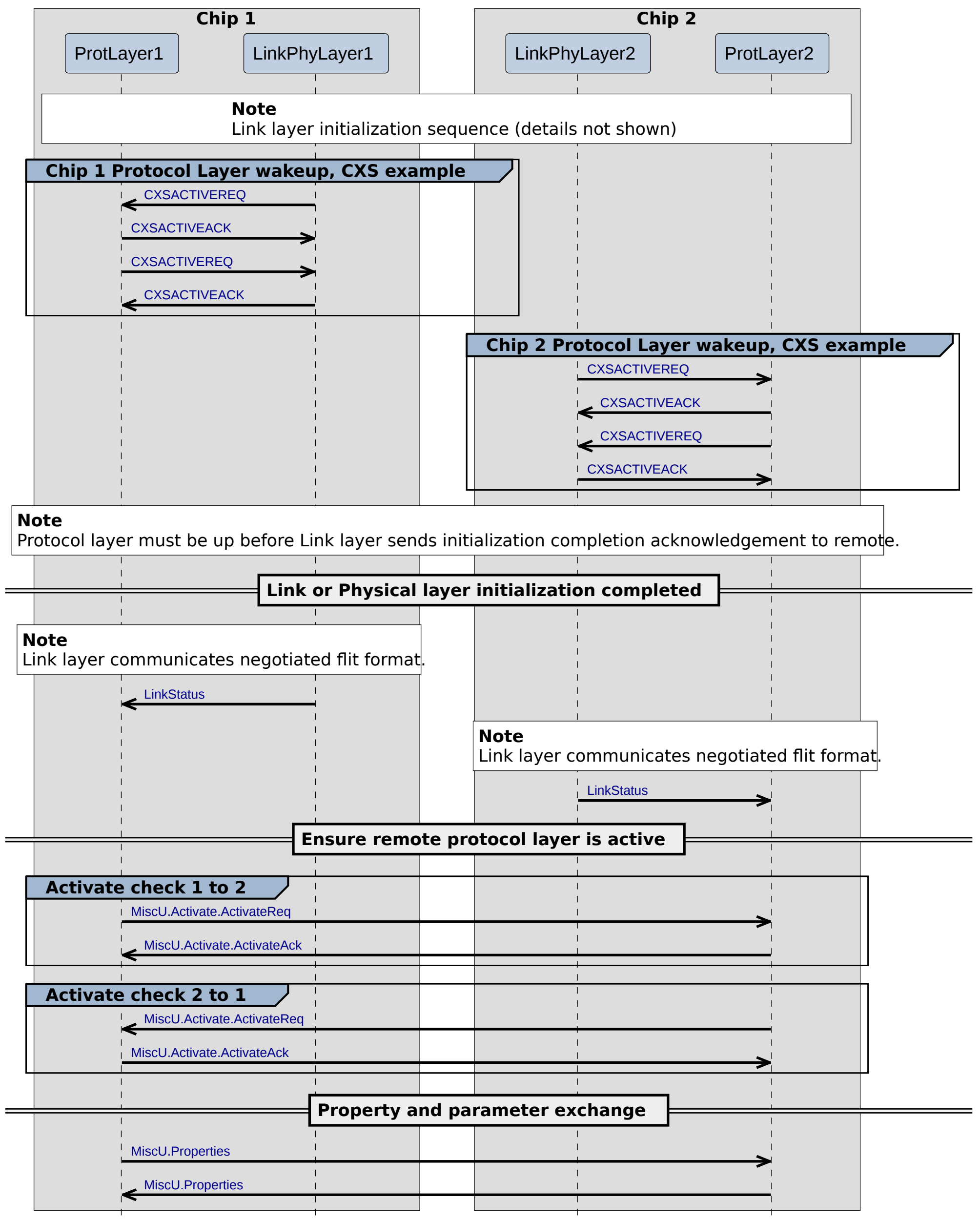
Chip 1
Chip 2
ProtLayer1
LinkPhyLayer1
LinkPhyLayer2
ProtLayer2
Note
Link layer initialization sequence (details not shown)
Chip 1 Protocol Layer wakeup, CXS example
CXSACTIVEREQ
CXSACTIVEACK
CXSACTIVEREQ
CXSACTIVEACK
Chip 2 Protocol Layer wakeup, CXS example
CXSACTIVEREQ
CXSACTIVEACK
CXSACTIVEREQ
CXSACTIVEACK
Note
Protocol layer must be up before Link layer sends initialization completion acknowledgement to remote.
Link or Physical layer initialization completed
Note
Link layer communicates negotiated flit format.
LinkStatus
Note
Link layer communicates negotiated flit format.
LinkStatus
Ensure remote protocol layer is active
Activate check 1 to 2
MiscU.Activate.ActivateReq
MiscU.Activate.ActivateAck
Activate check 2 to 1
MiscU.Activate.ActivateReq
MiscU.Activate.ActivateAck
Property and parameter exchange
MiscU.Properties
MiscU.Properties

Figure B9.1: C2C link initialization

The sequence for interface initialization is:

1. The interface initialization starts with link reset and initialization.

Before the link initialization is complete, the Link layer must wake up the Protocol layer. The Link layer to Protocol layer wake-up handshake method is IMPLEMENTATION DEFINED.

An example wake-up handshake using CXS interface between the Link layer and Protocol layer is shown in Figure B9.1.

- The Link layer sends **CXSACTIVEREQ** to the local Protocol layer.
- Once up, the Protocol layer sends the **CXSACTIVEACK** response.
- In response to **CXSACTIVEREQ** from the Link layer, the protocol goes through the same handshake in the reverse direction. This reverse-direction **CXSACTIVE** handshake can be performed as late as during the end-to-end, that is, PL-to-remote-PL Activate handshake.

2. After the Link layer initialization is complete, the Link layer must be able to receive and recognize ActivateReq from the remote side. This is to be able to receive the ActivateReq request message, as described in (4).
3. After the Link layer initialization is complete, the Link layer sends MiscU.LinkStatus message to the local Protocol layer.

   The LinkStatus message informs the Protocol layer of the completion of link initialization and includes information on negotiated flit format. For LinkStatus message details, see B9.2.1 *LinkStatus message format*.
4. Protocol layer to ensure that the remote Protocol layer is active:
   - Each Protocol layer sends ActivateReq message to the remote Protocol layer.
   - The Protocol layer responds to received ActivateReq with ActivateAck message.
5. At this stage, both sides are ready to exchange supported property values using Properties message, if requested in the ActivateReq:
   - The Protocol layer sends Properties message to the remote.
   - Once the Protocol layer receives Properties messages from the remote, the Protocol layer is permitted to grant credits to the remote.

For details of interface properties, their exchange and negotiation see Chapter B10 *Protocol layer property exchange*.

At the time when the link activation phase is completed, and properties exchange is required, the interface Coherency and DVM states are in Disabled state. If required, the interface coherency connection and DVM connection must be enabled by going through individual Coherency and DVM connect flows.

B9.2.1 LinkStatus message format

The MiscU.LinkStatus message is used by the Link layer to inform the Protocol layer the completion of Link layer initialization and change in power status of the link. This message is sent during the interface initialization as described in B9.2 *Initialization flow* and the Link layer is permitted to send whenever the physical link power state changes.

Table B9.1 shows the LinkStatus message fields.

Table B9.1: LinkStatus message fields

Field name	Width (bits)	Value
MsgType	4	`0b0000`
MiscOp	4	`0b0110`

Continued on next page

Table B9.1 – *Continued from previous page*

Field name	Width (bits)	Value
FlitFormat	3	
LinkPowerState	3	
RsvdZero	18	
Total	32	

B9.2.1.1 FlitFormat

The FlitFormat field in the LinkStatus message is used by the Link layer to inform the Protocol layer how many bytes are required to populate the Link layer specific fields, such as link header, FEC, and CRC.

Table B9.2 shows the mapping of the FlitFormat values to the Protocol layer container format.

Table B9.2: Encoding of FlitFormat in LinkStatus message

FlitFormat[2:0]	Bytes required	Comment
`000`	Reserved	
`001`	6	Compatible with Format X
`010`	20	Compatible with Format Y
Others	Reserved	

B9.2.1.2 LinkPowerState

The LinkPowerState field in the LinkStatus message allows the Link layer to inform the protocol of the power state of the link. This information is sent to the Protocol layer during link initialization and also whenever there is a change in link power state.

Table B9.3 shows the LinkPowerStatus message fields.

Table B9.3: Encoding of LinkPowerState in LinkStatus message

LinkPowerState[2:0]	Power state
`000`	Disabled
`001`	Active
`100`	Reset
Others	Reserved

Note

The Protocol layer upon receiving LinkStatus message with Reset may be required to reset some protocol configuration settings.

Chapter B10
Protocol layer property exchange

This chapter describes the exchange of properties for the Protocol layer. It contains the following sections:

B10.1 Introduction

After the interface initialization is completed and the activation sequence requires a property exchange, interface properties must be communicated in both directions for each logical link before any protocol messages are sent.

A property is used to declare a capability.

The interface properties are grouped into three classifications:

1. On-chip payload
    - Details either the presence or width, or both, of certain on-chip fields.
    - Allows for the optimization of the payload fields prior to the message selection, potentially allowing for more efficient packing.
2. On-chip opcode/flow
    - Details the on-chip support of certain classifications of transactions.
    - Ensures that the interface does not deadlock by preventing unsupported messages being sent across the C2C link.
3. C2C specific
    - Details which specification release the interface supports, to ensure backward compatibility.
    - Provides additional information to assist with link usage.

Both sides of the logical link send MiscU.Properties messages with an indication of their capabilities.

The property values in each Properties message are driven from a set of registers that define the capabilities of the chip sending the message.

There are certain properties that have Receiver, Rx, and Transmitter, Tx, variants:

Property_Rx This indicates the capability of the chip sending the Properties message as a Receiver. The Transmitter chip must be able to terminate any messages that the Receiver chip indicates it does not support.

Property_Tx This indicates the capability of the chip sending the Properties message as a Transmitter. Awareness at the Receiver chip of what it could receive enables potential power optimizations.

Where there is no _Tx or _Rx extension, the property is classified as Uniform and applies to the interface in both directions.

It is permitted for software to update the register values prior to sending a Properties message to reduce the advertised capabilities of the interface.

Upon receipt of a Properties message, the Receiver must compare each of the communicated properties against its own properties.

When the property values align, no modification of behavior is required.

When the properties differ, the Transmitter is permitted to adapt its behavior accordingly. The required adaptation depends upon the property classification and the differences seen in the:

- On-chip payload
    - If the Transmitter, _Tx, property is True and the Receiver, _Rx, property is False, it is permitted, but not required, that the associated field value is zeroed prior to message selection and container packing.
    - If the Transmitter, _Tx, property is False and the Receiver, _Rx, property is True, it is required that the associated field value is zeroed prior to message selection and container packing.

  - If the Transmitter, _Tx property is a larger value than the Receiver, _Rx, property, it is permitted, but not required, that the most significant bits that are not supported at the Receiver are zeroed at the Transmitter prior to message selection and container packing.
  - If the Transmitter, _Tx, property is a smaller value than the Receiver, _Rx, property, it is required that the associated field value is 0 extended at the Transmitter prior to message selection and container packing.
- On-chip opcode and flow
  - If the local Transmitter, _Tx, property is True and the Receiver, _Rx, property is False, the Transmitter must terminate or downgrade any associated on-chip message in an on-chip protocol-compliant manner. It is IMPLEMENTATION DEFINED how this downgrade or termination takes place.

Note

For example:

- Downgrade includes converting a SnpUniqueStash to a SnpUnique, if the target chip does not support stashing but has a copy of the data that is needed to complete a transaction.
- Termination includes a Requester performing an Atomic transaction locally when it is established that there is no downstream atomic support. This might be achieved through the use of an on-chip **BROADCASTATOMIC** signal on the Requester or an alternative control register.

  - If the Transmitter, _Tx, property is False and the Receiver, _Rx, property is True, there should be no further action required. The associated on-chip protocol messages should not be presented to the Transmitter for communication across the C2C link.
  - Optionally, if the Receiver, _Rx, property is True and the Transmitter, _Tx, property is False, the Receiver chip might be able to save power by turning off certain portions of logic.
- C2C specific
  - If the Transmitter, _Tx, property is True and the Receiver, _Rx, property is False, the Transmitter must not use the associated feature or transaction flow.
  - If the Transmitter, _Tx, property is False and the Receiver, _Rx, property is True, the Receiver might be able to save power by turning off certain portions of logic.
  - If the Transmitter, _Tx, and Receiver, _Rx, properties have values other than True or False, and these values differ between both sides of the link, additional statements will describe the expected behavior.
  - If the Uniform properties on either side of the link do not match, additional statements will describe the expected behavior.

The properties in opposing directions on a link can differ. That is, it may be possible that certain transactions can be initiated in one direction, but not the other.

B10.1.1 Properties message format

The MiscU.Properties field is used by the Protocol layers to exchange their supported property values with the remote Protocol layer.

Table B10.1 shows the Properties field encodings. See Table B10.11 for details of the Payload field.

Table B10.1: Properties message fields

Field name	Width (bits)	Value
MsgType	4	`0b0000`

Continued on next page

Table B10.1 – *Continued from previous page*

Field name	**Width (bits)**	**Value**
MiscOp	4	`0b0101`
Payload	152	
Total (bits)	160	

B10.2 On-chip payload properties

Table B10.2 lists the properties that are used to describe specific on-chip field payload characteristics of the chip as a Receiver.

Table B10.2: Receiver on-chip payload properties

Property	Width (bits)	Description	
MPAM_Support_Rx	2	Support for MPAM.	
		`0b00`	False. MPAM field is not present.
		`0b01`	True. MPAM field is present.
		Others	Reserved
PBHA_Support_Rx	2	Support for PBHA.	
		`0b00`	False. PBHA field is not present.
		`0b01`	True. PBHA field is present.
		Others	Reserved
MEC_Support_Rx	2	Support for MEC.	
		`0b00`	False. MECID field is not present.
		`0b01`	True. MECID field is present.
		Others	Reserved
RSVDC_REQ_Rx	3	Width of RSVDC on the REQ channel.	
		`0b0000`	0 bits
		`0b0001`	4 bits
		`0b0010`	8 bits
		`0b0011`	12 bits
		`0b0100`	16 bits
		`0b0101`	24 bits
		`0b0110`	32 bits
		Others	Reserved
RSVDC_DAT_Rx	3	Width of RSVDC on the DAT channel.	
		`0b0000`	0 bits
		`0b0001`	4 bits
		`0b0010`	8 bits
		`0b0011`	12 bits
		`0b0100`	16 bits
		`0b0101`	24 bits
		`0b0110`	32 bits
		Others	Reserved

Table B10.3 lists the properties that are used to describe specific on-chip field payload characteristics of the chip as a Transmitter.

Table B10.3: Transmitter on-chip payload properties

Property	Width (bits)	Description	
MPAM_Support_Tx	2	Support for MPAM.	
		`0b00`	False. MPAM field is not present.
		`0b01`	True. MPAM field is present.
		Others	Reserved
PBHA_Support_Tx	2	Support for PBHA.	
		`0b00`	False. PBHA field is not present.
		`0b01`	True. PBHA field is present.
		Others	Reserved
MEC_Support_Tx	2	Support for MEC.	
		`0b00`	False. MECID field is not present.
		`0b01`	True. MECID field is present.
		Others	Reserved
RSVDC_REQ_Tx	3	Width of RSVDC on the REQ channel.	
		`0b0000`	0 bits
		`0b0001`	4 bits
		`0b0010`	8 bits
		`0b0011`	12 bits
		`0b0100`	16 bits
		`0b0101`	24 bits
		`0b0110`	32 bits
		Others	Reserved
RSVDC_DAT_Tx	4	Width of RSVDC on the DAT channel.	
		`0b000`	0 bits
		`0b001`	4 bits
		`0b010`	8 bits
		`0b011`	12 bits
		`0b100`	16 bits
		`0b101`	24 bits
		`0b110`	32 bits
		Others	Reserved

B10.3 On-chip opcode and flow properties

Table B10.4 lists the properties that are used to describe specific on-chip opcode and flow support of the chip as a Receiver.

Table B10.4: Receiver on-chip opcode/flow properties

Property	Width (bits)	Description	
Atomic_Transactions_Rx	2	Support for Atomic transactions.	
		`0b00`	False. Atomic transactions not supported.
		`0b01`	True. Atomic transactions supported.
		Others	Reserved
Cache_Stash_Transactions_Rx	2	Support for Stashing requests and snoops.	
		`0b00`	False. Stash transactions not supported.
		`0b01`	True. Stash transactions supported.
		Others	Reserved
Persist_Transactions_Rx	2	Support for CMO operations to the PoP and PoDP.	
		`0b00`	False. Persist CMO transactions not supported.
		`0b01`	True. Persist CMO transactions supported.
		Others	Reserved
Deferrable_Write_Rx	2	Support for WriteNoSnpDef.	
		`0b00`	False. Deferrable Write transactions not supported.
		`0b01`	True. Deferrable Write transactions supported.
		Others	Reserved
RME_Support_Rx	2	Support for the Realm Management Extensions, including Realm and Root Physical Address Spaces and PoPA Cache Maintenance Operations.	
		`0b00`	False. RME transactions not supported.
		`0b01`	True. RME transactions supported.
		Others	Reserved
Snoop_Support_Rx	2	Support for Snoop transactions.	
		`0b00`	False. Snoop transactions not supported.
		`0b01`	True. Snoop transactions supported.
		Others	Reserved

Continued on next page

Table B10.4 – *Continued from previous page*

Property	Width (bits)	Description	
DVM_Support_Rx	2	Support for DVM messages.	
		`0b00`	False. DVM transactions not supported.
		`0b01`	True. DVM transactions supported.
		Others	Reserved
MTE_Support_Rx	2	Support for MTE.	
		`0b00`	False. MTE not supported.
		`0b01`	Reduced MTE transaction support.
		`0b10`	Full MTE transaction support.
		`0b11`	Reserved
CMO_Transactions_Rx	2	Support for CMO transactions.	
		`0b00`	False. CMO transactions not supported.
		`0b01`	True. CMO transactions supported.
		Others	Reserved

Table B10.5 lists the properties that are used to describe specific on-chip opcode and flow support of the chip as a Transmitter.

Table B10.5: Transmitter on-chip opcode/flow properties

Property	Width (bits)	Description	
Atomic_Transactions_Tx	2	Support for Atomic transactions.	
		`0b00`	False. Will not send Atomic transactions.
		`0b01`	True. Can send Atomic transactions.
		Others	Reserved
Cache_Stash_Transactions_Tx	2	Support for Stashing requests and snoop.	
		`0b00`	False. Will not send Stashing requests or snoops.
		`0b01`	True. Can send Stashing requests or snoops.
		Others	Reserved
Persist_Transactions_Tx	2	Support for CMO operations to the PoP and PoDP.	
		`0b00`	False. Will not send Persist CMO transactions.
		`0b01`	True. Can send Persist CMO transactions.
		Others	Reserved

Continued on next page

Table B10.5 – *Continued from previous page*

Property	Width (bits)	Description	
Deferrable_Write_Tx	2	Support for WriteNoSnpDef.	
		`0b00`	False. Will not send Deferrable Write transactions.
		`0b01`	True. Can send Deferrable Write transactions.
		Others	Reserved
RME_Support_Tx	2	Support for the Realm Management Extensions, including Realm and Root Physical Address Spaces, and PoPA Cache Maintenance Operations.	
		`0b00`	False. Will not send RME transactions.
		`0b01`	True. Can send RME transactions.
		Others	Reserved
Snoop_Support_Tx	2	Support for Snoop transactions.	
		`0b00`	False. Will not send Snoop transactions.
		`0b01`	True. Can send Snoop transactions.
		Others	Reserved
DVM_Support_Tx	2	Support for DVM messages.	
		`0b00`	False. Will not send DVM transactions.
		`0b01`	True. Can send DVM transactions.
		Others	Reserved
MTE_Support_Tx	2	Support for MTE.	
		`0b00`	False. Will only send transactions with TagOp *Invalid*.
		`0b01`	Reduced MTE transaction support. Will not send transactions with TagOp *Match*. Can send transactions with other TagOp values.
		`0b10`	Full MTE transaction support. Can send transactions with any TagOp value.
		`0b11`	Reserved
CMO_Transactions_Tx	2	Support for CMO transactions.	
		`0b00`	False. Will not send CMO transactions.
		`0b01`	True. Can send CMO transactions.
		Others	Reserved

B10.4 C2C specific properties

This section describes C2C specific properties, including:

- B10.4.1 *Uniform properties*
- B10.4.2 *Receiver and Transmitter properties*

B10.4.1 Uniform properties

Table B10.6 lists the properties that are used to describe C2C specific capabilities that are uniform in both directions of the interface.

Table B10.6: Uniform C2C specific properties

Property	Width (bits)	Description
Protocol	4	Protocol supported. `0b0000` CHI C2C Others Reserved
Version	4	Version supported. `0b0000` A Others Reserved
Num_Properties_Messages	4	Number of MiscU.Properties messages per logical link that can be sent and accepted by this interface. `0b0000` 1 Others Reserved
Container_Format	4	Container format supported by the interface in both directions. `0b0000` is not permitted. **Bit 0** Format X `0` Not supported `1` Supported **Bit 1** Format Y `0` Not supported `1` Supported **Bits 2 and 3** Reserved
Deactivation_Support	4	Deactivation modes supported by the interface. **Bit 0** Protocol Agnostic `0` Not supported `1` Supported **Bit 1** Protocol Aware `0` Not supported `1` Supported **Bits 2 and 3** Reserved

B10.4.2 Receiver and Transmitter properties

Table B10.7 lists the properties that are used to describe C2C specific capabilities of the interface .

Table B10.7: Receiver C2C specific properties

Property	Width (bits)	Description
Num_RP_REQ_Rx	4	Indicates the maximum number of resource planes the interface supports on the REQ channel. `0b0000` 1 `0b0001` 2 `0b0010` 3 `0b0011` 4 `0b0100` 5 `0b0101` 6 `0b0110` 7 `0b0111` 8 Others Reserved
WritePush_Support_Rx	2	Indicates if an interface supports WritePush flows for write transactions. `0b00` False. WritePush is not supported for Immediate Write and Atomic transactions. `0b01` True. WritePush is supported for both Immediate Write and Atomic transactions. Others Reserved
DVMPush_Support_Rx	2	Indicates if an interface supports WritePush flows for DVM transactions. `0b00` False. WritePush is not supported for DVM transactions. `0b01` True. WritePush is supported for DVM transactions. Others Reserved **Note** WrReqDataS messages cannot be used for DVM transactions, regardless of the DVMPush_Support_Rx property value.

Continued on next page

Table B10.7 – *Continued from previous page*

Property	Width (bits)	Description
DevAssign_Support_Rx	4	Support for RME-DA or RME-CDA. Only valid when RME_Support_Rx is True. Provided for informational purposes only, to assist with setup or debug. Not expected to have any functional effect in isolation. Each bit defines the roles of the other chip that this chip can connect to: **Bit 0** Host `0` Cannot connect to a host. `1` Can connect to a host. **Bit 1** Device_StreamID_SecSID1 `0` Cannot connect to a device. Does not use SecSID1 and StreamID fields. `1` Can connect a device. Can make use of SecSID1 and StreamID fields. **Bit 2** Device_NoStreamID_NoSecSID1 `0` Cannot connect to a device. Does not use SecSID1 and StreamID fields. `1` Can connect to a device. Does not use SecSID1 and StreamID fields. **Bit 3** Reserved

Table B10.8 lists the properties that are used to describe C2C specific capabilities of the interface as a Transmitter.

Table B10.8: Transmitter C2C specific properties

Property	Width (bits)	Description
Num_RP_REQ_Tx	4	Indicates the maximum number of resource planes the interface supports on the REQ channel. `0b0000` 1 `0b0001` 2 `0b0010` 3 `0b0011` 4 `0b0100` 5 `0b0101` 6 `0b0110` 7 `0b0111` 8 Others Reserved

Continued on next page

Table B10.8 – *Continued from previous page*

Property	Width (bits)	Description
WritePush_Support_Tx	2	Indicates if an interface supports a WritePush flow for write transactions. `0b00` False. WritePush is not supported for Immediate Write and Atomic transactions. `0b01` True. WritePush is supported for both Immediate Write and Atomic transactions. Others Reserved
DVMPush_Support_Tx	2	Indicates if an interface supports a WritePush flow for DVM transactions. `0b00` False. Will not use a WrReqDataL message for DVM transactions. `0b01` True. Can use a WrReqDataL message for DVM transactions. Others Reserved **Note** WrReqDataS messages cannot be sent for DVM transactions, regardless of the DVMPush_Support_Tx property value.
DevAssign_Support_Tx	4	Support for RME-DA or RME-CDA. Only valid when RME_Support_Tx is True. Provided for informational purposes only, to assist with setup or debug. Not expected to have any functional effect in isolation. Each bit defines the roles of the this chip that can be presented to the other chip: **Bit 0** Host `0` Cannot present as a host. `1` Can present as a host. **Bit 1** Device_StreamID_SecSID1 `0` Cannot present as a device. Does not send SecSID1 and StreamID fields. `1` Can present as a device. Sends SecSID1 and StreamID fields. **Bit 2** Device_NoStreamID_NoSecSID1 `0` Cannot present as a device. Does not send SecSID1 and StreamID fields. `1` Can present as a device. Does not send SecSID1 and StreamID fields. **Bit 3** Reserved

B10.4.3 Expectations when exchanged C2C specific properties differ

Table B10.9 details the expected outcome when C2C specific properties differ between two sides of a link and those properties are:

- Uniform properties
- Transmitter, _Tx, and Receiver, _Rx, properties that have values other than True or False

Table B10.9: Expectations when exchanged C2C specific properties differ

Property	Expectations
Protocol	If the remote interface supports a different protocol to the local interface, then there might be an incompatibility. Only one protocol is supported in this version of the specification.
Version	A newer version of the interface is expected to be backward compatible with older version. When the versions differ, both sides must negotiate to operate in the older version. Currently, there is only one version of the specification. Therefore, the two sides will not differ in their supported version.
Num_Properties_Msg	Linked to the Protocol property. The Transmitter should only send the correct number of MiscU.Properties messages that the Receiver can accept.
Received Num_RP_REQ_Rx and sent Num_RP_REQ_Tx	If the received Num_RP_REQ_Rx value is greater than the sent Num_RP_REQ_Tx value, there should not be any functional issues. The Transmitter might receive credits for Resource Planes that it does not intend to use. If the received Num_RP_REQ_Rx value is less than the sent Num_RP_REQ_Tx value, there might be functional issues if the Transmitter chip cannot consolidate traffic onto the reduced number of Resource Planes. The Transmitter will not receive credits for Resource Planes that it potentially needs to use for traffic dependency isolation. This should be considered at chip or chiplet selection and build time.
Received Num_RP_REQ_Tx and sent Num_RP_REQ_Rx	If the received Num_RP_REQ_Tx value is less than the sent Num_RP_REQ_Rx value, there should not be any functional issues. The Receiver is permitted, but not required, to redistribute the credits that it would have provided for the unused Resource Planes. If the received Num_RP_REQ_Tx value is greater than the sent Num_RP_REQ_Rx value, there is no expected change in behavior by the Receiver.
Received DevAssign_Support_Rx and sent DevAssign_Support_Tx	Provided for informational purposes only, to assist with setup or debug. Not expected to have any functional effect in isolation. Additional software attestation and configuration is expected.
Received DevAssign_Support_Tx and sent DevAssign_Support_Rx	Provided for informational purposes only, to assist with setup/debug. Not expected to have any functional effect in isolation. Additional software attestation and configuration is expected.
Container_Format	See B10.4.3.1 *Container_Format resolution*.

Continued on next page

Table B10.9 – *Continued from previous page*

Property	**Expectations**
Deactivation_Support	If none of the property bit position assertions match, deactivation is not supported by the protocol. If only one of the property bit position assertions matches, the associated mode is the only one supported. If multiple property bit position assertions match, any of associated deactivation approaches can be used.

B10.4.3.1 Container_Format resolution

There are two parts to the determination of the container format that is used:

1. The two Protocol layers exchange the Container_Format property to determine what options are available.
2. The container formats that can be supported at the Protocol layer are combined with the requirements of the FlitFormat needed by the Link layer.

Table B10.10 shows how these two aspects are combined.

Table B10.10: Enter table title

Container_Format supported	**FlitFormat negotiated**	
	6 bytes	**20 bytes**
Format X	Format X	Not compatible
Format Y	Format Y	Format Y
Format X and Format Y	Format X	Format Y

B10.5 Property packing in MiscU.Properties message

Table B10.11 illustrates the packing of properties in the payload of MiscU.Properties message.

Table B10.11: MiscU.Properties message format

MISC bit position	Field name	Width (bits)	Value	Property classification
3:0	MsgType	4	MiscU	Not applicable
7:4	MiscOp	4	Properties	
11:8	Num_Properties_Msg	4		Uniform properties
15:12	Protocol	4		
19:16	Version	4		
23:20	Container_Format	4		
27:24	Deactivation_Support	4		
39:28	Reserved	12		
43:40	Num_RP_REQ_Rx	4		Receiver properties
47:44	Reserved	4		
49:48	WritePush_Support_Rx	2		
51:50	DVMPush_Support_Rx	2		
53:52	Snoop_Transactions_Rx	2		
55:54	DVM_Support_Rx	2		
57:56	Atomic_Transactions_Rx	2		
59:58	Cache_Stash_Transactions_Rx	2		
61:60	Persist_Transactions_Rx	2		
63:62	MTE_Support_Rx	2		
65:64	Deferrable_Write_Rx	2		
67:66	RME_Support_Rx	2		
69:68	CMO_Transactions_Rx	2		
71:70	MPAM_Support_Rx	2		
73:72	PBHA_Support_Rx	2		
75:74	MEC_Support_Rx	2		
79:76	RSVDC_REQ_Rx	4		
83:80	RSVDC_DAT_Rx	4		
87:84	DevAssign_Support_Rx	4		
99:88	Reserved	12		
103:100	Num_RP_REQ_Tx	4		Transmitter properties

Continued on next page

Table B10.11 – *Continued from previous page*

MISC bit position	Field name	Width (bits)	Value	Property classification
107:104	Reserved	4		
109:108	WritePush_Support_Tx	2		
111:110	DVMPush_Support_Rx	2		
113:112	Snoop_Transactions_Tx	2		
115:114	DVM_Support_Tx	2		
117:116	Atomic_Transactions_Tx	2		
119:118	Cache_Stash_Transactions_Tx	2		
121:120	Persist_Transactions_Tx	2		
123:122	MTE_Support_Tx	2		
125:124	Deferrable_Write_Tx	2		
127:126	RME_Support_Tx	2		
129:128	CMO_Transactions_Tx	2		
131:130	MPAM_Support_Tx	2		
133:132	PBHA_Support_Tx	2		
135:134	MEC_Support_Tx	2		
139:136	RSVDC_REQ_Tx	4		
140:143	RSVDC_DAT_Tx	4		
147:144	DevAssign_Support_Tx	4		
159:148	Reserved	12		

B10.6 Property packing into registers

It is expected that each interface will have multiple sets of 64-bit registers that contain the properties for use during runtime. The defined sets of registers are detailed in Table B10.12.

Table B10.12: Property register sets

Register sets	Access permissions	Number of instances	Description
Supported	RO	1	Provides information on the local chip interface directly out of reset
Advertised	RW	1 per logical link	A copy of the Supported register that can be software-modified to downgrade the properties that are communicated in the MiscU.Properties message
Informed	RW	1 per logical link	A copy of the properties received in the MiscU.Properties message sent by the other side of the logical link
Negotiated	RO	1 per logical link	The outcome following the comparison of each property from both sides of the logical link. This can be used by the: – Transmitter to control what messages are sent. – Receiver to enable power saving on logic that will not be utilized.

The format used for Uniform property registers is shown in Table B10.13.

Table B10.13: Uniform register property format

Bit position	Bits	Property
3:0	4	Num_Properties_Msg
7:4	4	Protocol
11:8	4	Version
15:12	4	Container_Format
19:16	4	Deactivation_Support
63:20	44	Reserved

The format used for Receiver property registers is shown in Table B10.14.

Table B10.14: Receiver register property format

Bit position	Bits	Property
3:0	4	Num_RP_REQ_Rx
7:4	4	Reserved
9:8	2	WritePush_Support_Rx
11:10	2	DVMPush_Support_Rx
13:12	2	Snoop_Transactions_Rx
15:14	2	DVM_Support_Rx
17:16	2	Atomic_Transactions_Rx
19:18	2	Cache_Stash_Transactions_Rx
21:20	2	Persist_Transactions_Rx
23:22	2	MTE_Support_Rx
25:24	2	Deferrable_Write_Rx
27:26	2	RME_Support_Rx
29:28	2	CMO_Transactions_Rx
31:30	2	MPAM_Support_Rx
33:32	2	PBHA_Support_Rx
35:34	2	MEC_Support_Rx
39:36	4	RsvdC_REQ_Rx
43:40	4	RsvdC_DAT_Rx
47:44	4	DevAssign_Support_Rx
63:48	16	Reserved

The format used for Transmitter property registers is shown in Table B10.15.

Table B10.15: Transmitter register property format

Bit position	Bits	Property
3:0	4	Num_RP_REQ_Tx
7:4	4	Reserved
9:8	2	WritePush_Support_Tx
11:10	2	DVMPush_Support_Tx
13:12	2	Snoop_Transactions_Tx
15:14	2	DVM_Support_Tx
17:16	2	Atomic_Transactions_Tx
19:18	2	Cache_Stash_Transactions_Tx

Continued on next page

Table B10.15 – *Continued from previous page*

Bit position	**Bits**	**Property**
21:20	2	Persist_Transactions_Tx
23:22	2	MTE_Support_Tx
25:24	2	Deferrable_Write_Tx
27:26	2	RME_Support_Tx
29:28	2	CMO_Transactions_Tx
31:30	2	MPAM_Support_Tx
33:32	2	PBHA_Support_Tx
35:34	2	MEC_Support_Tx
39:36	4	RsvdC_REQ_Tx
43:40	4	RsvdC_DAT_Tx
47:44	4	DevAssign_Support_Tx
63:48	16	Reserved