AMBA® CHI

Architecture Specification

Document number	IHI0050
Document quality	Release
Document version	F.b
Document confidentiality	Non-confidential
Date of issue	Feb 2024

AMBA® CHI Architecture Specification

Release information

Date	Version	Changes
2024/Feb/28	F.b	• Eighth public release
2022/Sep/26	F	• Seventh public release
2022/Sep/26	E.c	• Sixth public release
2021/Aug/16	E.b	• Fifth public release
2020/Aug/19	E.a	• Fourth public release
2019/Aug/28	D	• Third public release
2018/May/08	C	• Second public release
2017/Aug/04	B	• First public release
2014/Jun/12	A	• First limited release

Non-Confidential Proprietary Notice

This document is **NON-CONFIDENTIAL** and any use by you is subject to the terms of this notice and the Arm AMBA Specification Licence set about below.

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-21451 version 2.2

AMBA SPECIFICATION LICENCE

THIS END USER LICENCE AGREEMENT ("LICENCE") IS A LEGAL AGREEMENT BETWEEN YOU (EITHER A SINGLE INDIVIDUAL, OR SINGLE LEGAL ENTITY) AND ARM LIMITED ("ARM") FOR THE USE OF ARM'S INTELLECTUAL PROPERTY (INCLUDING, WITHOUT LIMITATION, ANY COPYRIGHT) IN THE RELEVANT AMBA SPECIFICATION ACCOMPANYING THIS LICENCE. ARM LICENSES THE RELEVANT AMBA SPECIFICATION TO YOU ON CONDITION THAT YOU ACCEPT ALL OF THE TERMS IN THIS LICENCE. BY CLICKING "I AGREE" OR OTHERWISE USING OR COPYING THE RELEVANT AMBA SPECIFICATION YOU INDICATE THAT YOU AGREE TO BE BOUND BY ALL THE TERMS OF THIS LICENCE.

"LICENSEE" means You and your Subsidiaries. "Subsidiary" means, if You are a single entity, any company the majority of whose voting shares is now or hereafter owned or controlled, directly or indirectly, by You. A company shall be a Subsidiary only for the period during which such control exists.

1. Subject to the provisions of Clauses 2, 3 and 4, Arm hereby grants to LICENSEE a perpetual, non-exclusive, non-transferable, royalty free, worldwide licence to:

   (i) use and copy the relevant AMBA Specification for the purpose of developing and having developed products that comply with the relevant AMBA Specification;

   (ii) manufacture and have manufactured products which either: (a) have been created by or for LICENSEE under the licence granted in Clause 1(i); or (b) incorporate a product(s) which has been created by a third party(s) under a licence granted by Arm in Clause 1(i) of such third party's AMBA Specification Licence; and

   (iii) offer to sell, sell, supply or otherwise distribute products which have either been (a) created by or for LICENSEE under the licence granted in Clause 1(i); or (b) manufactured by or for LICENSEE under the licence granted in Clause 1(ii).

2. LICENSEE hereby agrees that the licence granted in Clause 1 is subject to the following restrictions:

   (i) where a product created under Clause 1(i) is an integrated circuit which includes a CPU then either: (a) such CPU shall only be manufactured under licence from Arm; or (b) such CPU is neither substantially compliant with nor marketed as being compliant with the Arm instruction sets licensed by Arm from time to time;

   (ii) the licences granted in Clause 1(iii) shall not extend to any portion or function of a product that is not itself compliant with part of the relevant AMBA Specification; and

   (iii) no right is granted to LICENSEE to sublicense the rights granted to LICENSEE under this Agreement.

3. Except as specifically licensed in accordance with Clause 1, LICENSEE acquires no right, title or interest in any Arm technology or any intellectual property embodied therein. In no event shall the licences granted in accordance with Clause 1 be construed as granting LICENSEE, expressly or by implication, estoppel or otherwise, a licence to use any Arm technology except the relevant AMBA Specification.

4. THE RELEVANT AMBA SPECIFICATION IS PROVIDED "AS IS" WITH NO REPRESENTATION OR WARRANTIES EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF SATISFACTORY QUALITY, MERCHANTABILITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE, OR THAT ANY USE OR IMPLEMENTATION OF SUCH ARM TECHNOLOGY WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADE SECRETS OR OTHER INTELLECTUAL PROPERTY RIGHTS.

5. NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS AGREEMENT, TO THE FULLEST EXTENT PETMITTED BY LAW, THE MAXIMUM LIABILITY OF ARM IN AGGREGATE FOR ALL CLAIMS MADE AGAINST ARM, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS AGREEMENT (INCLUDING WITHOUT LIMITATION (I) LICENSEE'S USE OF THE ARM TECHNOLOGY; AND (II) THE IMPLEMENTATION OF THE ARM TECHNOLOGY IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS AGREEMENT) SHALL NOT EXCEED THE FEES PAID (IF ANY) BY LICENSEE TO ARM UNDER THIS AGREEMENT. THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

6. No licence, express, implied or otherwise, is granted to LICENSEE, under the provisions of Clause 1, to use the Arm tradename, or AMBA trademark in connection with the relevant AMBA Specification or any products based thereon. Nothing in Clause 1 shall be construed as authority for LICENSEE to make any representations on behalf of Arm in respect of the relevant AMBA Specification.
7. This Licence shall remain in force until terminated by you or by Arm. Without prejudice to any of its other rights if LICENSEE is in breach of any of the terms and conditions of this Licence then Arm may terminate this Licence immediately upon giving written notice to You. You may terminate this Licence at any time. Upon expiry or termination of this Licence by You or by Arm LICENSEE shall stop using the relevant AMBA Specification and destroy all copies of the relevant AMBA Specification in your possession together with all documentation and related materials. Upon expiry or termination of this Licence, the provisions of clauses 6 and 7 shall survive.
8. The validity, construction and performance of this Agreement shall be governed by English Law.

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 Architecture Specification

Chapter B13 Link Layer

Part A
Preface

About this specification

This specification describes the AMBA® *Coherent Hub Interface* (CHI) architecture.

Intended audience

This specification is written for hardware and software engineers who want to become familiar with the CHI architecture and design systems and modules that are 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 architecture and the terminology in this specification.

Chapter B2 *Transactions*

Read this for an overview of the communication channels between nodes, the associated packet fields, transaction structures, transaction ID flows, and the supported transaction ordering.

Chapter B3 *Network Layer*

Read this for a description of the Network layer that is responsible for determining the node ID of a destination node.

Chapter B4 *Coherence Protocol*

Read this for an introduction to the coherence protocol.

Chapter B5 *Interconnect Protocol Flows*

Read this for examples of protocol flows for different transaction types.

Chapter B6 *Exclusive accesses*

Read this for a description of the mechanisms that the architecture includes to support Exclusive accesses.

Chapter B7 *Cache Stashing*

Read this for a description of the cache stashing mechanism whereby data can be installed in a cache.

Chapter B8 *DVM Operations*

Read this for a description of DVM operations that the protocol uses to manage virtual memory.

Chapter B9 *Error Handling*

Read this for a description of the error response requirements.

Chapter B10 *Realm Management Extension*

Read this for a description of the *Realm Management Extension* (RME).

Chapter B11 *System Control, Debug, Trace, and Monitoring*

Read this for a description of the mechanisms that provide additional support for the control, debugging, tracing, and performance measurement of systems.

Chapter B12 *Memory Tagging*

Read this for a description of the *Memory Tagging Extension* (MTE) that provides a mechanism to check the correct usage of data held in memory.

Chapter B13 *Link Layer*

Read this for a description of the Link layer that provides a mechanism for packet based communication between protocol nodes and the interconnect.

Chapter B14 *Link Handshake*

Read this for a description of the Link layer handshake requirements.

Chapter B15 *System Coherency Interface*

Read this for a description of the interface signals that support connecting and disconnecting components from both the Coherency and DVM domains.

Chapter B16 *Properties, Parameters, and Broadcast Signals*

Read this for a description of the optional signals that provide flexibility in configuring optional interface properties.

Chapter C1 *Message Field Mappings*

Read this for the field mappings for messages.

Chapter C2 *Communicating Nodes*

Read this for the node pairs that can legally communicate within the protocol.

Chapter C3 *Revisions*

Read this for a description of the technical changes between released issues of this specification.

Chapter D1 *Glossary*

Read this for definitions of terms used in this specification.

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.

Timing diagrams

The components used in timing diagrams are explained in Figure 1. Variations have clear labels, when they occur. Do not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

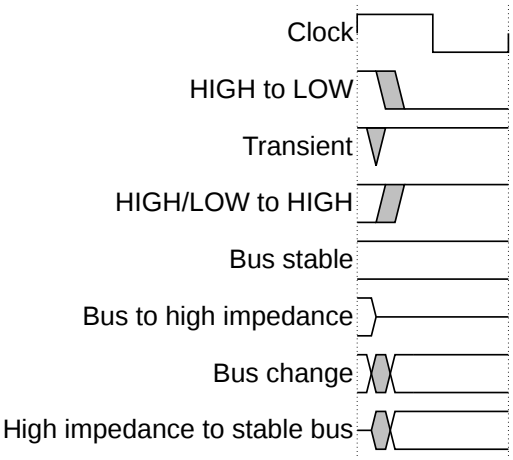

Figure 1: Key to timing diagram conventions

Timing diagrams sometimes show single-bit signals as HIGH and LOW at the same time and they look similar to the bus change shown in Figure 1 diagram conventions. If a timing diagram shows a single-bit signal in this way, then its value does not affect the accompanying description.

Time-Space diagrams

The Figure 2 figure explains the format used to illustrate protocol flow.

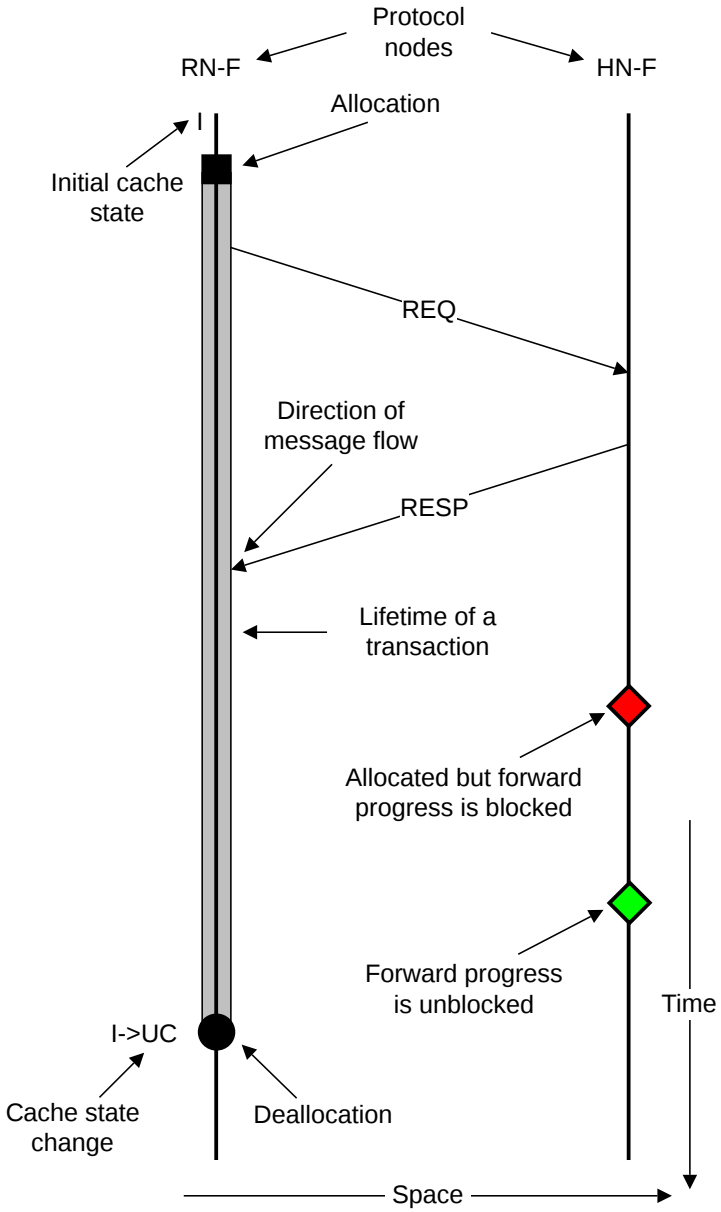

Figure 2: Key to Time-Space diagram conventions

In the Time-Space diagram:

- The protocol nodes are positioned along the horizontal axis and time is indicated vertically, top to bottom.
- The lifetime of a transaction at a protocol node is shown by an elongated shaded rectangle along the time axis from allocation to the deallocation time.
- The initial cache state at the node is shown at the top.
- The diamond shape on the timeline indicates arrival of a request and whether its processing is blocked waiting for another event to complete.
- The cache state transition, upon the occurrence of an event, is indicated by I->UC.

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.

Arm publications

- *AMBA® AXI Protocol Specification* (ARM IHI 0022).
- *Arm® Architecture Reference Manual for A-profile Architecture* (ARM DDI 0487).

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 Architecture Specification).
- The number (IHI0050 F.b).
- 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.

Previous issues of this document included terms that can be offensive. We have replaced these terms. If you find offensive terms in this document, please contact terms@arm.com.

Part B
Specification

Chapter B1
Introduction

This chapter introduces the CHI architecture and the terminology used throughout this specification. It contains the following sections:

B1.1 Architecture overview

The CHI architecture is a scalable, coherent hub interface and on-chip interconnect used by multiple components. The CHI architecture allows for flexible topologies of component connections, driven by performance, power, and area system requirements.

B1.1.1 Components

The components of CHI-based systems can comprise of:

- Standalone processors
- Processor clusters
- Graphic processors
- Memory controllers
- I/O bridges
- PCIe subsystems
- Interconnects

B1.1.2 Key features

The key features of the architecture are:

- Scalable architecture, enabling modular designs that scale from small to large systems.
- Independent layered approach, comprising of Protocol, Network, and Link layer, with distinct functionalities.
- Packet-based communication.
- All transactions handled by an interconnect-based Home Node that co-ordinates required snoops, cache, and memory accesses.
- The CHI coherence protocol supports:
  - Coherency granule of 64-byte cache line.
  - Snoop filter and directory-based systems for snoop scaling.
  - Both MESI and MOESI cache models with forwarding of data from any cache state.
  - Additional partial and empty cache line states.
- The CHI transaction set includes:
  - Enriched transaction types that permit performance, area, and power efficient system cache implementation.
  - Support for atomic operations and synchronization within the interconnect.
  - Support for the efficient execution of Exclusive accesses.
  - Transactions for the efficient movement and placement of data, to move data in a timely manner closer to the point of anticipated use.
  - Virtual memory management through *Distributed Virtual Memory* (DVM) operations.
- Request retry to manage protocol resources.
- Support for end-to-end *Quality of Service* (QoS).
- Support for the Arm *Memory Tagging Extension* (MTE).
- Support for the Arm *Realm Management Extension* (RME).

- Configurable data width to meet the requirements of the system.
- ARM TrustZone™ support on a transaction-by-transaction basis.
- Optimized transaction flow for coherent writes with a producer-consumer ordering model.
- Error reporting and propagation across components and interconnect for system reliability and integrity.
- Handling sub cache line Data Errors using Data Poisoning and per byte error indication.
- Power-aware signaling on the component interface:
  - Enabling flit-level clock gating.
  - Component activation and deactivation sequence for clock-gate and power-gate control.
  - Protocol activity indication for power and clock control.

B1.1.3 Architecture layers

Functionality is grouped into the following layers:

- Protocol
- Network
- Link

Table B1.1 describes the primary function of each layer.

Table B1.1: Layers of the CHI architecture

Layer	Communication granularity	Primary function
Protocol	Transaction	The Protocol layer is the topmost layer in the CHI architecture. The function of the Protocol layer is to: – Generate and process requests and responses at the protocol nodes. – Define the permitted cache state transitions at the protocol nodes that include caches. – Define the transaction flows for each request type. – Manage the protocol level flow control.
Network	Packet	The function of the Network layer is to: – Packetize the protocol message. – Determine the source and target node IDs required to route the packet over the interconnect to the required destination and add to the packet.
Link	Flit	The function of the Link layer is to: – Provide flow control between network devices. – Manage link channels to provide deadlock-free switching across the network.

B1.2 Topology

The CHI architecture is primarily topology-independent. However, certain topology-dependent optimizations are included in this specification to make implementation more efficient. Figure B1.1 shows three examples of topologies selected to show the range of interconnect bandwidth and scalability options that are available.

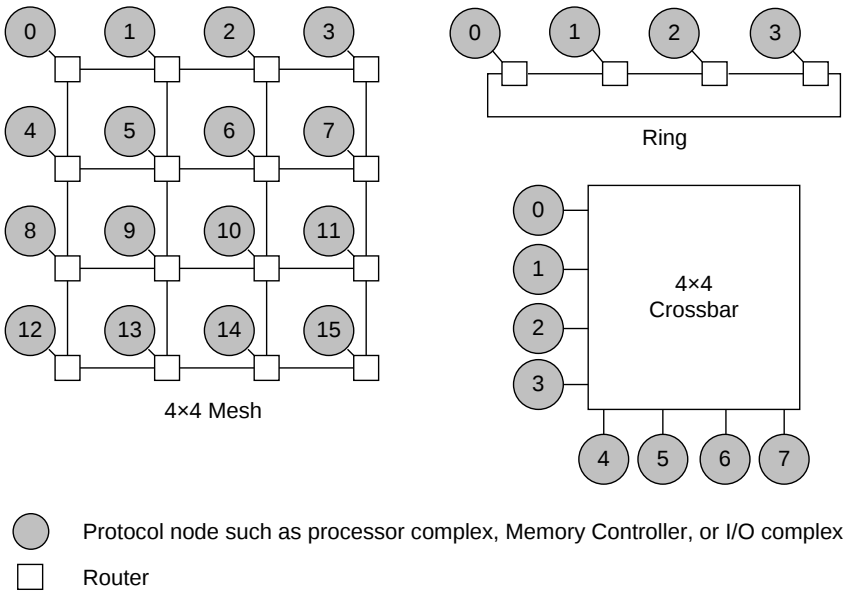

Figure B1.1: Example interconnect topologies

Crossbar Crossbar topology is simple to build and naturally provides an ordered network with low latency. Crossbar topology is suitable where the wire counts are still relatively small. Crossbar topology is suitable for an interconnect with a small number of nodes.

Ring Ring topology provides a trade-off between interconnect wiring efficiency and latency. The latency increases linearly with the number of nodes on the ring. Ring topology is suitable for a medium sized interconnect.

Mesh Mesh topology provides greater bandwidth at the cost of more wires. Mesh topology is very modular and can be easily scaled to larger systems by adding more rows and columns of switches. Mesh topology is suitable for a larger scale interconnect.

B1.3 Terminology

The following terms have a specific meaning within this specification:

Transaction

A transaction carries out a single operation. Typically, a transaction either reads from memory or writes to memory.

Message

A message is a protocol layer term that defines the granule-of-exchange between two components. Examples are:

- Request
- Data response
- Snoop request

A single Data response message can be made up of a number of packets.

Packet

A packet is the granule-of-transfer over the interconnect between endpoints. A message could be made up of one or more packets. For example, a single Data response message can be made up of 1 to 4 packets. Each packet contains routing information, such as destination ID and source ID, allowing for independent routing over the interconnect.

Flit

FLow control unIT (Flit) is the smallest flow control unit. A packet can be made up of one or more flits. All the flits of a given packet follow the same path through the interconnect.

> **Note**
>
> For CHI, all packets consist of a single flit.

Phit

PHysical layer transfer unIT (Phit) is one transfer between two adjacent network devices. A flit can be made up of one or more phits.

> **Note**
>
> For CHI, all flits consist of a single phit.

PoS

Point of Serialization (PoS) is a point within the interconnect where the ordering between requests from different agents is determined.

PoC

Point of Coherence (PoC) is a point at which all agents that can access memory are guaranteed to see the same copy of a memory location. In a typical CHI-based system, the PoC is the HN-F in the interconnect.

PoP

Point of Persistence (PoP) is a potential point in a memory system at or beyond the Point of Coherency, where a write to memory is maintained when system power is removed, and reliably recovered when power is restored to the affected locations in memory.

PoDP

Point of Deep Persistence (PoDP) is a point in the memory system where the data is preserved even when the power and the back up battery fail simultaneously.

PoPA

Point of Physical Aliasing (PoPA) is a point where updates to a location in one Physical Address Space (PAS) are visible to all other Physical Address Spaces.

Downstream cache

A downstream cache is defined from the perspective of a Request Node. A downstream cache for a Request is a cache that the Request accesses using CHI request transactions. A Request Node can send a request with data to allocate data into a downstream cache.

Requester

A component that starts a transaction by issuing a request message. The term Requester can be used for a component that independently initiates transactions. The term Requester can also be used for an interconnect component that issues a downstream Request message independently or as a side-effect of other transactions that are occurring in the system.

Completer

Any component that responds to a received transaction from another component. A Completer can either be an interconnect component, such as a Home Node or a Miscellaneous Node, or a component, such as a Subordinate, that is outside of the interconnect.

Subordinate

An agent that receives transactions and completes them appropriately. Typically, a Subordinate is the most downstream agent in a system. A Subordinate can also be referred to as a Completer or Endpoint.

Endpoint

Another name for a Subordinate component. An Endpoint is the final destination for a transaction.

Protocol Credit

A credit, or guarantee, from a Completer that it will accept a transaction.

Link layer Credit

A credit, or guarantee, that a flit will be accepted on the other side of the link. An L-Credit is a credit for a single hop at the Link layer.

ICN

Interconnect (ICN) is the CHI transport mechanism that is used for communication between protocol nodes. An interconnect can include an IMPLEMENTATION DEFINED fabric of switches connected in a ring, mesh, crossbar, or another topology. The interconnect can also include protocol nodes, such as Home Node and Miscellaneous Node.

IPA

Intermediate Physical Address (IPA). In two-stage address translation:

- Stage 1 provides an Intermediate Physical Address (IPA).
- Stage 2 provides the Physical Address (PA).

RN

Request Node (RN) generates protocol transactions, including reads and writes, to the interconnect.

HN

Home Node (HN) is a node within the interconnect that receives protocol transactions from Request Nodes, completes the required coherency action, and returns a response.

SN

Subordinate Node (SN) is a node that receives a Request from a Home Node, completes the required action, and returns a response.

MN

Miscellaneous or Misc Node (MN) is a node located within the interconnect that receives DVM messages from Request Nodes, completes the required action, and returns a response.

IO Coherent node

A Request Node that generates a subset of Snoopable requests in addition to Non-snoopable requests. The Snoopable requests that an IO Coherent node generates do not result in the caching of the received data in a coherent state. Therefore, an IO Coherent node does not receive any Snoop requests.

Snoopee

A Request Node that is receiving a snoop.

Write-Invalidate protocol

A protocol in which a Request Node writing to a shared cache line in the system must invalidate all copies before proceeding with the write. The CHI protocol is a Write-Invalidate protocol.

In a timely manner

The protocol cannot define an absolute time within which something must occur. A sufficiently idle system can progress and complete without explicit action.

Inapplicable

A field value that indicates that the field is not used in the processing of the message.

B1.4 Transaction classification

The protocol transactions that this specification supports, and their major classification, are listed in Table B1.2.

Table B1.2: Transaction classification

Classification	Supporting transactions
Read	ReadNoSnp
	ReadNoSnpSep
	ReadOnce
	ReadOnceCleanInvalid
	ReadOnceMakeInvalid
	ReadClean
	ReadNotSharedDirty
	ReadShared
	ReadUnique
	ReadPreferUnique
	MakeReadUnique
Dataless	CleanUnique
	MakeUnique
	Evict
	StashOnceUnique
	StashOnceSepUnique
	StashOnceShared
	StashOnceSepShared
	CleanShared
	CleanSharedPersist
	CleanSharedPersistSep
	CleanInvalid
	CleanInvalidPoPA
	MakeInvalid
Write	WriteNoSnpPtl
	WriteNoSnpFull
	WriteNoSnpZero
	WriteNoSnpDef
	WriteUniquePtl
	WriteUniqueFull

Continued on next page

Table B1.2 – *Continued from previous page*

Classification	Supporting transactions
	WriteUniqueZero
	WriteUniquePtlStash
	WriteUniqueFullStash
	WriteBackPtl
	WriteBackFull
	WriteCleanFull
	WriteEvictFull
	WriteEvictOrEvict
Combined Write	WriteNoSnpPtlCleanInv
	WriteNoSnpPtlCleanSh
	WriteNoSnpPtlCleanShPerSep
	WriteNoSnpPtlCleanInvPoPA
	WriteNoSnpFullCleanInv
	WriteNoSnpFullCleanSh
	WriteNoSnpFullCleanShPerSep
	WriteNoSnpFullCleanInvPoPA
	WriteUniquePtlCleanSh
	WriteUniquePtlCleanShPerSep
	WriteUniqueFullCleanSh
	WriteUniqueFullCleanShPerSep
	WriteBackFullCleanInv
	WriteBackFullCleanSh
	WriteBackFullCleanShPerSep
	WriteBackFullCleanInvPoPA
	WriteCleanFullCleanSh
	WriteCleanFullCleanShPerSep
Atomic	AtomicStore
	AtomicLoad
	AtomicSwap
	AtomicCompare
Other	DVMOp
	PrefetchTgt
	PCrdReturn
Snoop	SnpOnceFwd

Continued on next page

Table B1.2 – *Continued from previous page*

Classification	Supporting transactions
	SnpOnce
	SnpStashUnique
	SnpStashShared
	SnpCleanFwd
	SnpClean
	SnpNotSharedDirtyFwd
	SnpNotSharedDirty
	SnpSharedFwd
	SnpShared
	SnpUniqueFwd
	SnpUnique
	SnpPreferUniqueFwd
	SnpPreferUnique
	SnpUniqueStash
	SnpCleanShared
	SnpCleanInvalid
	SnpMakeInvalid
	SnpMakeInvalidStash
	SnpQuery
	SnpDVMOp

Table B1.3 shows the representations of transactions.

Table B1.3: Representation of transactions

Specification use	Represents collectively
ReadOnce*	ReadOnce, ReadOnceCleanInvalid, and ReadOnceMakeInvalid
WriteNoSnp	WriteNoSnpPtl and WriteNoSnpFull
WriteUnique	WriteUniquePtl, WriteUniqueFull, WriteUniquePtlStash, and WriteUniqueFullStash
WriteBack	WriteBackPtl and WriteBackFull
StashOnce	StashOnceUnique and StashOnceShared
StashOnceSep	StashOnceSepUnique and StashOnceSepShared
StashOnce*	StashOnce and StashOnceSep
StashOnce*Shared	StashOnceShared and StashOnceSepShared
StashOnce*Unique	StashOnceUnique and StashOnceSepUnique

Continued on next page

Table B1.3 – *Continued from previous page*

Specification use	**Represents collectively**
CleanSharedPersist*	CleanSharedPersist and CleanSharedPersistSep
SnpStash*	SnpStashUnique and SnpStashShared
DBIDResp*	DBIDResp and DBIDRespOrd

B1.5 Coherence overview

Hardware coherency enables system components to share memory without the requirement of software cache maintenance to maintain coherency.

Regions of memory are coherent if writes to the same memory location by two components are observable in the same order by all components.

B1.5.1 Coherency model

Figure B1.2 shows an example coherent system that includes three Requester components, each with a local cache and coherent protocol node. The protocol permits cached copies of the same memory location to reside in the local cache of one or more Requester components.

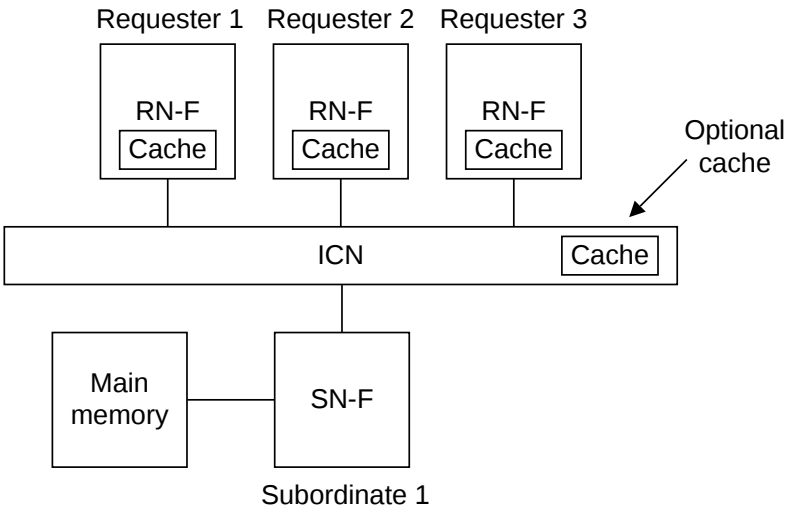

Figure B1.2: Example coherency model

The coherence protocol enforces that no more than one copy of a data value exists whenever a store occurs at an address location. The coherence protocol ensures all Requesters observe the correct data value at any given address location. After each store to a location, other Requesters can obtain a new copy of the data for their own local cache to permit multiple cached copies to exist.

A cache line is defined as a 64-byte aligned memory region. All coherency is maintained at cache line granularity.

Main memory is only required to be updated before a copy of the memory location is no longer held in any cache. The coherence protocol does not require main memory to be up to date at all times.

> **Note**
>
> Although not a requirement, it is permitted to update main memory while cached copies still exist.

The coherence protocol enables Requester components to determine whether a cache line is the only copy of a particular memory location or if other copies of the same location exist. The coherence protocol ensures:

- If a cache line is the only copy, a Requester component can change the value of the cache line without notifying any other Requester components in the system.
- If a cache line can also be present in another cache, a Requester component must notify the other caches using an appropriate transaction.

B1.5.2 Cache state model

When a component accesses a cache line, the protocol defines cache states to determine whether an action is required. Each cache state is based on the following cache line characteristics:

Valid, Invalid When Valid, the cache line is present in the cache.
When Invalid, the cache line is not present in the cache.

Unique, Shared When Unique, the cache line exists only in the Unique cache.
When Shared, the cache line can exist in more than one cache. The cache line is not guaranteed to exist in more than once cache.

Clean, Dirty When Clean, the cache does not have responsibility for updating main memory.
When Dirty, the cache line has been modified with respect to main memory. The Dirty cache must ensure that main memory is eventually updated.

Full, Partial, Empty A Full cache line has all bytes valid.
A Partial cache line can have some bytes valid, where some include none or all bytes.
An Empty cache line has no bytes valid.

Figure B1.3 shows the seven state cache model. B4.1 *Cache line states* gives further information about each cache state.

A valid cache state name that is not Partial or Empty is considered to be Full. In Figure B1.3, UC, UD, SC, and SD are all Full cache line states.

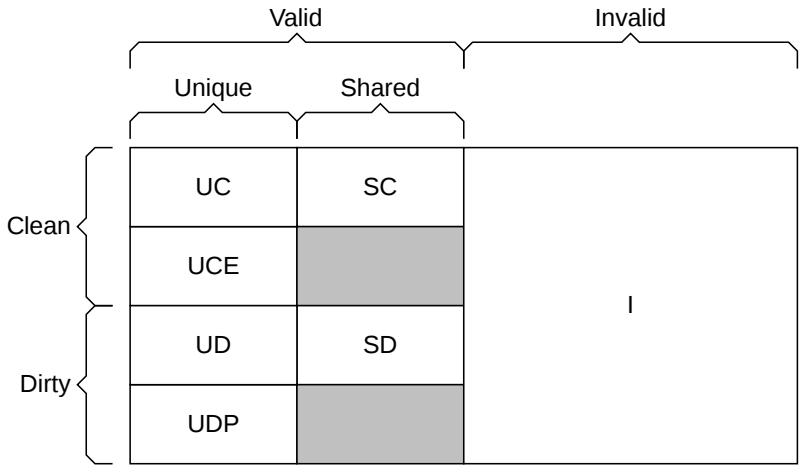

Figure B1.3: Cache state model

B1.6 Component naming

The components in the CHI protocol are classified by node type:

RN Request Node. Generates protocol transactions, including reads and writes, to the interconnect. A Request Node is further categorized as:

RN-F Fully Coherent Request Node:

- Includes a hardware-coherent cache.
- Permitted to generate all transactions defined by the protocol.
- Supports all Snoop transactions.

RN-D IO Coherent Request Node with DVM support:

- Does not include a hardware-coherent cache.
- Receives DVM transactions.
- Generates a subset of transactions defined by the protocol.

RN-I IO Coherent Request Node:

- Does not include a hardware-coherent cache.
- Does not receive DVM transactions.
- Generates a subset of transactions defined by the protocol.
- Does not require snoop functionality.

HN Home Node. Node located within the interconnect that receives protocol transactions from Request Nodes. A Home Node is further categorized as:

HN-F Fully Coherent Home Node:

- Expected to receive all request types, except DVMOp.
- Includes a *Point of Coherence* (PoC) that manages coherency by snooping the required RN-Fs, consolidating the Snoop responses for a transaction, and sending a single response to the requesting Request Node.
- Expected to be the *Point of Serialization* (PoS) that manages order between memory requests.
- Could include a directory or snoop filter to reduce redundant snoops.

> **Note**
>
> IMPLEMENTATION SPECIFIC, can include an integrated interconnect cache.

HN-I Non-coherent Home Node:

- Processes a limited subset of request types defined by the protocol.
- Does not include a PoC and is not capable of processing a Snoopable request. On receipt of a Snoopable request, HN-I must respond with a protocol compliant message.
- Expected to be the PoS that manages order between IO requests targeting the IO subsystem.

MN Miscellaneous Node.
Receives a DVM transaction from a Request Node, completes the required action, and returns a response.

SN Subordinate Node. Receives a request from a Home Node, completes the required action, and returns a response. A Subordinate Node is further categorized as:

SN-F Subordinate Node.

- Used for Normal memory.
- Can process Non-snoopable Read, Write, and Atomic requests, including exclusive variants of them, and *Cache Maintenance Operation* (CMO) requests.

SN-I Subordinate Node.

- Used for peripherals or Normal memory.
- Can process Non-snoopable Read, Write, and Atomic requests, including exclusive variants of them, and CMO requests.

Figure B1.4 shows various protocol node types connected through an interconnect.

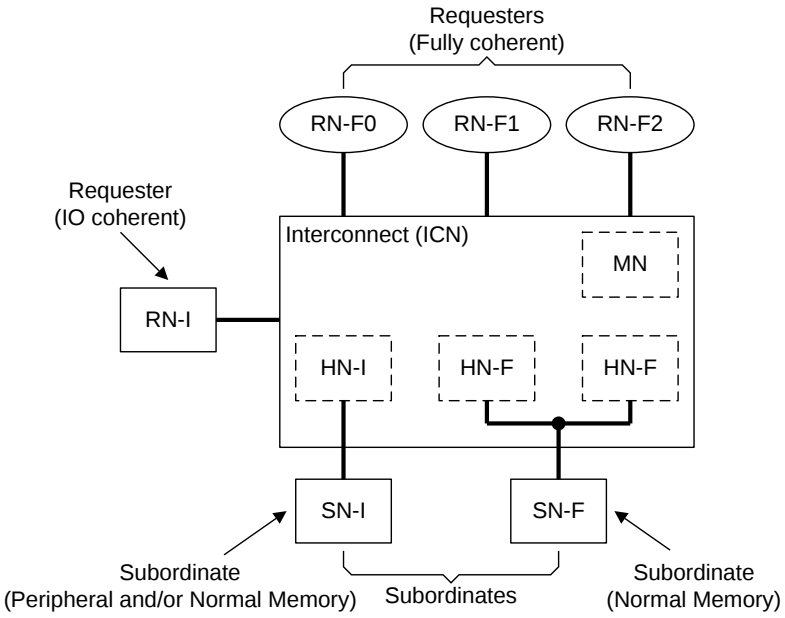

Figure B1.4: Protocol node examples

B1.7 Read data source

In a CHI-based system, a Read request can obtain data from different sources. Figure B1.5 shows that these sources are:

- Cache within the interconnect
- Subordinate Node
- Peer RN-F

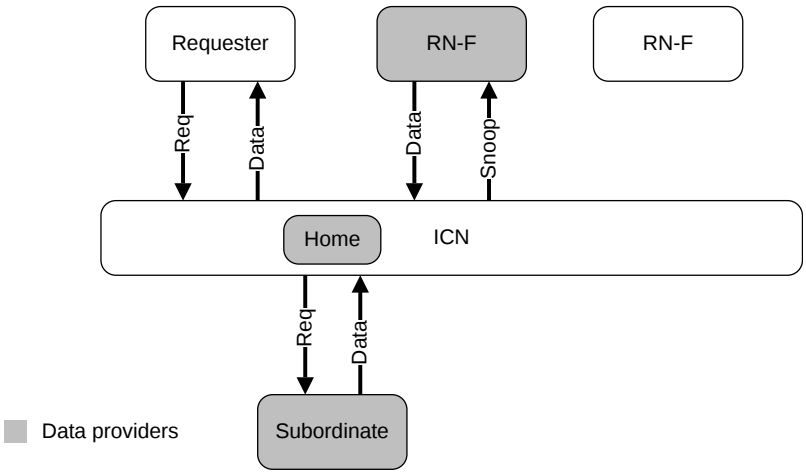

Figure B1.5: Possible Data providers for a Read request

One option for the Home is to request that the RN-F or Subordinate Node returns data only to Home. The Home, in turn, forwards a copy of the received data to the Requester. A hop in obtaining Data in the Read transaction flow can be removed if the Data provider is enabled to forward the Data response directly to the Requester instead of via the Home.

Several techniques can be used to reduce the number of hops to complete a transaction. The reduction in the number of hops results in Read and Write latency savings and interconnect bandwidth utilization. The techniques are categorized as:

Direct Memory Transfer (DMT)

Defines the feature that permits the Subordinate Node to send data directly to the Requester.

Direct Cache Transfer (DCT)

Defines the feature which permits a peer RN-F to send data directly to the Requester. The Data provider in the DCT Read transaction flows has to inform the Home that Data has been sent to the Requester. In some cases, the Data provider also has to send a copy of data to the Home.

Direct Write-data Transfer (DWT)

Defines the feature which permits the requesting Request Node to send write data directly to the Subordinate Node.

Chapter B2
Transactions

This chapter gives an overview of the communication channels between nodes, the associated packet fields, and the transaction structure. It contains the following sections:

B2.1 Channels overview

This section uses shorthand naming for the channels to describe the transaction structure. Table B2.1 shows the shorthand name and the physical channel name that exists on the Request Node or Subordinate Node component.

Communication between nodes is channel-based. Table B2.1 shows the channel naming and the channel designations at the Request Nodes and Subordinate Nodes.

See B13.4 *Channel* for the mapping of physical channels on the Request Node and Subordinate Node components.

Table B2.1: Channel naming and designation at the Request Node and Subordinate Node

Channel	Request Node channel designation	Subordinate Node channel designation
REQ	TXREQ. Outbound Request.	RXREQ. Inbound Request.
WDAT	TXDAT. Outbound Data. Use for write data, atomic data, snoop data, forward data.	RXDAT. Inbound Data. Use for write data, atomic data.
SRSP	TXRSP. Outbound Response. Use for snoop response and completion acknowledge.	-
CRSP	RXRSP. Inbound Response. Use for responses from the Completer.	TXRSP. Outbound Response. Use for responses from the Completer.
RDAT	RXDAT. Inbound Data. Use for read data, atomic data.	TXDAT. Outbound Data. Use for read data, atomic data.
SNP	RXSNP. Inbound Snoop request.	-

B2.2 Channel fields

This section gives a brief overview of the channel fields and indicates which fields affect the transaction structure. The associated fields with each channel are described in the following sections:

- B2.2.1 *Transaction request fields*
- B2.2.2 *Response fields*
- B2.2.3 *Snoop request fields*
- B2.2.4 *Data fields*

The term Transaction Structure is used to describe the different packets that form a transaction. The transaction structure can vary depending on several factors.

B2.2.1 Transaction request fields

Table B2.2 shows the Request fields associated with a Request packet.

More information on the different transaction structures can be found in B2.3 *Transaction structure* and B13.9 *Flit packet definitions*.

Table B2.2: Request channel fields

Field	Affects structure	Description
QoS	No	Quality of Service priority. Specifies one of 16 possible priority levels for the transaction with ascending values of QoS indicating higher priority levels. See B13.10.1 *Quality of Service, QoS*.
TgtID	No	Target Identifier. The node identifier of the port on the component to which the packet is targeted. See B2.4.1 *Target Identifier, TgtID, and Source Identifier, SrcID* and B3.1 *System Address Map, SAM*.
SrcID	No	Source Identifier. The node identifier of the port on the component from which the packet was sent. See B2.4.1 *Target Identifier, TgtID, and Source Identifier, SrcID*.
TxnID	No	Transaction Identifier. A transaction has a unique transaction identifier per source node. See B2.4.2 *Transaction Identifier, TxnID*.
ReturnNID	No	Return Node Identifier. The recipient node identifier for the Data response, Persist response, or TagMatch response. See B2.4.10 *Return Node Identifier, ReturnNID*.
StashNID	No	Stash Node Identifier. The node identifier of the Stash target. See B2.4.9 *Stash Node Identifier, StashNID* and B13.10.9 *Stash Node Identifier, StashNID*.
SLCRepHint	No	System Level Cache Replacement Hint. Forwards cache replacement hints from the Requesters to the caches in the interconnect. See B11.3 *SLC Replacement Hint*.
StashNIDValid	Yes	Stash Node Identifier Valid. Indicates that the StashNID field has a valid Stash target value. See B13.10.10 *Stash Node Identifier Valid, StashNIDValid*.
Endian	No	Endianness. Indicates the endianness of data in the data packet for Atomic transactions. See B2.8.5.3 *Endianness*.

Continued on next page

Table B2.2 – *Continued from previous page*

Field	Affects structure	Description
Deep	No	Deep persistence. Indicates that the Persist response must not be sent until all earlier writes are written to the final destination. See B4.2.2.2.3 *Deep Persistent CMO*.
ReturnTxnID	No	Return Transaction Identifier. The unique transaction identifier that conveys the value of TxnID in the data response from the Subordinate. See B2.4.4 *Return Transaction Identifier, ReturnTxnID*.
StashLPIDValid	No	Stash Logical Processor Identifier Valid. Indicates that the StashLPID field value is the Stash target. See B13.10.12 *Stash Logical Processor Identifier Valid, StashLPIDValid*.
StashLPID	No	Stash Logical Processor Identifier. The identifier of the Logical Processor (LP) at the Stash target. See B13.10.11 *Stash Logical Processor Identifier, StashLPID*.
Opcode	Yes	Request opcode. Specifies the transaction type and is the primary field that determines the transaction structure. See B4.2 *Request types* and B13.10.18.1 *REQ channel opcodes*.
Size	Yes	Data size. Specifies the size of the data associated with the transaction and determines the number of data packets within the transaction. See B2.8 *Data transfer*.
Addr	No	Address. The address of the memory location being accessed for Read and Write requests. See B2.7.1 *Address* and B13.10.20 *Address, Addr*.
NS	No	Non-secure. Combined with NSE to establish the Physical Address Space (PAS) of an access. See B2.7.2 *Physical Address Space, PAS*.
NSE	No	Non-secure Extension. Combined with NS to establish the PAS of an access. See B2.7.2 *Physical Address Space, PAS*.
LikelyShared	No	Likely Shared. Provides an allocation hint for downstream caches. See B2.7.5 *Likely Shared*.
AllowRetry	Yes	Allow Retry. Determines if the target is permitted to give a Retry response. See B2.9 *Request Retry*.
Order	Yes	Order requirement. Determines the ordering requirement for a request with respect to other requests from the same agent. See B2.6 *Ordering*.
PCrdType	No	Protocol Credit Type. Indicates the type of Protocol Credit being used by a request that has the B13.10.33 *Allow Retry, AllowRetry* field deasserted. See B2.9 *Request Retry*.
MemAttr	No	Memory attribute. Determines the memory attributes associated with the transaction. See B2.7.3 *Memory Attributes*.
SnpAttr	No	Snoop attribute. Specifies the snoop attributes associated with the transaction. See B2.7.6 *Snoop attribute*.
DoDWT	Yes	Do Direct Write Transfer. Supports Direct Write-data Transfer and the handling of Combined Writes. See B13.10.26 *Do Direct Write Transfer, DoDWT*.

Continued on next page

Table B2.2 – *Continued from previous page*

Field	Affects structure	Description
PGroupID	No	Persistence Group Identifier. Indicates the set of CleanSharedPersistSep transactions to which the request applies. See B13.10.8 *Persistence Group Identifier, PGroupID*.
StashGroupID	No	Stash Group Identifier. Indicates the set of StashOnceSep transactions to which the request applies. See B13.10.13 *Stash Group Identifier, StashGroupID*.
TagGroupID	No	Tag Group Identifier. Precise contents are IMPLEMENTATION DEFINED. Typically expected to contain Exception Level, *Translation Table Base Register* (TTBR) value, and CPU identifier. See B13.10.43 *Tag Group Identifier, TagGroupID*.
LPID	No	Logical Processor Identifier. Used with the SrcID field to uniquely identify the LP that generated the request. See B2.4.7 *Logical Processor Identifier, LPID*.
Excl	No	Exclusive access. Indicates that the corresponding transaction is an Exclusive access transaction. See Chapter B6 *Exclusive accesses*.
SnoopMe	No	Snoop Me. Indicates that Home must determine whether to send a snoop to the Requester during an Atomic transaction. See B2.3.3 *Atomic transactions*.
CAH	Yes	CopyAtHome. In CopyBack requests, CAH indicates to the Home if the Requester modifies the line or MTE tags since Home indicated a copy of the line was kept. See B13.10.30 *CopyAtHome, CAH*.
ExpCompAck	Yes	Expect CompAck. Indicates that the transaction includes a completion acknowledge message. See B2.3 *Transaction structure* and B2.6 *Ordering*.
TagOp	Yes	Tag Operation. Indicates the operation to be performed and on the tags present in the corresponding DAT channel. See B13.10.40 *Tag Operation, TagOp*.
TraceTag	No	Trace Tag. Provides extra support for the debugging, tracing, and performance measurement of systems. See Chapter B11 *System Control, Debug, Trace, and Monitoring*.
MPAM	No	Memory System Resource Partitioning and Monitoring. Efficiently utilizes the memory resources among users and monitors their use. See B11.4 *MPAM*.
PBHA	No	Page-based Hardware Attributes. 4 bits from the translation tables that can be used for IMPLEMENTATION DEFINED hardware control. See B11.5 *Page-based Hardware Attributes*.
RSVDC	No	User-defined. See B13.10.59 *Reserved for Customer Use, RSVDC*.

B2.2.2 Response fields

Table B2.3 describes the fields associated with a Response packet.

Table B2.3: Response packet fields

Field	Description
QoS	Quality of Service priority. As defined in Table B2.2. See B11.1 *Quality of Service (QoS) mechanism*.
TgtID	Target Identifier. As defined in Table B2.2. See B2.4.1 *Target Identifier, TgtID, and Source Identifier, SrcID*.
SrcID	Source Identifier. As defined in Table B2.2. See B2.4.1 *Target Identifier, TgtID, and Source Identifier, SrcID*.
TxnID	Transaction Identifier. As defined in Table B2.2. See B2.4.2 *Transaction Identifier, TxnID*.
Opcode	Response opcode. Specifies response type. See B13.10.18.2 *RSP channel opcodes*.
RespErr	Response Error status. As defined in Table B2.5. See Chapter B6 *Exclusive accesses* and B9.1.2 *Error response fields*.
Resp	Response status. As defined in Table B2.5. See B4.5 *Response types*.
FwdState	Forward State. As defined in Table B2.5. See B13.10.49 *Forward State, FwdState*.
DataPull	Data Pull. As defined in Table B2.5. See B7.1.1 *Snoop requests and Data Pull*.
CBusy	Completer Busy. As defined in Table B2.5. See B11.6 *Completer Busy*.
DBID	Data Buffer Identifier. As defined in Table B2.5. See B2.4.3 *Data Buffer Identifier, DBID* and B2.6 *Ordering*.
PGroupID	Persistence Group Identifier. As defined in Table B2.2. See B13.10.8 *Persistence Group Identifier, PGroupID*.
StashGroupID	Stash Group Identifier. As defined in Table B2.2. See B13.10.13 *Stash Group Identifier, StashGroupID*.
TagGroupID	Tag Group Identifier. As defined in Table B2.2. See B13.10.43 *Tag Group Identifier, TagGroupID*.
PCrdType	Protocol Credit Type. See B2.9.2.2 *PCRdType*.
TagOp	Tag Operation. As defined in Table B2.2. See B13.10.40 *Tag Operation, TagOp*.
TraceTag	Trace Tag. As defined in Table B2.2. See Chapter B11 *System Control, Debug, Trace, and Monitoring*.

B2.2.3 Snoop request fields

Table B2.4 shows the Snoop request fields. Many of the Snoop request fields are the same as fields defined for the Request channel.

Table B2.4: Snoop request fields

Field	Affects structure	Description
QoS	No	Quality of Service priority. As defined in Table B2.2. See B11.1 *Quality of Service (QoS) mechanism*.
SrcID	No	Source Identifier. As defined in Table B2.2. See B2.4.1 *Target Identifier, TgtID, and Source Identifier, SrcID*.
TxnID	No	Transaction Identifier. As defined in Table B2.2. See B2.4.2 *Transaction Identifier, TxnID*.
FwdNID	No	Forward Node Identifier. The node identifier of the original Requester. See B2.4.12 *Forward Node Identifier, FwdNID*.
PBHA	No	Page-based Hardware Attributes. 4 bits from the translation tables that can be used for IMPLEMENTATION DEFINED hardware control. See B11.5 *Page-based Hardware Attributes*.
FwdTxnID	No	Forward Transaction Identifier. The transaction identifier used in the Request by the original Requester. See B2.4.5 *Forward Transaction Identifier, FwdTxnID*.
StashLPIDValid	No	Stash Logical Processor Identifier Valid. As defined in Table B2.2. See B7.5 *Stash messages*.
StashLPID	No	Stash Logical Processor Identifier. As defined in Table B2.2. See B7.5 *Stash messages*.
VMIDExt	No	Virtual Machine Identifier Extension. See B8.3.3 *Snoop DVMOp field value restrictions*.
Opcode	Yes	Snoop opcode. See B13.10.18.3 *SNP channel opcodes*.
Addr	No	Address. The address of the memory location being accessed for Snoop requests. See B2.7.1 *Address* and B13.10.20 *Address, Addr*.
NS	No	Non-secure. As defined in Table B2.2. See B2.7.2 *Physical Address Space, PAS*.
NSE	No	Non-secure Extension. As defined in Table B2.2. See B2.7.2 *Physical Address Space, PAS*.
DoNotGoToSD	No	Do Not Go To SD state. Controls Snoopee use of SD state. See B4.10 *Do not transition to SD*.
RetToSrc	Yes	Return to Source. Instructs the Receiver of the snoop to return data with the Snoop response. See B4.9 *Returning Data with Snoop response*.
TraceTag	No	Trace Tag. As defined in Table B2.2. See Chapter B11 *System Control, Debug, Trace, and Monitoring*.
MPAM	No	Memory System Resource Partitioning and Monitoring. As defined in Table B2.2. See B11.4 *MPAM*.

Note

This specification does not define a TgtID field for the Snoop request. See B3.3 *TgtID determination*.

B2.2.4 Data fields

Table B2.5 describes the fields associated with a Data packet. Data packets can be sent on the RDAT or WDAT channels. The fields in a Data packet do not affect the transaction structure.

Table B2.5: Data packet fields

Field	Description
QoS	Quality of Service priority. As defined in Table B2.2. See B11.1 *Quality of Service (QoS) mechanism*.
TgtID	Target Identifier. As defined in Table B2.2. See B2.4.1 *Target Identifier, TgtID, and Source Identifier, SrcID*.
SrcID	Source Identifier. As defined in Table B2.2. See B2.4.1 *Target Identifier, TgtID, and Source Identifier, SrcID*.
TxnID	Transaction Identifier. As defined in Table B2.2. See B2.4.2 *Transaction Identifier, TxnID*.
HomeNID	Home Node Identifier. The node identifier of the target of the CompAck response to be sent from the Requester. See B2.4.11 *Home Node Identifier, HomeNID*.
PBHA	Page-based Hardware Attributes. 4 bits from the translation tables that can be used for IMPLEMENTATION DEFINED hardware control. See B11.5 *Page-based Hardware Attributes*.
Opcode	Data opcode. Indicates, for example, if the data packet is related to a Read transaction, a Write transaction, or a Snoop transaction. See B13.10.18.4 *DAT channel opcodes*.
RespErr	Response Error status. Indicates the error status associated with a data transfer. See Chapter B6 *Exclusive accesses* and B9.1.2 *Error response fields*.
Resp	Response status. Indicates the cache line state associated with a data transfer. See B4.5 *Response types*.
DataSource	Data Source. The value indicates the source of the data in a Read Data response. See B11.2 *Data Source indication*.
FwdState	Forward State. Indicates the cache line state associated with a data transfer to the Requester from the receiver of the snoop. See B13.10.49 *Forward State, FwdState*.
DataPull	Data Pull. Indicates the inclusion of an implied Read request in the Data response. See B7.1.1 *Snoop requests and Data Pull*.
CBusy	Completer Busy. Indicates the current level of activity at the Completer. See B11.6 *Completer Busy*.
DBID	Data Buffer Identifier. The identifier to be used as the TxnID in the response to the Data message. See B2.4.3 *Data Buffer Identifier, DBID* and B2.6 *Ordering*.
CCID	Critical Chunk Identifier. Replicates the address offset of the original Transaction request. See B2.8 *Data transfer*.

Continued on next page

Table B2.5 – *Continued from previous page*

Field	Description
DataID	Data Identifier. Provides the address offset of the data provided in the packet. See B2.8 *Data transfer*.
TagOp	Tag Operation. As defined in Table B2.2. See B13.10.40 *Tag Operation, TagOp*.
Tag	Memory Tag. Provides sets of 4-bit tags. Each tag is associated with an aligned 16-byte of data See B13.10.41 *Tag*.
TU	Tag Update. Indicates which of the Allocation Tags must be updated. See B13.10.42 *Tag Update, TU*.
TraceTag	Trace Tag. As defined in Table B2.2. See Chapter B11 *System Control, Debug, Trace, and Monitoring*.
CAH	CopyAtHome. In responses from Home or a Snoopee, indicates if the Home keeps a copy of the line that is provided to the Requester. Also defined in Table B2.2. For more information, see B13.10.30 *CopyAtHome, CAH*.
RSVDC	User-defined. See B13.10.59 *Reserved for Customer Use, RSVDC*.
BE	Byte Enable. For a data write, or data provided in response to a snoop, indicates which bytes are valid. See B2.8 *Data transfer*.
Data	Data payload. See B2.8 *Data transfer*.
DataCheck	Data Check. Detects Data Errors in the DAT packet. See B9.2.2 *Data Check*.
Poison	Poison. Indicates that a set of data bytes has previously been corrupted. See B9.2.1 *Poison*.

B2.3 Transaction structure

This section describes the ways that a transaction can complete. This section describes all the permitted options that can be used by the various components that participate in a transaction.

All transaction types, except PCrdReturn and PrefetchTgt, can have a Retry sequence at the start of the transaction. For ease of presentation, the Retry sequence is described separately, see B2.3.8 *Retry*.

Other independent transactions may need to be performed to complete certain transactions, such as a snoop or memory transaction. These transactions are described separately in the later section, B2.3.9 *Home Initiated transactions*.

Some transactions from Home to Subordinate support the use of a separate ReturnNID and ReturnTxnID field, which allow certain responses to be returned to the original Requester rather than the Home. It is permitted, but not required, to set the ReturnNID and ReturnTxnID fields, such that they are equal to the SrcID and TxnID. This means all responses are returned to the Home. In this instance, the transaction from Home to Subordinate is considered as if it were an independent transaction. See B2.3.9 *Home Initiated transactions* for more details.

Typically, a field or opcode in the request determines whether a particular message can be included in the transaction flow, described as Optional. Optional messages do not express whether a sender chooses to send the message.

The term Requester is always used to refer to the original Requester of a transaction in this section. Requester does not refer to an intermediate agent that issues a secondary request to complete the original transaction. The term Requester always refers to a Request Node, RN-F, RN-I, or RN-D.

In this section, a diagram with an arrow containing multiple message labels indicates any one of the messages can be sent in a flow. Requirements of when a particular message can be used in the transaction flow can be derived from elsewhere in the specification.

The flows described in this section are intended to include a complete description of the messages that can be included in a transaction flow. The following is not included:

- The channel that is used for a particular transaction.
- A description of when a transaction can be issued or what the reason is for using a particular transaction.
- A description of which combinations of fields can be used in a request. Request fields are only highlighted when they affect the options for completing a transaction.

Dependencies described in this section use the following approach:

- An implicit dependency exists for each component when first participating in a transaction sequence. Except for the original Requester, any other component must receive a first message associated with a transaction before any subsequent messages are sent for that transaction.
- Where a component sends multiple messages associated with the same transaction, the messages are assumed to be sent in any order. The messages are also assumed to be received in any order, unless an explicit dependency is described.
- Data transfers are shown in the diagrams as a single message. This message can consist of multiple data transfers, see B2.8 *Data transfer* for more details. Where a dependency is described, the dependency is from the first data transfer received, except when the dependency is explicitly stated to be different.
- Where a required or permitted dependency exists in one direction between two components, a dependency in the opposite direction is not permitted. The text in this section only describes the dependency in one direction.
- The amount of detail in a dependency rule is context dependent. When the source of the dependency and what is dependent is obvious, the agents are not included.
- For any agent that has one or more outputs:

  - If the agent is the original Requester of the transaction, dependencies are described for each input to that agent.
  - If the agent is not the original Requester of the transaction, dependencies are described for each input to that agent after the first input that agent received. Every output is considered dependent on the first input.
- This section does not describe all dependencies across different transactions, even when they are both to the same address. See B2.6 *Ordering* and B4.11 *Hazard conditions*.

The conventions used to describe actions in this section are:

Issues	Used only for the first message in a transaction, for example: The Requester issues a WriteNoSnp request...
Sends	A message sent by an agent in a direction away from the Requester.
Sends a downstream	A message relayed by an intermediate agent in a direction away from the Requester, for example: The Home sends a downstream Read request to the Subordinate.
Returns	A message sent by an agent in a direction towards the Requester.
Provides	A message sent by an agent in response to a snoop, for example: ...provides a snoop response.
Permitted, but not required	The action is not encouraged nor discouraged, and is compliant in either case.
Not permitted	The action described would cause non-compliance to the specification. For example: The Home is not permitted to wait for...before sending...

B2.3.1 Read transactions

Read transactions are grouped into the following types:

- B2.3.1.1 *Allocating Read*
- B2.3.1.2 *Non-allocating Read*

B2.3.1.1 Allocating Read

Figure B2.1 shows the possible transaction flows for an Allocating Read transaction.

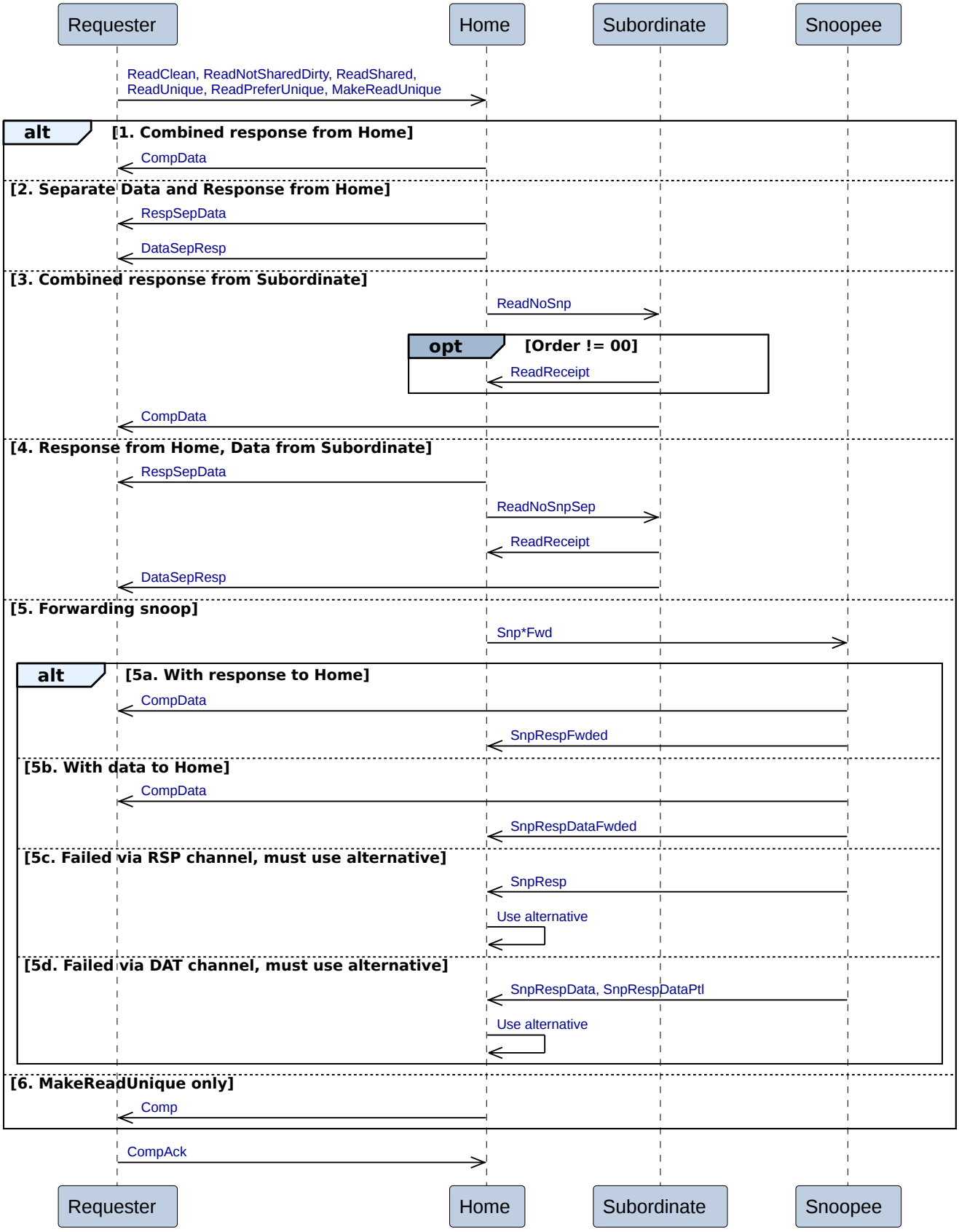

Figure B2.1: Allocating Read

The sequence for the Allocating Read transaction is:

- The transaction starts with the Requester issuing an Allocating Read request to the Home. The initial request is one of the following:
  - ReadClean
  - ReadNotSharedDirty
  - ReadShared
  - ReadUnique
  - ReadPreferUnique
  - MakeReadUnique
- Alternatives 1-6 show different ways that the Home can process the transaction:

  1. **Combined response from Home**

     The Home returns a combined response and read data, CompData, to the Requester. Typically, this option is used by the Home when data and response can be returned at the same time. An example is when the data is cached locally.

  2. **Separate data and response from Home**

     The Home returns a separate response, RespSepData, and read data, DataSepResp, to the Requester. Typically, this option is used by the Home when a response can be returned quicker than the Home can provide the data.

  3. **Combined response from Subordinate**

     - The Home sends a downstream read request, ReadNoSnp, to the Subordinate.
     - Optionally, when the Home requests a ReadReceipt response, the Subordinate returns a read receipt, ReadReceipt, to the Home.
     - The Subordinate returns a combined response and read data, CompData, to the Requester.

     Typically, this option is used by the Home either to reduce the message count or reduce design complexity.

  4. **Response from Home, Data from Subordinate**

     - The Home returns a separate response, RespSepData, to the Requester.
     - The Home sends a downstream read data only request, ReadNoSnpSep, to the Subordinate.
     - The Subordinate returns a read receipt, ReadReceipt, to the Home. It is permitted, but not required, for the Home to wait for ReadReceipt from the Subordinate before sending RespSepData to the Requester.
     - The Subordinate returns read data, DataSepResp, to the Requester. Typically, this option is used by the Home when a response can be returned quickly. However, the Home does not have the data available and requires the Subordinate to return the data.

     Note

     In many circumstances, the Requester receives RespSepData far in advance to DataSepResp. The Requester is permitted, but not required, to send the CompAck response after receiving RespSepData without waiting for DataSepResp. The prompt response from the Requester, and potentially receiving ReadReceipt around the same time, permits the Home to complete the transaction faster than when using combined completion and data responses from the Subordinate.

  5. **Forwarding snoop**

– The Home requests a Snoopee to forward read data, Snp*Fwd, to the Requester. See B4.4 *Request transactions and corresponding Snoop requests* to determine which Snp*Fwd transactions can be used. Typically, this option is used by the Home when the data is not cached locally and the Home determines that a Snoopee is likely to have a copy.

– Alternatives 5a-5d show how the Snoopee can process the transaction:

Alt 5a. With response to Home

– The Snoopee returns a combined response and read data, CompData, to the Requester.

– The Snoopee provides a snoop response, SnpRespFwded, to the Home. Typically, this option is used by the Snoopee when data can be forwarded to the Requester and is not required to provide a copy of data to the Home.

Alt 5b. With data to Home

– The Snoopee returns a combined response and read data, CompData, to the Requester.

– The Snoopee provides a snoop response with data, SnpRespDataFwded, to the Home.

> **Note**
>
> Typically, this option is used by the Snoopee when data can be forwarded to the Requester while also providing a copy of data to the Home. For example, when the Snoopee holds a Dirty copy of the cache line, but the data returned to the Requester must be Clean. This option can also happen when the Home has requested a copy of the data.

Alt 5c. Failed via RSP channel, must use alternative
The Snoopee provides a snoop response, SnpResp, to the Home. The Home must use another alternative described in this section to complete the transaction to the Requester.

Alt 5d. Failed via DAT channel, must use alternative
The Snoopee provides a snoop response with data, SnpRespData or SnpRespDataPtl, to the Home. The Home must use another alternative described in this section to complete the transaction to the Requester.

6. **MakeReadUnique only**

   The Home returns a completion response, Comp, to the Requester. This option is only applicable for a MakeReadUnique transaction when a read data message is not required.

- The transaction ends when the Requester sends a completion acknowledge, CompAck, to the Home. The CompAck must only be sent after a CompData or RespSepData is received. It is permitted, but not required, to wait for DataSepResp before sending CompAck.

B2.3.1.2 Non-allocating Read

Figure B2.2 shows the possible transaction flows for a Non-allocating Read transaction.

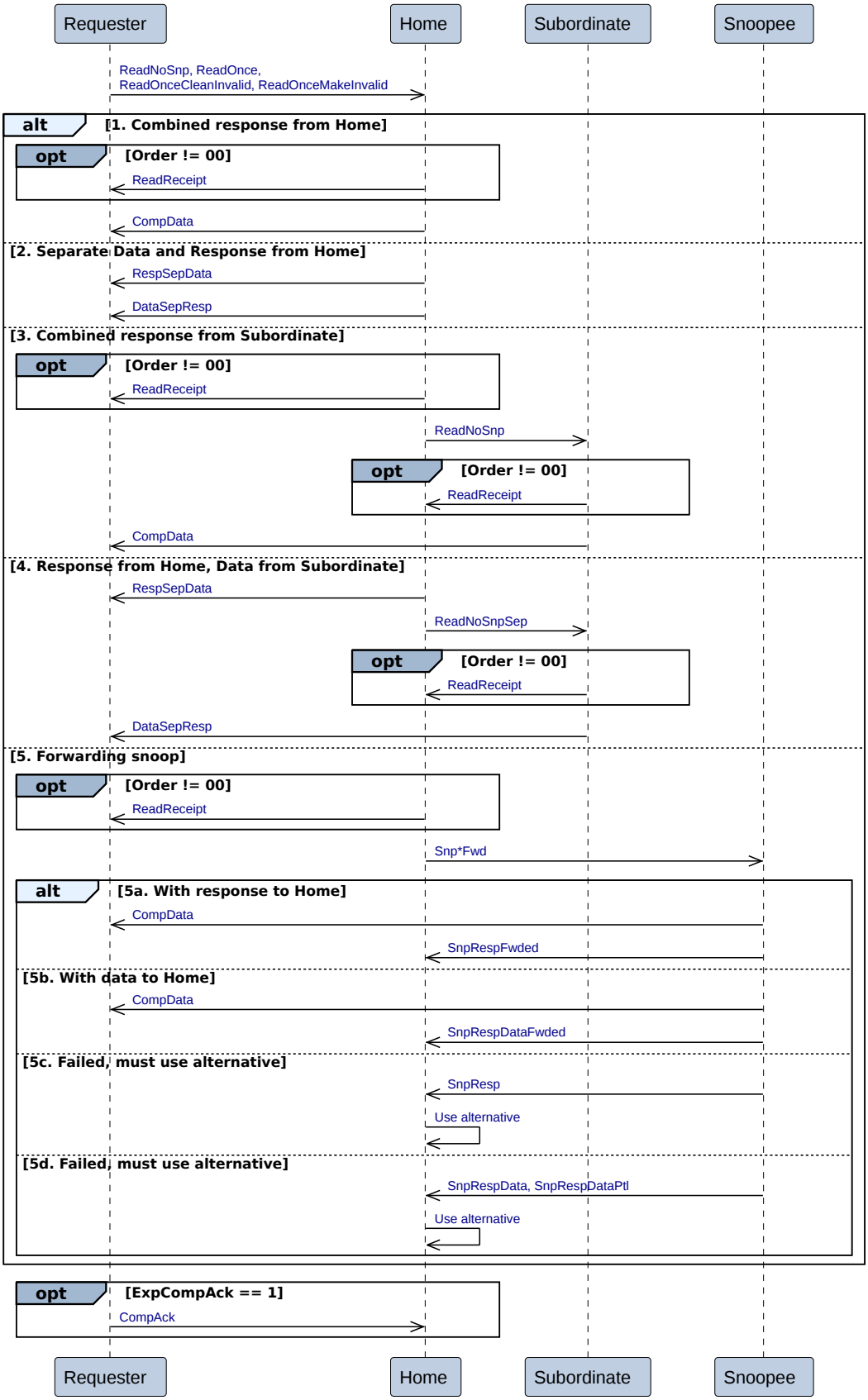

Figure B2.2: Non-allocating Read

The sequence for the Non-allocating Read transactions is:

- The transaction starts with the Requester issuing a Read request to the Home. The Non-allocating Read transactions are:
  - ReadNoSnp
  - ReadOnce
  - ReadOnceCleanInvalid
  - ReadOnceMakeInvalid

  The request contains the following fields which affect the transaction flow:
  - Order
  - ExpCompAck
- Alternatives 1-6 show how the Home can process the transaction. For a description of the typical use of the different alternatives, see B2.3.1.1 *Allocating Read*.

  1. **Combined response from Home**
     - Optionally, when the original request has an ordering requirement, the Home returns a read receipt, ReadReceipt, to the Requester.
     - The Home returns a combined response and read data, CompData, to the Requester.
  2. **Separate data and response from Home**

     The Home returns a separate response, RespSepData, and read data, DataSepResp, to the Requester. This alternative cannot be used if the request has an ordering requirement and a completion acknowledge is not required.
  3. **Combined response from Subordinate**
     - Optionally, when the original request has an ordering requirement, the Home returns a read receipt, ReadReceipt, to the Requester.
     - The Home sends a downstream read request, ReadNoSnp, to the Subordinate.
     - Optionally, when the Home requests a ReadReceipt response, the Subordinate returns ReadReceipt to the Home. The Home must do this when a completion acknowledge is not required. It is permitted, but not required, for the Home to wait for ReadReceipt from the Subordinate before returning ReadReceipt to the Requester.
     - The Subordinate returns a combined response and read data, CompData, to the Requester.

     This alternative cannot be used if the request has an ordering requirement and a completion acknowledge is not required.
  4. **Response from Home, data from Subordinate**
     - The Home returns a separate response, RespSepData, to the Requester and sends a downstream read data only request, ReadNoSnpSep, to the Subordinate.
     - Optionally, when the Home requests a ReadReceipt response, the Subordinate returns a read receipt, ReadReceipt, to the Home. The Home must request a ReadReceipt unless the original request indicates a requirement for both ordering and a completion acknowledge. It is permitted, but not required, for the Home to wait for ReadReceipt from the Subordinate before returning RespSepData to the Requester.
     - The Subordinate returns read data, DataSepResp, to the Requester.

     This alternative cannot be used if the request has an ordering requirement and a completion acknowledge is not required.
  5. **Forwarding snoop**

- Optionally, when the original request has an ordering requirement, the Home returns a read receipt, ReadReceipt, to the Requester.
- The Home requests a Snoopee to forward read data, Snp*Fwd, to the Requester. See B4.4 *Request transactions and corresponding Snoop requests* to determine which Snp*Fwd transactions can be used.
- The alternatives, 5a-5d, show how the Snoopee can process the transaction:

  Alt 5a. With response to Home

  - The Snoopee provides a combined response and read data, CompData, to the Requester.
  - The Snoopee provides a snoop response, SnpRespFwded, to the Home.

  Alt 5b. With data to Home

  - The Snoopee provides a combined response and read data, CompData, to the Requester.
  - The Snoopee provides a Snoop response with data, SnpRespDataFwded, to the Home.

  Alt 5c. Failed, must use alternative

  - The Snoopee provides a snoop response, SnpResp, to the Home.
  - The Home must use another alternative described in this section to complete the transaction to the Requester.

  Alt 5d. Failed, must use alternative

  - The Snoopee provides a snoop response with data, SnpRespData or SnpRespDataPtl, to the Home.
  - The Home must use another alternative described in this section to complete the transaction to the Requester.

- If the original request has ExpCompAck asserted, the Requester must only provide a CompAck response after the following:
- At least one CompData packet is received.
- RespSepData, if the request does not have an ordering requirement. The request is permitted, but not required, to wait for DataSepResp.
- RespSepData and at least one DataSepResp packet, if the request has an ordering requirement.

If the original request has an ordering requirement, it is permitted, but not required, for the Requester to wait for ReadReceipt before sending CompAck.

Table B2.6 lists the permitted DMT and DCT transactions for ReadNoSnp and ReadOnce* from a Request Node. The following key is used:

Y Yes, permitted

N No, not permitted

- The flow is not used in the transaction

Table B2.6: Permitted DMT and DCT for ReadNoSnp and ReadOnce* from a Request Node

Order[1:0]	ExpCompAck	DMT	DCT	Notes
00	0	Y	Y	Home does not need to be notified of transaction completion. For DMT, Home must obtain a ReadReceipt from Subordinate Node to ensure the request to Subordinate Node is not given a RetryAck response.

Continued on next page

Table B2.6 – *Continued from previous page*

Order[1:0]	ExpCompAck	DMT	DCT	Notes
	1	Y	Y	When not using DMT, Home does not need to be notified of transaction completion. For DMT, to ensure that the request from the Home to the Subordinate Node is not given a RetryAck response: • When using ReadNoSnp to the Subordinate Node, Home must either obtain a ReadReceipt from the Subordinate Node or wait for the CompAck response from the Request Node. • When using ReadNoSnpSep to the Subordinate Node, Home must obtain a ReadReceipt from the Subordinate Node.
01	-	-	-	Not permitted.
10 11	0	N	Y	For DCT, Home uses the SnpRespFwded or SnpRespDataFwded snoop response to determine transaction completion.
	1	Y	Y	For DMT, Home uses the CompAck response to determine transaction completion. For DCT, Home uses the SnpRespFwd or SnpRespDataFwded snoop response to determine transaction completion.

For partial ReadNoSnp or ReadOnce* transactions, that is, where the size is less than 64B:

- The Home cannot use a DCT flow.
- If a DMT flow is used to forward data directly from Subordinate to Requester, the Home must use a partial ReadNoSnp request to the Subordinate.
- If the Home does not request a DMT flow, a full cache line or partial cache line ReadNoSnp can be used. The Home must only return the requested number of DAT packets to the Requester, regardless of the number of DAT packets it receives from the Subordinate. For more information, see B2.8.4 *Data packetization*.

B2.3.2 Write transactions

Write transactions are grouped into the following types:

- B2.3.2.1 *Immediate Write*
- B2.3.2.2 *Write Zero*
- B2.3.2.3 *CopyBack Write*
- B2.3.2.4 *Combined Immediate Write and CMO*
- B2.3.2.5 *Combined Immediate Write and Persist CMO*
- B2.3.2.6 *Combined CopyBack Write and CMO*

B2.3.2.1 Immediate Write

Figure B2.3 shows the possible transaction flows for an Immediate Write transaction.

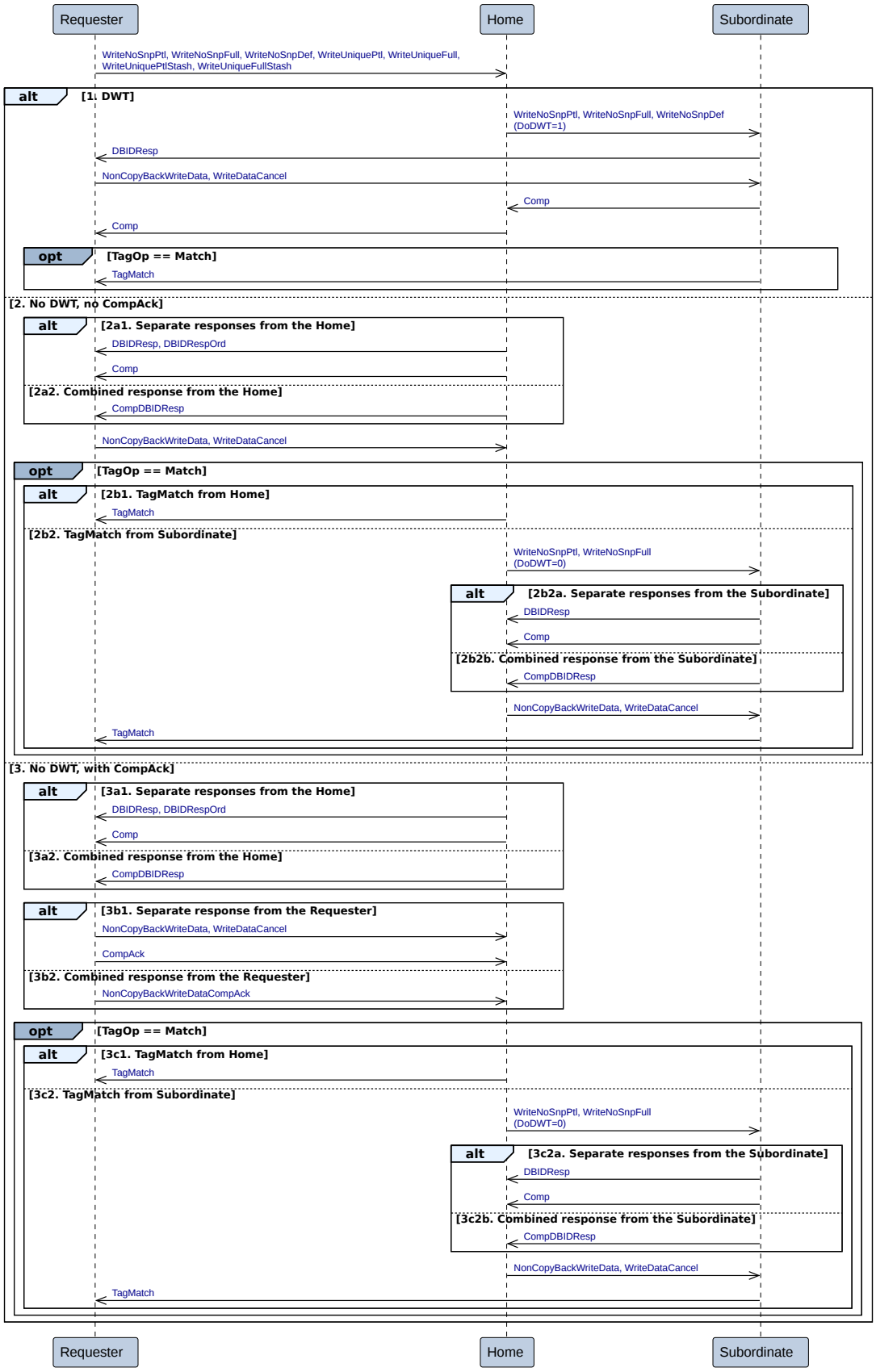

Figure B2.3: Immediate Write

The sequence for Immediate Write transactions is:

- The transaction starts with the Requester issuing an Immediate Write request to the Home. The Immediate Write transactions are:
  - WriteNoSnpPtl
  - WriteNoSnpFull
  - WriteNoSnpDef
  - WriteUniquePtl
  - WriteUniqueFull
  - WriteUniquePtlStash
  - WriteUniqueFullStash

  Snoop requests that are generated to complete these transactions are considered independent transactions from the Home and are not shown in this flow. Write requests that are generated to downstream Subordinates, which are not part of the DWT flow, are considered as independent transactions and are not shown in this flow. See B2.3.9 *Home Initiated transactions* for more details of independent transactions from the Home. Stash snoops that are generated to complete the WriteUniquePtlStash or WriteUniqueFullStash transactions are described in the later section on Stash transactions, see B2.3.4 *Stash transactions*.

  The request contains the following fields which affect the transaction flow:

  - ExpCompAck
  - TagOp

- The Home can choose to complete the transaction using DWT or without DWT. The remainder of the transaction flow depends on whether the original request requires a completion acknowledge response, as determined by the ExpCompAck field. The combinations are described in alternatives 1-3:

  1. **DWT**

     The Home uses DWT.

     - The Home sends a downstream write request, WriteNoSnpPtl, WriteNoSnpFull, or WriteNoSnpDef, with DoDWT = 1 to the Subordinate.
     - The Subordinate returns a data request, DBIDResp, to the Requester.
     - The Requester sends write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, to the Subordinate. The Requester must only send this after receiving DBIDResp.
     - The Subordinate returns a completion response, Comp, to the Home. It is permitted, but not required, for the Subordinate to wait for write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, from the Requester before returning Comp to the Home.
     - The Home returns a completion response, Comp, to the Requester. It is permitted, but not required, for the Home to wait for Comp from the Subordinate before returning Comp to the Requester.
     - Optionally, when the request requires a TagMatch response, the Subordinate returns a tag match response, TagMatch, to the Requester. It is permitted, but not required, to wait for write data before returning TagMatch.

  2. **No DWT, no CompAck**

     The Home does not use DWT for a request that does not require a completion acknowledge, CompAck.

     - The Home has two alternatives to send the completion response and the data request response to the Requester.

       Alt 2a1. Separate responses from the Home
       The Home does both the following:

       - Returns a data request, DBIDResp or DBIDRespOrd, to the Requester.

– Returns a completion response, Comp, to the Requester. It is permitted, but not required, to wait for write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, before returning Comp.

Alt 2a2. Combined response from the Home
The Home returns a combined data request and completion response, CompDBIDResp, to the Requester.

– The Requester sends write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, to the Home. The Requester must only send this after receiving DBIDResp, DBIDRespOrd, or CompDBIDResp.

– Optionally, when the request requires a TagMatch response, the Home has two alternatives.

Alt 2b1. TagMatch from Home
The Home returns a tag match response, TagMatch, to the Requester. It is permitted, but not required, to wait for write data before returning TagMatch.

Alt 2b2. TagMatch from Subordinate

– The Home sends a downstream write request, WriteNoSnpPtl or WriteNoSnpFull, with DoDWT = 0 to the Subordinate. The Subordinate has two alternatives to send return data request and completion response to the Home.

Alt 2b2a. Separate responses from the Subordinate
The Subordinate does both the following:

– Returns a data request, DBIDResp, to the Home.

– Returns a completion response, Comp, to the Home.
It is permitted, but not required, for the Subordinate to wait for write data from the Home before returning Comp to the Home.

Alt 2b2b. Combined response from the Subordinate
The Subordinate returns a combined data request and completion response, CompDBIDResp, to the Home.

– The Home sends write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, to the Subordinate. The Home must only send this after receiving DBIDResp or CompDBIDResp.

– The Subordinate returns a tag match response, TagMatch, to the Requester. It is permitted, but not required, to wait for write data before returning TagMatch.

3. **No DWT, with CompAck**

The Home does not use DWT for a request that does require a completion acknowledge, CompAck.

- The Home has two alternatives to return the completion response and the data request response to the Requester.

Alt 3a1. Separate response from the Home
The Home does both the following:

– Returns a data request, DBIDResp or DBIDRespOrd, to the Requester.

– Returns a completion response, Comp, to the Requester.

It is permitted, but not required, to wait for write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, before returning Comp.

Alt 3a2. Combined response from the Home
The Home returns a combined data request and completion response, CompDBIDResp, to the Requester.

- The Requester has two alternatives to send write data and completion acknowledge to the Home.

  Alt 3b1. Separate response from the Requester
  The Requester does both the following:

  – Sends write data, NonCopyBackWriteData, or a write cancellation, WriteDataCancel, to the Home.
  The Requester must only send this after receiving DBIDResp, DBIDRespOrd, or CompDBIDResp.

  – Sends a completion acknowledge, CompAck, to the Home. The Requester must only send this after it has received DBIDResp, DBIDRespOrd, CompDBIDResp, or Comp. It is permitted, but not required, to wait for DBIDResp or DBIDRespOrd before sending CompAck. It is not permitted to wait for Comp before sending CompAck. It is permitted, but not expected, to wait for TagMatch before returning CompAck.

  Alt 3b2. Combined response from the Requester
  The Requester sends a combined write data and completion acknowledge, NonCopyBackWriteDataCompAck, to the Home.
  The Requester must only send this after it has received DBIDResp, DBIDRespOrd, or CompDBIDResp.
  It is not permitted to wait for Comp before sending NCBWRrDataCompAck if DBIDResp or DBIDRespOrd have been received.

- Optionally, when the request requires a TagMatch response, the Home has two alternatives to return the response.

  Alt 3c1. TagMatch from Home
  The Home returns a tag match response, TagMatch, to the Requester. It is permitted, but not required, to wait for write data before returning TagMatch.

  Alt 3c2. TagMatch from Subordinate

  – The Home sends a downstream write request, WriteNoSnpPtl or WriteNoSnpFull, with DoDWT = 0 to the Subordinate. The Subordinate has two alternatives to return data request and completion response to the Home.

    Alt 3c2a. Separate responses from Subordinate
    The Subordinate does both the following:

    – Returns a data request, DBIDResp, to the Home.

    – Returns a completion response, Comp, to the Home.
    It is permitted, but not required, for the Subordinate to wait for write data before sending Comp to the Home.

    Alt 3c2b. Combined response from Subordinate
    The Subordinate returns a combined data request and completion response, CompDBIDResp, to the Home.

  – The Home sends write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, to the Subordinate.
  The Home must only send this after receiving DBIDResp or CompDBIDResp.

  – The Subordinate returns a tag match response, TagMatch, to the Requester. It is permitted, but not required, to wait for write data before returning TagMatch.

The Completer of a Write transaction is permitted to return a Comp response when a WriteDataCancel response is received without dependency on either the processing of the write request or the completion of any snoops sent due to the write.

B2.3.2.2 Write Zero

Figure B2.4 shows the possible transaction flows for a Write Zero transaction.

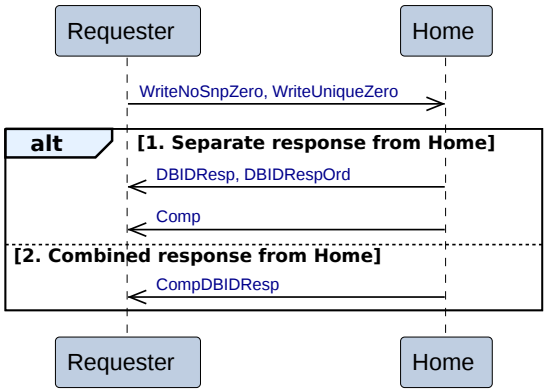

Figure B2.4: Write Zero

- The transaction starts with the Requester issuing a Write Zero request to the Home. The Write Zero transactions are:
  - WriteUniqueZero
  - WriteNoSnpZero
- The Home has two alternatives to send the completion response and the data request response to the Requester.
  1. **Separate response from Home**
     - The Home returns a data request response, DBIDResp or DBIDRespOrd, to the Requester.
     - The Home returns a completion response, Comp, to the Requester.
  2. **Combined response from Home**

     The Home returns a combined data request and completion response, CompDBIDResp, to the Requester.

B2.3.2.3 CopyBack Write

Figure B2.5 shows the possible transaction flows for a CopyBack Write transaction.

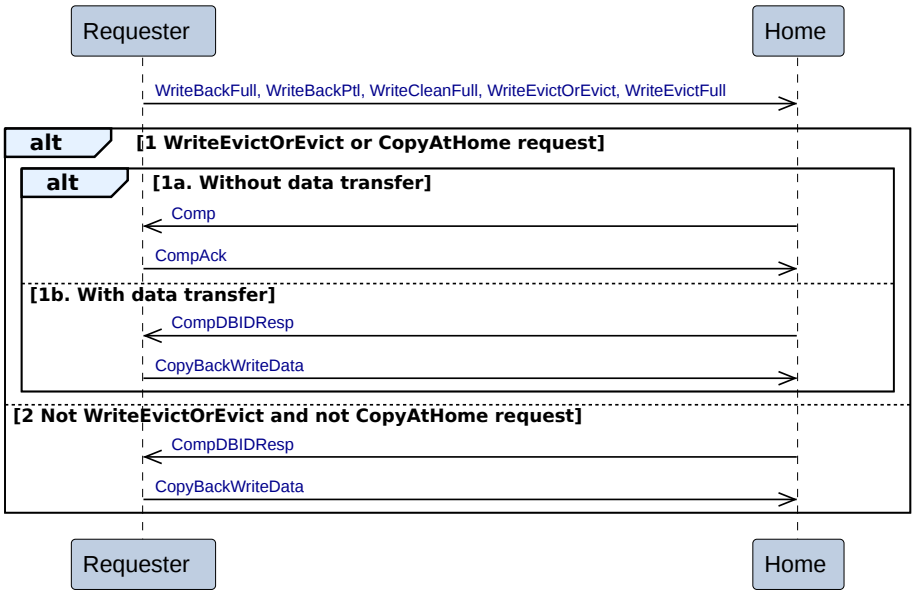

Figure B2.5: CopyBack Write

The sequence for CopyBack Write is:

- The transaction starts with the Requester issuing a CopyBack Write request to the Home. The CopyBack Write transactions are:
  - WriteBackPtl
  - WriteBackFull
  - WriteCleanFull
  - WriteEvictFull
  - WriteEvictOrEvict

  The request contains the following fields which affect the transaction flow:
  - CAH
  - Opcode
- The Home can choose to complete the transaction with a Comp or CompDBIDResp response. The choice of response from the Home is determined by the request type and CAH value. The combinations are described in alternatives 1-2:

  1. **WriteEvictOrEvict or CopyAtHome request**

     The request is WriteEvictOrEvict or the CAH bit value in the request is set to 1.

     The Home has two alternative responses to return to the Requester:

     Alt 1a. Without data transfer

     - The Home returns a completion response, Comp, to the Requester to avoid the data transfer.
     - The Requester sends a completion acknowledge, CompAck.
       The Requester must send this regardless of the ExpCompAck value in the original request and only after receiving the Comp response.

     Alt 1b. With data transfer

     - The Home returns a combined data request and completion response, CompDBIDResp, to the Requester.

– The Requester sends write data, CopyBackWriteData, to the Home.
The Requester must only send this after receiving the CompDBIDResp response.

2. **Not WriteEvictOrEvict and not CopyAtHome request**

The request is not WriteEvictOrEvict and the CAH bit value in the request is set to 0.

– The Home sends a combined data request and completion response, CompDBIDResp, to the Requester.
– The Requester sends write data, CopyBackWriteData, to the Home. The Requester must only send this after receiving the CompDBIDResp response.

B2.3.2.4 Combined Immediate Write and CMO

Figure B2.6 shows the possible transaction flows for a Combined Write and CMO transaction. This only covers Non-persist Cache Maintenance Operations, see B2.3.2.5 *Combined Immediate Write and Persist CMO* for transaction flows for a Combined Immediate Write and Persist CMO transaction.

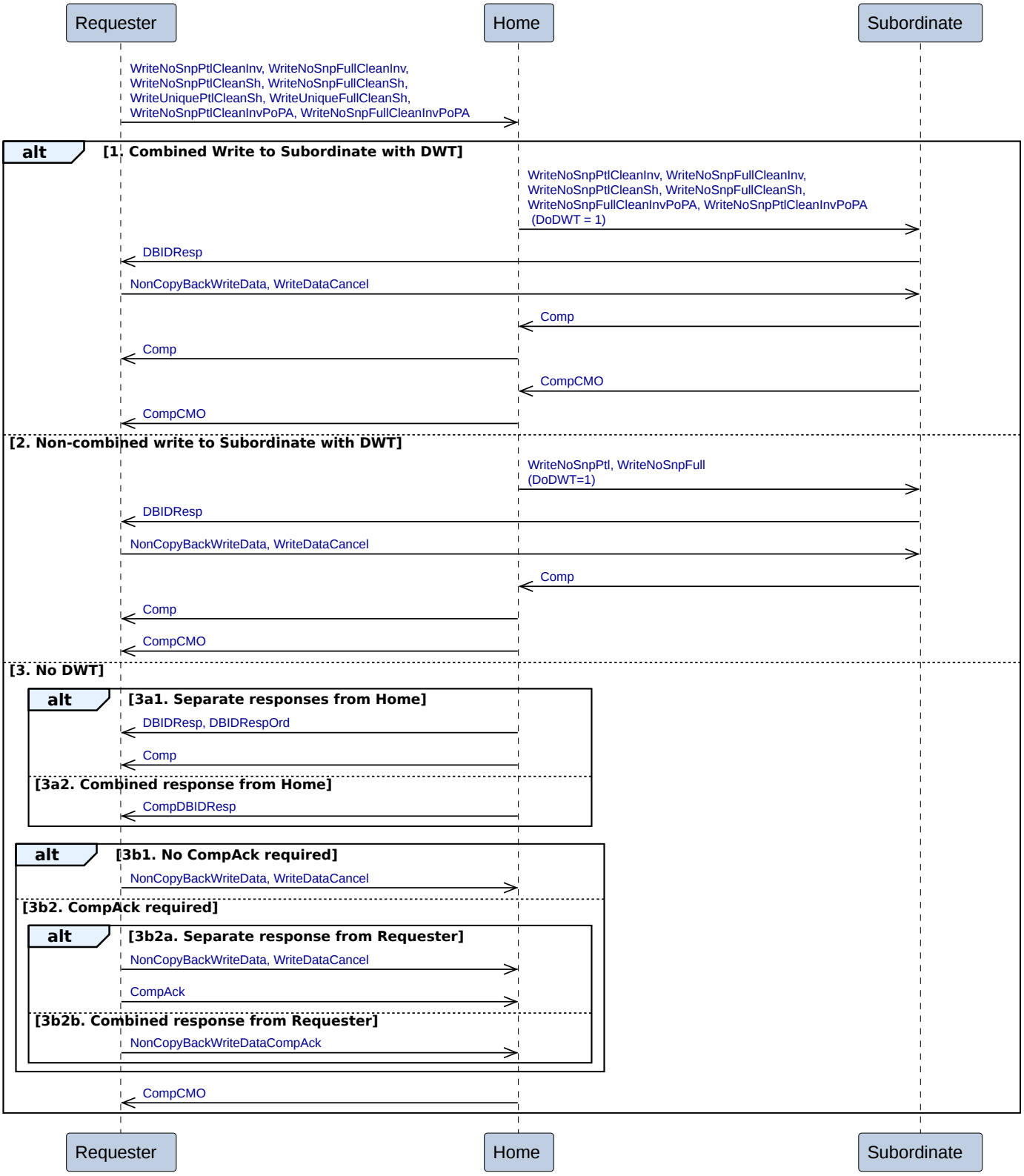

Figure B2.6: Combined Immediate Write and CMO

The sequence for the Combined Immediate Write with CMO transactions is:

- The transaction starts with the Requester issuing a Combined Write and CMO request to the Home. The Combined Immediate Write and CMO transactions are:
  - WriteNoSnpPtlCleanInv
  - WriteNoSnpPtlCleanSh
  - WriteNoSnpFullCleanInv
  - WriteNoSnpFullCleanSh
  - WriteUniquePtlCleanSh
  - WriteUniqueFullCleanSh
  - WriteNoSnpPtlCleanInvPoPA
  - WriteNoSnpFullCleanInvPoPA

  Snoop requests that are generated to complete these transactions are considered as independent transactions from the Home and are not shown in this flow. Write requests that are generated to downstream Subordinates, which are not part of the DWT flow, are considered as independent transactions and are not shown in this flow. Also, CMO requests that only return responses to the Home and are generated to downstream Subordinates are considered independent transactions. See B2.3.9 *Home Initiated transactions* for more details on independent transactions from the Home. See B2.3.9.2 *Home to Subordinate Write transactions* for more details on independent Combined Write and CMO transactions from the Home.

  The request contains the following fields which affect the transaction flow:

  - Opcode
  - ExpCompAck

  Note

  A TagOp value of Match is not permitted in a Combined Immediate Write and CMO transaction, therefore no TagMatch responses are permitted and the TagOp field does not affect the transaction flow.

- The Home has three alternatives to choose from to complete the transaction:
  - Combined Write to Subordinate with DWT.
  - Non-combined Write to Subordinate with DWT.
  - Without DWT.

  The three approaches are described in the alternatives 1-3.

- The remainder of the transaction flow depends on whether the original request required a completion acknowledge, as determined by the ExpCompAck.

  1. **Combined Write to Subordinate with DWT**

     The Home uses a Combined Write with DWT.

     - The Home sends a downstream combined write request with DoDWT = 1 to the Subordinate.
     - The Subordinate returns a data request, DBIDResp, to the Requester.
     - The Requester sends write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, to the Subordinate. The Requester must only send this after receiving DBIDResp.
     - The Subordinate returns a completion response, Comp, to the Home. It is permitted, but not required, for the Subordinate to wait for write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, from the Requester before returning Comp to the Home.
     - The Home returns a completion response, Comp, to the Requester. It is permitted, but not required, for the Home to wait for Comp from the Subordinate before returning Comp to the Requester.
     - The Subordinate returns a CMO completion response, CompCMO, to the Home. It is permitted, but not required, for the Subordinate to wait for write data from the Requester before returning CompCMO to the Home.

- The Home returns a CMO completion response, CompCMO, to the Requester. It is permitted, but not required, for the Home to wait for Comp or CompCMO from the Subordinate before returning CompCMO to the Requester. If there is an observer downstream of the Home, the Home must wait for the CompCMO response from the Subordinate before returning CompCMO to the Requester.

2. **Non-combined Write to Subordinate with DWT**

   The Home uses a Non-combined Write with DWT.

   - The Home sends a downstream WriteNoSnpPtl or WriteNoSnpFull, with DoDWT = 1 to the Subordinate.
   - The Subordinate returns a data request, DBIDResp, to the Requester.
   - The Requester sends write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, to the Subordinate. The Requester must only send this after receiving DBIDResp.
   - The Subordinate returns a completion response, Comp, to the Home. It is permitted, but not required, for the Subordinate to wait for write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, from the Requester before returning Comp to the Home.
   - The Home returns a completion response, Comp, to the Requester. It is permitted, but not required, for the Home to wait for Comp from the Subordinate before returning Comp to the Requester.
   - The Home returns a CMO completion response, CompCMO, to the Requester. It is permitted, but not required, for the Home to wait for Comp from the Subordinate before returning CompCMO to the Requester.

3. **No DWT**

   - The Home does not use DWT.
   - The Home has two alternatives to request write data.

     Alt 3a1. Separate responses from Home
     The Home does both the following:

     - Returns a data request, DBIDResp or DBIDRespOrd, to the Requester.
     - Returns a completion response, Comp, to the Requester.
       It is permitted, but not required, to wait for write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, before returning Comp.

     Alt 3a2. Combined response from Home
     The Home returns a combined data request and completion response, CompDBIDResp, to the Requester.

   - The Requester has several alternatives to send write data depending on whether the transaction requires a completion acknowledge.

     Alt 3b1. No CompAck required
     A completion acknowledge, CompAck, is not required and the Requester sends write data, NonCopyBackWriteData, or a write cancellation, WriteDataCancel, to the Home. The Requester must only send this after receiving DBIDResp, DBIDRespOrd, or CompDBIDResp.

     Alt 3b2. CompAck required
     A completion acknowledge is required. The Requester has two alternatives to send write data and completion acknowledge to the Home.

     Alt 3b2a. Separate response from Requester
     The Requester does both the following:

     - Sends write data, NonCopyBackWriteData, or a write cancellation, WriteDataCancel, to the Home. The Requester must only send this after receiving DBIDResp, DBIDRespOrd, or CompDBIDResp.

– Sends CompAck to the Home. The Requester must only send this after it has received DBIDResp, DBIDRespOrd, CompDBIDResp, or Comp. It is permitted, but not required, to wait for DBIDResp or DBIDRespOrd before sending CompAck. It is not permitted to wait for Comp or CompCMO before sending CompAck.

Alt 3b2b. Combined response from Requester
The Requester sends a combined write data and completion acknowledge, NonCopyBackWriteDataCompAck, to the Home. The Requester must only send this after it has received DBIDResp, DBIDRespOrd, or CompDBIDResp. It is not permitted to wait for Comp before sending NonCopyBackWriteDataCompAck if DBIDResp or DBIDRespOrd have been received. It is not permitted to wait for CompCMO before sending NonCopyBackWriteDataCompAck.

– The Home returns a CMO completion response, CompCMO, to the Requester. It is permitted, but not required, for the Home to wait for write data from the Requester before returning CompCMO.

B2.3.2.5 Combined Immediate Write and Persist CMO

Figure B2.7 shows the possible transaction flows for Combined Immediate Write and Persist CMO transaction.

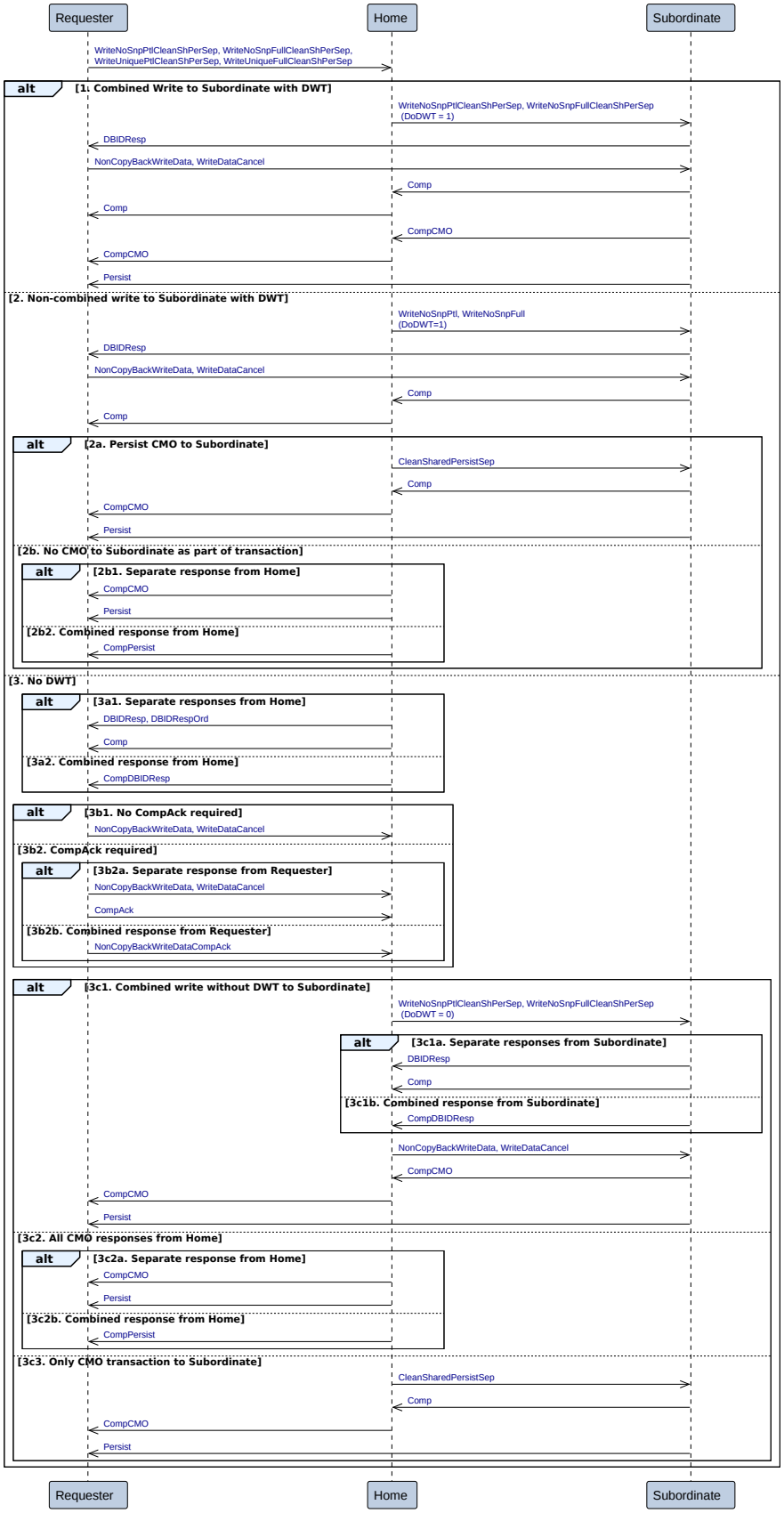

Figure B2.7: Combined Immediate Write and Persist CMO

The sequence for Combined Immediate Write and Persist CMO is:

- The transaction starts with the Requester issuing a Combined Immediate Write and Persist CMO request to the Home.

  The Combined Immediate Write and Persist CMO transactions are:

  – WriteNoSnpPtlCleanShPerSep
  – WriteNoSnpFullCleanShPerSep
  – WriteUniquePtlCleanShPerSep
  – WriteUniqueFullCleanShPerSep

  Snoop requests that are generated to complete these transactions are considered as independent transactions from the Home and are not shown in this flow. Write requests that are generated to downstream Subordinates, which are not part of the DWT flow, are considered as independent transactions and are not shown in this flow. Also, CMO requests that only return responses to the Home and are generated to downstream Subordinates are considered independent transactions. See B2.3.9 *Home Initiated transactions* for more details on independent transactions from the Home. See B2.3.9.2 *Home to Subordinate Write transactions* for more details on independent Combined Write and CMO transactions from the Home.

  The request contains the following fields which affect the transaction flow:

  – Opcode
  – ExpCompAck

  Note

  A TagOp value of Match is not permitted in a Combined Immediate Write and Persist CMO transaction. Therefore, no TagMatch responses are permitted and the TagOp field does not affect the transaction flow.

- The Home can choose to complete the transaction using a:

  – Combined Write to the Subordinate with DWT.
  – Non-combined Write to the Subordinate with DWT.
  – Without DWT.

  The three approaches are described in alternatives 1-3.

  The remainder of the transaction flow depends on whether the original request required a completion acknowledge, as determined by the ExpCompAck field.

  1. **Combined Write to Subordinate with DWT**

     – The Home sends a downstream combined write request with DoDWT = 1 to the Subordinate.

     – The Subordinate returns a data request, DBIDResp, to the Requester.

     – The Requester sends write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, to the Subordinate. The Requester must only send this after receiving DBIDResp.

     – The Subordinate returns a completion response, Comp, to the Home. It is permitted, but not required, for the Subordinate to wait for write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, from the Requester before returning Comp to the Home.

     – The Home returns a completion response, Comp, to the Requester. It is permitted, but not required, for the Home to wait for Comp from the Subordinate before returning Comp to the Requester.

     – The Subordinate returns a CMO completion response, CompCMO, to the Home. It is permitted, but not required, for the Subordinate to wait for write data from the Requester before returning CompCMO to the Home.

- The Home returns a CMO completion response, CompCMO, to the Requester. It is permitted, but not required, for the Home to wait for Comp or CompCMO from the Subordinate before returning CompCMO to the Requester. If there is an observer downstream of the Home, the Home must wait for the CompCMO response from the Subordinate before returning CompCMO to the Requester.
- The Subordinate returns a persist response, Persist, to the Requester. It is permitted, but not required, to wait for write data before returning Persist.

2. **Non-combined Write to Subordinate with DWT**
   - The Home sends a downstream Non-combined Write request, WriteNoSnpPtl or WriteNoSnpFull, with DoDWT = 1 to the Subordinate.
   - The Subordinate returns a data request, DBIDResp, to the Requester.
   - The Requester sends write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, to the Subordinate. The Requester must only send this after receiving DBIDResp.
   - The Subordinate returns a completion response, Comp, to the Home. It is permitted, but not required, for the Subordinate to wait for write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, from the Requester before returning Comp to the Home.
   - The Home returns a completion response, Comp, to the Requester. It is permitted, but not required, for the Home to wait for Comp from the Subordinate before returning Comp to the Requester.
   - The Home has two alternatives to send the CMO responses to the Requester.

     Alt 2a. Persist CMO to Subordinate
     - The Home sends a downstream request, CleanSharedPersistSep, to the Subordinate.
     - The Subordinate returns a completion response, Comp, to the Home.
     - The Home returns a CMO completion response, CompCMO, to the Requester.
       If there is an observer downstream of the Home, the Home must wait for the Comp response from the Subordinate before returning CompCMO to the Requester.
     - The Subordinate returns a persist response, Persist, to the Requester.

     Alt 2b. No CMO to Subordinate as part of transaction
     The Home sends all the CMO responses to the Requester. All CMO responses can be sent from the Home in two alternative ways.

     Alt 2b1. Separate responses from Home
     The Home does both the following:
     - Returns a CMO completion response, CompCMO, to the Requester.
     - Returns a persist response, Persist, to the Requester.

     Alt 2b2. Combined response from Home
     The Home returns a combined completion and persist response, CompPersist, to the Requester.

3. **No DWT**

   The Home does not use DWT.
   - The Home has two alternatives to request write data.

     Alt 3a1. Separate responses from Home
     The Home does both of the following:
     - Returns a data request, DBIDResp or DBIDRespOrd, to the Requester.
     - Returns a completion response, Comp, to the Requester. It is permitted, but not required, to wait for write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, before returning Comp.

Alt 3a2. Combined response from Home
The Home returns a combined data request and completion response, CompDBIDResp, to the Requester.

– The Requester has several alternatives to send write data depending on whether the transaction requires a completion acknowledge.

Alt 3b1. No CompAck required
The Requester sends write data, NonCopyBackWriteData, or a write cancellation, WriteDataCancel, to the Home. The Requester must only send this after receiving DBIDResp, DBIDRespOrd, or CompDBIDResp.

Alt 3b2. CompAck required
A completion acknowledge response, CompAck, is required. The Requester has two alternatives to send write data and CompAck to the Home.

Alt 3b2a. Separate responses from Requester
The Requester does both the following:

– Sends write data, NonCopyBackWriteData, or a write cancellation, WriteDataCancel, to the Home.
The Requester must only send this after receiving DBIDResp, DBIDRespOrd, or CompDBIDResp.

– Sends a completion acknowledge, CompAck, to the Home. The Requester must only send this after it has received DBIDResp, DBIDRespOrd, CompDBIDResp, or Comp. It is permitted, but not required, to wait for DBIDResp or DBIDRespOrd before sending CompAck. It is not permitted to wait for Comp, CompCMO, or Persist before sending CompAck.

Alt 3b2b. Combined response from Requester
The Requester sends a combined write data and completion acknowledge, NonCopyBackWriteDataCompAck, to the Home. The Requester must only send this after it has received DBIDResp, DBIDRespOrd, or CompDBIDResp. It is not permitted to wait for Comp before sending NonCopyBackWriteDataCompAck if DBIDResp or DBIDRespOrd have been received. It is not permitted to wait for CompCMO or Persist before sending NonCopyBackWriteDataCompAck.

– The Home has several alternatives to complete the remainder of the transaction.

Alt 3c1. Combined Write without DWT to Subordinate

– The Home sends a Combined Write without DWT to the Subordinate.

– The Subordinate has two alternatives to request write data.

Alt 3c1a. Separate responses from Subordinate
The Subordinate does both the following:

– Returns a data request, DBIDResp, to the Home.

– Returns a completion response, Comp, to the Home.
It is permitted, but not required, to wait for write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, before returning Comp.

Alt 3c1b. Combined response from Subordinate
The Subordinate returns a combined data request and completion response, CompDBIDResp, to the Home.

– The Home sends write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, to the Subordinate. The Home must only send this after receiving DBIDResp or CompDBIDResp.

– The Subordinate returns a CMO completion response, CompCMO, to the Home. It is permitted, but not required, to wait for write data before returning CompCMO.

– The Home returns a CMO completion response, CompCMO, to the Requester. It is permitted, but not required, for the Home to wait for CompCMO from the Subordinate before returning CompCMO to the Requester.

– The Subordinate returns a persist response, Persist, to the Requester. It is permitted, but not required, to wait for write data before returning Persist.

Alt 3c2. All CMO transactions from Home
The Home has two alternatives to send all CMO responses to the Requester.

Alt 3c2a. Separate responses from Home
The Home does both the following:

– Returns a CMO completion response, CompCMO, to the Requester.

– Returns a persist response, Persist, to the Requester.

Alt 3c2b. Combined response from Home
The Home returns a combined completion and persist response, CompPersist, to the Requester.

Alt 3c3. Only CMO transactions to Subordinate

– The Home sends a downstream request, CleanSharedPersistSep, to the Subordinate. Typically, this alternative would only be used where a write to the Subordinate has been previously sent as an independent transaction.

– The Subordinate returns a completion response, Comp, to the Home.

– The Home returns a CMO completion response, CompCMO, to the Requester. If there is an observer downstream of the Home, the Home must wait for the Comp response from the Subordinate before returning CompCMO to the Requester.

– The Subordinate returns a persist response, Persist, to the Requester.

B2.3.2.6 Combined CopyBack Write and CMO

Figure B2.8 shows the possible transaction flows for a Combined CopyBack Write and CMO transaction.

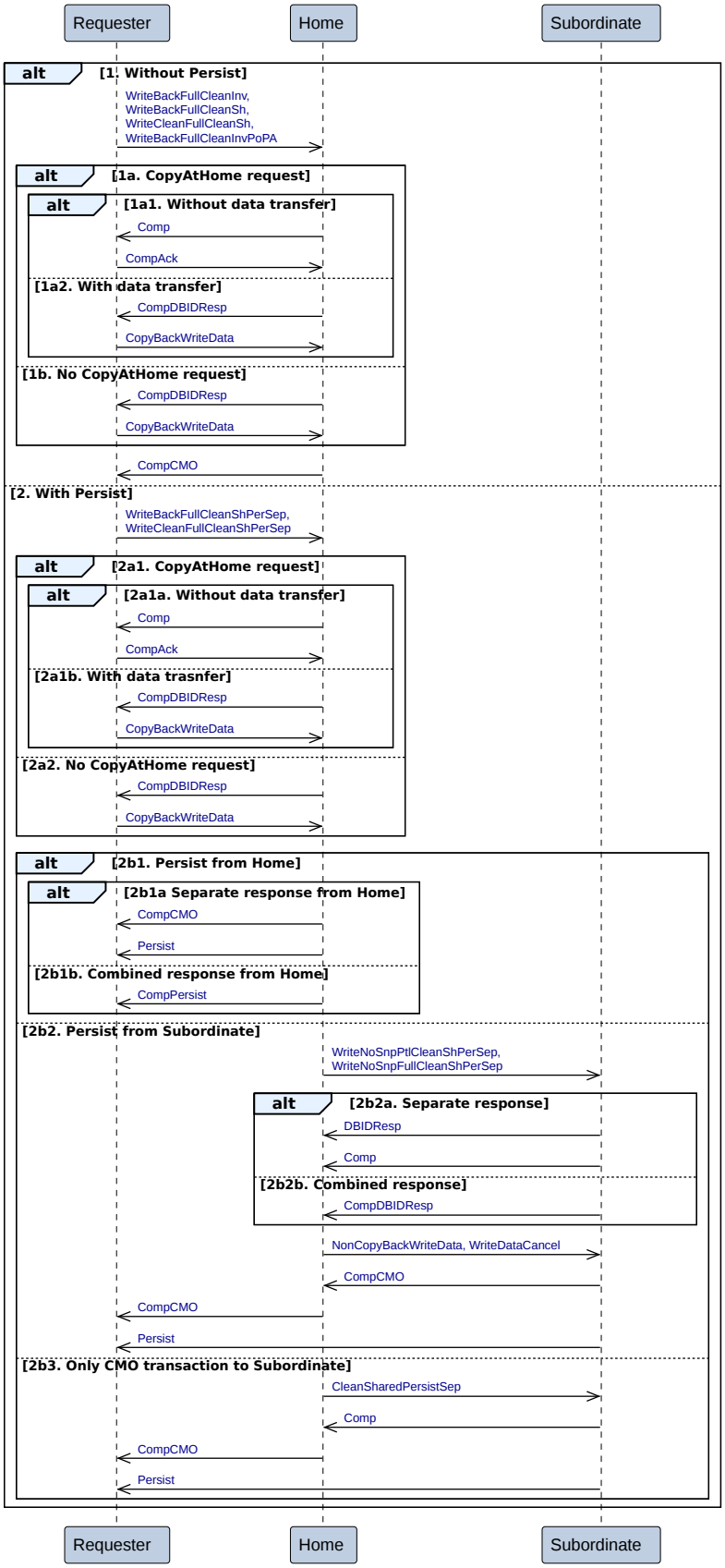

Figure B2.8: Combined CopyBack Write and CMO

There are two possible sequences for Combined CopyBack and CMO transactions.

Write requests that are generated to downstream Subordinates and not part of the DWT or persist flow. These write requests are considered independent transactions and are not shown in this flow. CMO requests that are generated to downstream Subordinates, which only return responses to the Home, are considered as independent transactions and are not shown in this flow. See B2.3.9 *Home Initiated transactions* for more details on independent transactions from the Home. See B2.3.9.4 *Home to Subordinate Combined Write and CMO transactions* for more details on independent Combined Write and CMO transactions from the Home.

The request contains the following field which affects the transaction flow:

- Opcode
- CAH

1. **Without Persist**

   The Combined CopyBack Write and CMO transactions without persist are:

   - WriteBackFullCleanInv
   - WriteBackFullCleanSh
   - WriteCleanFullCleanSh
   - WriteBackFullCleanInvPoPA

   The Requester issues a Combined CopyBack Write and CMO request without Persist response.

   - The Requester issues a request to the Home.

     Alt 1a. CopyAtHome request
     The CAH bit value in the request is set to 1.
     The Home has two alternative responses to return to the Requester.

     Alt 1a1. Without data transfer

     – The Home returns a completion response, Comp, to the Requester to avoid the data transfer.

     – The Requester sends a completion acknowledge, CompAck.
     The Requester must send this regardless of the ExpCompAck value in the original request and only after receiving the Comp response.

     Alt 1a2. With data transfer

     – The Home returns a combined data request and completion response, CompDBIDResp, to the Requester.

     – The Requester sends write data, CopyBackWriteData, to the Home.
     The Requester must only send this after receiving CompDBIDResp.

     Alt 1b. No CopyAtHome request
     The CAH bit value in the request is set to 0.

     – The Home sends a combined data request and completion response, CompDBIDResp, to the Requester.

     – The Requester sends write data, CopyBackWriteData, to the Home. The Requester must only send this after receiving CompDBIDResp.

   - The Home returns a CMO completion response, CompCMO, to the Requester. It is permitted, but not required, to wait for CopyBackWriteData or CompAck before returning CompCMO.

2. **With Persist**

   The Combined CopyBack write and CMO transactions are:

   - WriteBackFullCleanShPerSep
   - WriteCleanFullCleanShPerSep

The Requester issues a Combined CopyBack Write and CMO request with Persist response.

- The Requester issues a request to the Home.

Alt 2a1. CopyAtHome request
The CAH bit value in the request is set to 1. The Home has two alternative responses to return to the Requester.

Alt 2a1a. Without data transfer

– The Home returns a completion response, Comp, to the Requester to avoid the data transfer.

– The Requester sends a completion acknowledge, CompAck. The Requester must send this regardless of the ExpCompAck value in the original request and only after receiving the Comp response.

Alt 2a1b. With data transfer

– The Home returns a combined data request and completion response, CompDBIDResp, to the Requester.

– The Requester sends write data, CopyBackWriteData, to the Home. The Requester must only send this after receiving CompDBIDResp.

Alt 2a2. No CopyAtHome request
The CAH bit value in the request is set to 0.

– The Home sends a combined data request and completion response, CompDBIDResp, to the Requester.

– The Requester sends write data, CopyBackWriteData, to the Home. The Requester must only send this after receiving CompDBIDResp.

The Home has three alternatives to complete the transaction, with the persist response either coming from the Home or from the Subordinate.

Alt 2b1. Persist from Home
The Home has two alternatives to send the CMO completion response and persist response. It is permitted, but not required, for the Home to wait for CopyBackWriteData or CompAck before returning CompCMO, Persist, or CompPersist.

Alt 2b1a. Separate response from Home

– Returns a CMO completion response, CompCMO, to the Requester.

– Returns a persist response, Persist, to the Requester.

Alt 2b2b. Combined response from Home
The Home returns a combined CMO completion response and persist response, CompPersist, to the Requester.

Alt 2b2. Persist from Subordinate
When a Combined Write is sent to the Subordinate and the Persist response is returned to the Requester, the following happens:

– The Home sends a downstream write request, WriteNoSnpPtlCleanShPerSep or WriteNoSnpFullCleanShPerSep, to the Subordinate.
It is permitted, but not required, for the Home to wait for CopyBackWriteData or CompAck before sending the downstream write request.

– The Subordinate has two alternatives to return the completion response and the data request response to the Home.

Alt 2b2a. Separate response
The Subordinate does both the following:

- Returns a data request, DBIDResp, to the Home.
- Returns a completion response, Comp, to the Home. It is permitted, but not required, to wait for write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, before returning Comp.

Alt 2b2b. Combined response
The Subordinate returns a combined data request and completion response, CompDBIDResp, to the Home.

- The Home sends write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, to the Subordinate.
  The Home must only send this after receiving DBIDResp or CompDBIDResp.
- The Subordinate returns a CMO completion response, CompCMO, to the Home.
  It is permitted, but not required, to wait for write data before returning CompCMO.
- The Home returns a CMO completion response, CompCMO, to the Requester.
  It is permitted, but not required, for the Home to wait for CompCMO from the Subordinate before returning CompCMO to the Requester.
- The Subordinate returns a persist response, Persist, to the Requester.
  It is permitted, but not required, to wait for write data before returning Persist.

Alt 2b3. Only CMO transaction to Subordinate
When a persist CMO is sent to the Subordinate and the Persist response is returned to the Requester, the following happens:

- The Home sends a downstream request, CleanSharedPersistSep, to the Subordinate.
  Typically, this alternative would only be used where a write to the Subordinate has been previously sent as an independent transaction or the write has been canceled.
- The Subordinate returns a completion response, Comp, to the Home.
- The Home returns a completion response, CompCMO, to the Requester.
  If there is an observer downstream of the Home, Home must wait for the Comp response from the Subordinate before returning CompCMO to the Requester.
- The Subordinate returns a persist response, Persist, to the Requester.

B2.3.3 Atomic transactions

Figure B2.9 shows the possible transaction flows for an Atomic transaction.

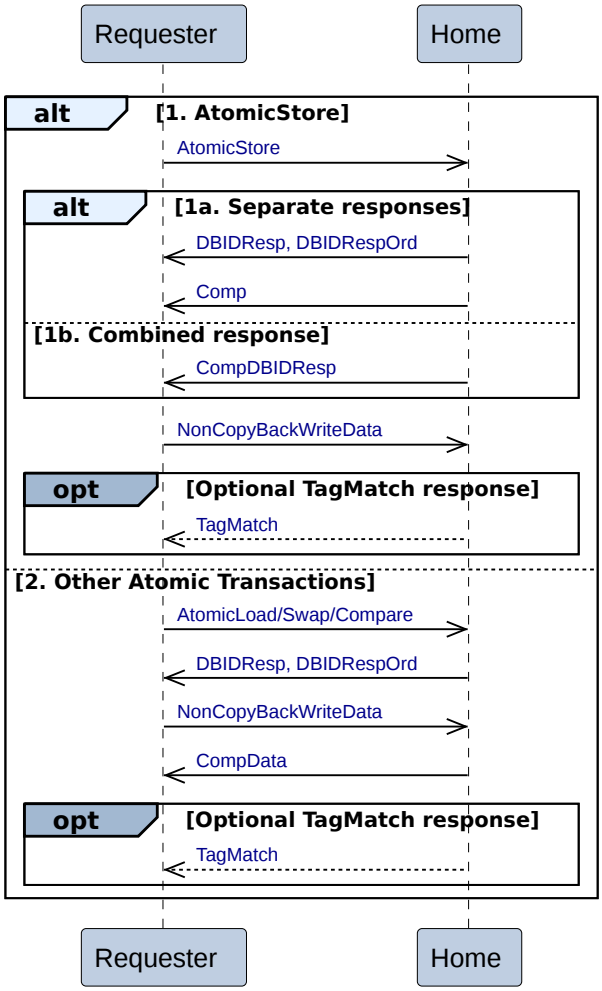

Figure B2.9: Atomic transactions

There are two possible sequences for the Atomic transactions.

The Requester alternatives are:

1. **AtomicStore**

   For an AtomicStore transaction:

   - The Requester sends an AtomicStore request to the Home.
   - The Home has two alternatives to send the completion response and the data request response to the Requester.

     Alt 1a. Separate responses
     The Home does both the following:

     – Returns a data request, DBIDResp or DBIDRespOrd, to the Requester.

     – Returns a completion response, Comp, to the Requester. It is permitted, but not required, to wait for write data before returning Comp.

     Alt 1b. Combined response
     The Home returns a combined data request and completion response, CompDBIDResp, to the Requester.

- The Requester sends write data, NonCopyBackWriteData, to the Home. The Requester must only send this after receiving DBIDResp, DBIDRespOrd, or CompDBIDResp.
- Optionally, when the request requires a TagMatch response, the Home returns a tag match response, TagMatch, to the Requester. It is permitted, but not required, to wait for write data before returning TagMatch.

2. **Other Atomic transactions**

   For an AtomicLoad, AtomicSwap, or AtomicCompare transaction:

   - The Requester sends an AtomicLoad, AtomicSwap, or AtomicCompare request to the Home.
   - The Home sends a data request response, DBIDResp or DBIDRespOrd, to the Requester.
   - The Requester sends write data, NonCopyBackWriteData, to the Home. The Requester must only send this after receiving DBIDResp or DBIDRespOrd. The Requester must not wait to receive CompData before write data is sent.
   - The Home returns a combined data and completion response, CompData, to the Requester. It is permitted, but not required, to wait for write data before returning CompData.
   - Optionally, when the request requires a TagMatch response, the Home returns a tag match response, TagMatch, to the Requester. It is permitted, but not required, to wait for write data before returning TagMatch.

B2.3.4 Stash transactions

Figure B2.10 shows the possible transaction flows for stash transactions.

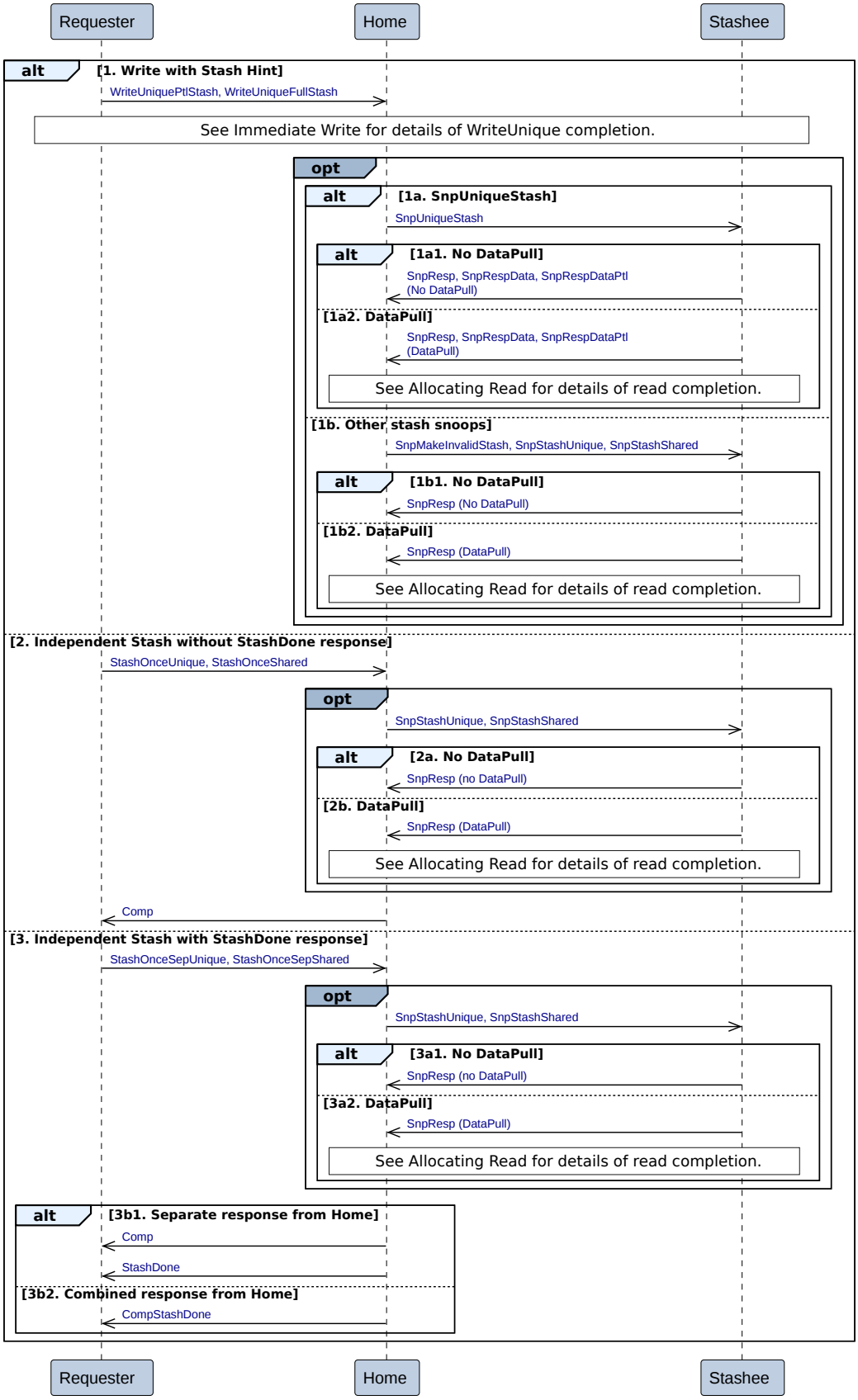

Figure B2.10: Stash transactions

There are three possible sequences for Stash transactions.

The following can affect transaction flow:

- The Home is permitted to ignore the Stash request and not perform any stash snoops.
- The Stashee is permitted to ignore the Stash request and not request a data pull to complete the transaction.
- StashOnceSepUnique and StashOnceSepShared can have a separate StashDone response or combined CompStashDone response.

1. **Write with Stash Hint**

   A write with Stash hint starts the transaction. The write with Stash hint requests are:

   - WriteUniquePtlStash
   - WriteUniqueFullStash

   The WriteUnique uses the same transaction flows as an Immediate Write, see B2.3.2.1 *Immediate Write* for details.

   The Home can optionally send a stash snoop request.

   Alt 1a. SnpUniqueStash

   – The Home sends SnpUniqueStash to the Stashee.
   Typically the Home sends SnpUniqueStash for a partial line write. Other snoops, including other stash snoops, are permitted.

   – The Stashee has two alternatives to respond to the stash snoop request.

   Alt 1a1. No DataPull
   Not request a data pull.

   Alt 1a2. DataPull
   Request a data pull.
   The completion of a request with data pull is identical to the completion of an Allocating Read transaction, see B2.3.1.1 *Allocating Read* for details.

   Alt 1b. Other stash snoops

   – Send SnpMakeInvalidStash, SnpStashShared, or SnpStashUnique, to the Stashee.
   Typically, the Home sends SnpMakeInvalidStash for a full line write. Other snoops, including other stash snoops, are permitted.

   – The Stashee has two alternatives to respond to the stash snoop request.

   Alt 1b1. No DataPull
   Not request a data pull

   Alt 1b2. DataPull
   Request a data pull.
   The completion of a request with data pull is identical to the completion of an Allocating Read transaction, see B2.3.1.1 *Allocating Read* for details.

2. **Independent Stash without StashDone response**

   An independent Stash without a StashDone response starts the transaction. The independent Stash requests without a StashDone response are:

   - StashOnceUnique
   - StashOnceShared

   The Home can optionally send a stash snoop request, SnpStashUnique or SnpStashShared, to the Stashee. Typically the Home sends SnpStashUnique when the original request is StashOnceUnique and SnpStashShared when the original request is StashOnceShared.

- The Stashee has two alternatives to respond to the stash snoop request.

  Alt 2a. No DataPull
  Not request a data pull.

  Alt 2b. DataPull
  Request a data pull.
  The completion of a request with data pull is identical to the completion of an Allocating Read transaction, see B2.3.1.1 *Allocating Read* for details.

The transaction completes with the Home returning a completion response, Comp, to the original Requester.

It is permitted, but not required, to wait for the stash transaction to complete before the Comp response is returned.

3. **Independent Stash with StashDone response**

An independent Stash with a StashDone response starts the transaction. The independent Stash requests with a StashDone response are:

- StashOnceSepUnique
- StashOnceSepShared

The Home can optionally send a stash snoop request, SnpStashUnique or SnpStashShared, to the Stashee. Typically the Home sends SnpStashUnique when the original request is StashOnceSepUnique and SnpStashShared when the original request is StashOnceSepShared.

- The Stashee has two alternatives to respond to the stash snoop request.

  Alt 3a1. No DataPull
  Not request a data pull.

  Alt 3a2. DataPull
  Request a data pull.
  The completion of a request with data pull is identical to the completion of an Allocating Read transaction, see B2.3.1.1 *Allocating Read* for details.

The Home has two alternatives to complete the transaction.

Alt 3b1. Separate responses from Home
The Home does both the following:

– Returns a completion response, Comp, to the Requester.

– Returns a stash done response, StashDone, to the Requester.

Alt 3b2. Combined response from Home
The Home returns a combined completion and stash done response, CompStashDone, to the Requester.

B2.3.5 Dataless transactions

Figure B2.11 shows the transaction flow for a Dataless transaction.

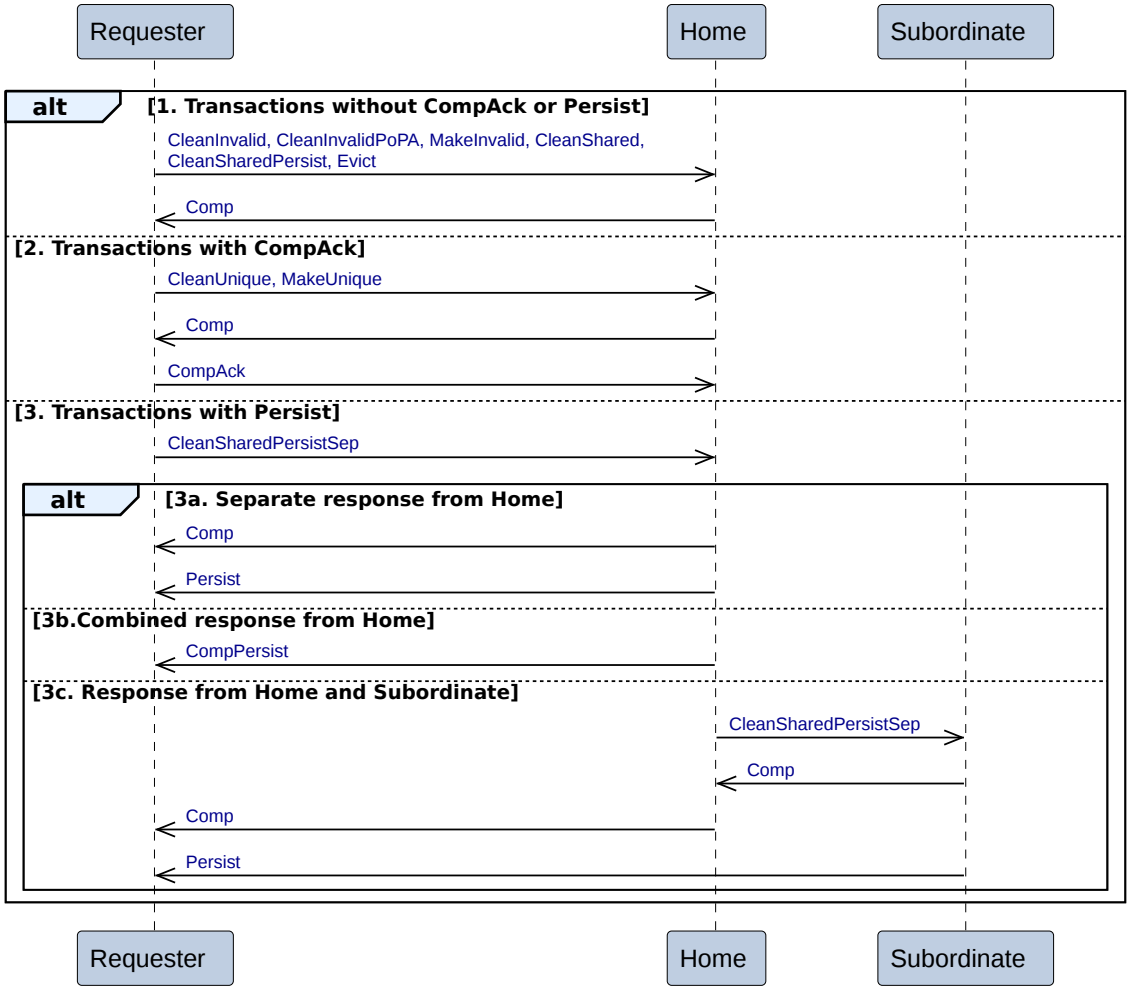

Figure B2.11: Dataless transactions

There are three possible sequences for Dataless transactions.

1. **Transactions without CompAck or Persist**

   The Dataless transactions without CompAck or Persist are:

   - CleanInvalid
   - CleanInvalidPoPA
   - MakeInvalid
   - CleanShared
   - CleanSharedPersist
   - Evict

   The Requester sends the request to the Home.

   The Home returns a completion response, Comp, to the Requester.

2. **Transactions with CompAck**

   The Dataless transactions with CompAck are:

   - CleanUnique
   - MakeUnique

The Requester sends the request to the Home.

The Home returns a completion response, Comp, to the Requester.

The Requester sends a completion acknowledge, CompAck, to the Home.

The Requester must only send this after receiving Comp.

3. **Transactions with Persist**

   The Dataless transaction with Persist is:

   - CleanSharedPersistSep

   The Requester sends the request to the Home.

   The Home has three alternatives to complete the transaction.

   Alt 3a. Separate responses from the Home
   The Home does both the following:

   – Returns a completion response, Comp, to the Requester.

   – Returns a persist response, Persist, to the Requester.

   Alt 3b. Combined response from Home
   The Home returns a combined completion and persist response, CompPersist, to the Requester.

   Alt 3c. Response from Home and Subordinate
   With the Persist response from the Subordinate, the following happens:

   – The Home sends a downstream request, CleanSharedPersistSep, to the Subordinate.

   – The Subordinate returns a completion response, Comp, to the Home.

   – The Home returns a completion response, Comp, to the Requester. If there is an observer downstream of the Home, the Home must wait for the Comp response from the Subordinate before returning the Comp response to the Requester.

   – The Subordinate returns a persist response, Persist, to the Requester.

B2.3.6 Prefetch transactions

Figure B2.12 shows the transaction flow for a Prefetch transaction.

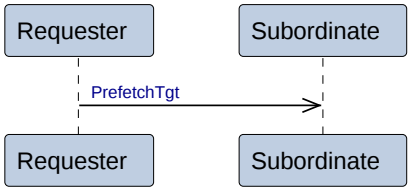

Figure B2.12: Prefetch transactions

The sequence for the Prefetch transaction is:

- The Requester sends a PrefetchTgt request directly to the Subordinate.

> **Note**
>
> No response is given.

B2.3.7 DVM transactions

Figure B2.13 shows the transaction flows for a DVM transaction.

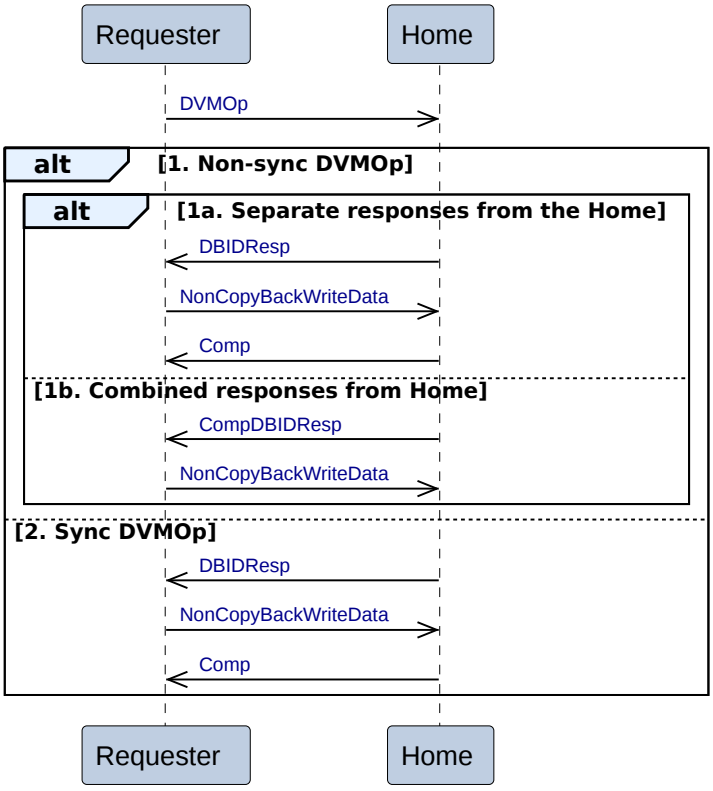

Figure B2.13: DVM transactions

Snoop requests that are generated to complete a DVM transaction are considered as independent transactions from the Home and are not shown in this flow. See B2.3.9.8 *Home to Snoopee DVM transactions* for more details.

The sequence for the DVM transaction is:

- The transaction starts with the Requester issuing a DVMOp request to the Home.
- The Home has two alternatives to send the completion response and the data request response to the Requester:

1. **Non-sync DVMOp**

   Alt 1a. Separate responses from the Home
   The Home does both the following:

   – Returns a data request, DBIDResp, to the Requester.

   – The Requester sends write data, NonCopyBackWriteData, to the Home.
   The Requester must only send this after receiving DBIDResp

   – Returns a completion response, Comp, to the Requester. It is permitted, but not required, to wait for write data before returning Comp.

   Alt 1b. Combined response from the Home

   – The Home returns a combined data request and completion response, CompDBIDResp, to the Requester.

– The Requester sends write data, NonCopyBackWriteData, to the Home. The Requester must only send this after receiving CompDBIDResp.

2. **Sync DVMOp**

   The Home must return separate responses.

   - The Home returns a data request, DBIDResp, to the Requester.
   - The Requester sends write data, NonCopyBackWriteData, to the Home. The Requester must only send this after receiving DBIDResp.
   - The Home returns a completion response, Comp, to the Requester. The Home must only return this after receiving write data.

B2.3.8 Retry

Figure B2.14 shows the possible transaction flows for a Retry sequence.

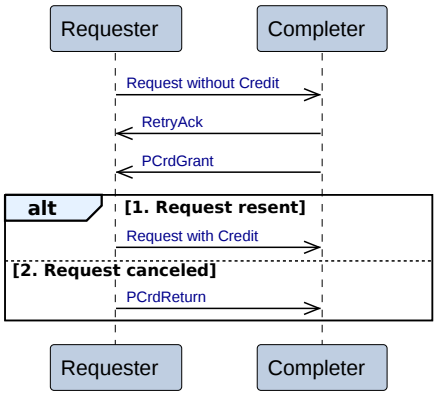

Figure B2.14: Retry transactions

A request transaction is first sent without a *Protocol Credit* (P-Credit). If the transaction cannot be accepted at the Completer, a RetryAck response is given indicating that the transaction is not accepted and can be sent again when an appropriate credit is provided. The transaction includes a credit when it is sent a second time, and is guaranteed to be accepted.

The sequence for the Retry transaction is:

- The Requester issues a request without credit.
- The Completer returns a retry response, RetryAck, to the Requester.
- The Completer returns a protocol credit grant, PCrdGrant, to the Requester. Typically the protocol credit grant is returned a significant time after the Retry response. However, in an atypical case, the PCrdGrant response can be returned before the Retry response.
- The Requester has two alternatives to conclude the Retry sequence. This step must only occur after the Requester has received both RetryAck and PCrdGrant.

  1. **Resend the original request**

     The Requester issues a request with credit.

  2. **Cancel the request and return the credit**

     The Requester sends a protocol credit return, PCrdReturn, to the Completer.

B2.3.9 Home Initiated transactions

The Home Initiated transactions are:

- B2.3.9.1 *Home to Subordinate Read transactions*
- B2.3.9.2 *Home to Subordinate Write transactions*
- B2.3.9.3 *Home to Subordinate Write Zero transactions*
- B2.3.9.4 *Home to Subordinate Combined Write and CMO transactions*
- B2.3.9.5 *Home to Subordinate Dataless transactions*
- B2.3.9.6 *Home to Subordinate Atomic transactions*
- B2.3.9.7 *Home to Snoopee transactions*
- B2.3.9.8 *Home to Snoopee DVM transactions*

B2.3.9.1 Home to Subordinate Read transactions

Figure B2.15 shows the possible transaction flows for a Home to Subordinate Read transaction.

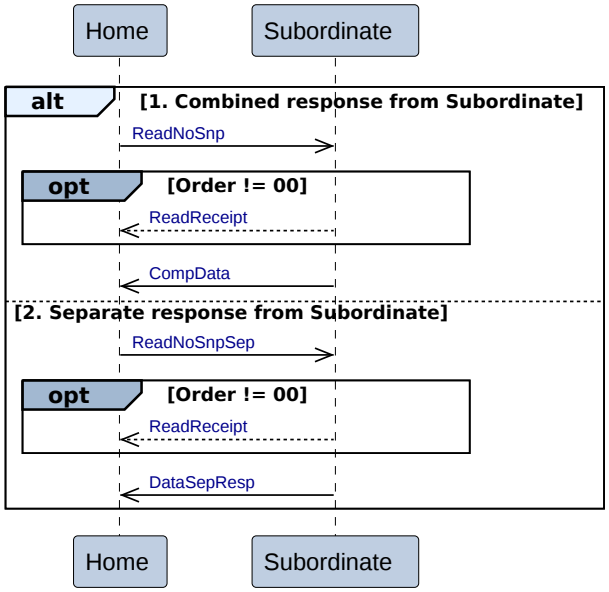

Figure B2.15: Home to Subordinate Read transactions

There are two possible sequences for Home Read transactions.

1. **Combined response from Subordinate**
   - For a ReadNoSnp transaction, the Home issues the request to the Subordinate.
   - Optionally, when the request has Order set to non-zero the Subordinate returns a ReadReceipt response.
   - The Subordinate returns a combined completion response and read data, CompData, to the Home.
2. **Separate responses from Subordinate**
   - For a ReadNoSnpSep transaction, the Home issues the request to the Subordinate.
   - Optionally, when the request has Order set to non-zero, the Subordinate returns a ReadReceipt response.
   - The Subordinate returns read data, DataSepResp, to the Home.

B2.3.9.2 Home to Subordinate Write transactions

Figure B2.16 shows the possible transaction flows for a Home to Subordinate Write transaction.

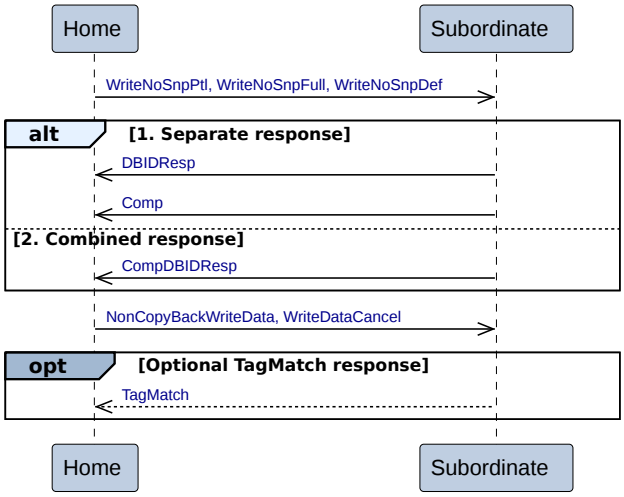

Figure B2.16: Home to Subordinate Write transactions

The sequence for the Home to Subordinate Write transaction is:

- The transaction starts with the Home issuing a WriteNoSnpPtl, WriteNoSnpFull, or WriteNoSnpDef request to the Subordinate.
- The Subordinate has two alternatives to return the completion response and the data request response to the Home.
  1. **Separate response**

     The Subordinate does both the following:

     – Returns a Data request response, DBIDResp, to the Home.

     – Returns a completion response, Comp, to the Home. It is permitted, but not required, to wait for write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, before returning Comp.
  2. **Combined response**

     The Subordinate returns a combined data request and completion response, CompDBIDResp, to the Home.
- The Home sends write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, to the Subordinate. The Home must only send this after receiving DBIDResp or CompDBIDResp.
- Optionally, when the request requires a TagMatch response, the Subordinate returns a Tag match response, TagMatch, to the Home. It is permitted, but not required, to wait for write data before returning TagMatch.

B2.3.9.3 Home to Subordinate Write Zero transactions

Figure B2.17 shows the possible transaction flows for a Home to Subordinate Write Zero transaction.

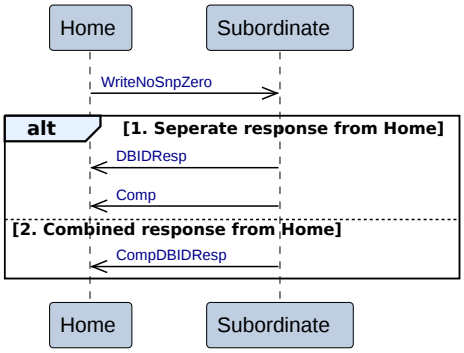

Figure B2.17: Home to Subordinate Write Zero transactions

The sequence for Write Zero is:

- The transaction starts with the Home issuing a Write Zero request to the Subordinate. The Write Zero transaction is:
  - WriteNoSnpZero
- The Subordinate has two alternatives to return the completion response and data request response to the Home.
  1. **Separate response from Home**

     The Subordinate does both the following:
     - Returns a data request response, DBIDResp, to the Home.
     - Returns a completion response, Comp, to the Home.
  2. **Combined response from Home**

     The Subordinate returns a combined data request and completion response, CompDBIDResp, to the Home.

B2.3.9.4 Home to Subordinate Combined Write and CMO transactions

Figure B2.18 shows the possible transaction flows for a Home to Subordinate Combined Write and CMO transaction.

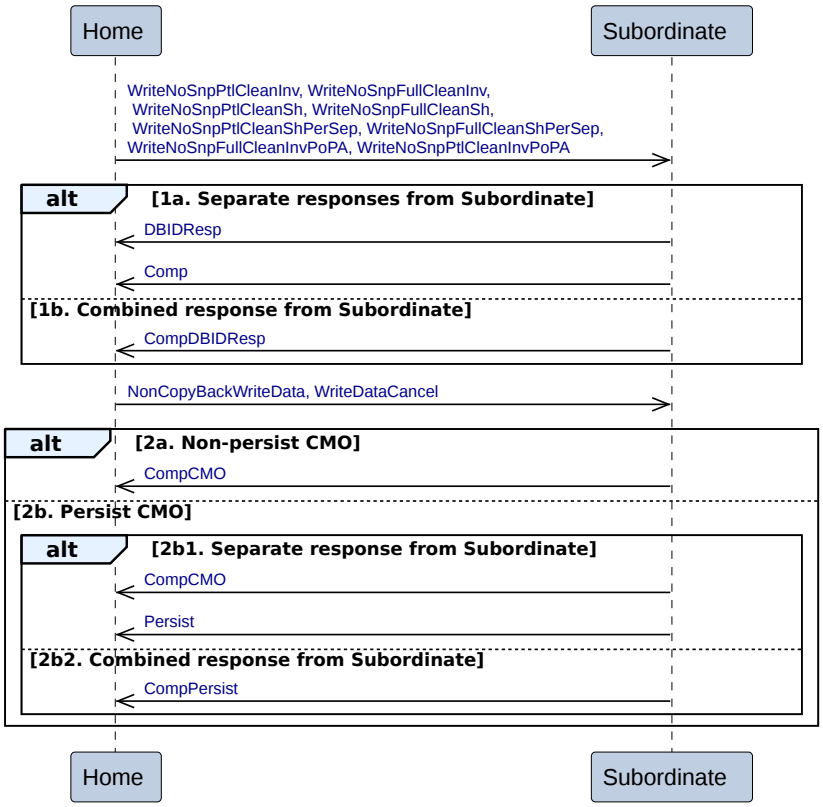

Figure B2.18: Home to Subordinate Combined Write and CMO transactions

The sequence for the Home to Subordinate Combined Write with CMO transaction is:

- The transaction starts with the Home issuing a Combined Write and CMO request to the Subordinate. The Home Combined Write and CMO transactions are:
  - WriteNoSnpPtlCleanInv
  - WriteNoSnpFullCleanInv
  - WriteNoSnpPtlCleanSh
  - WriteNoSnpFullCleanSh
  - WriteNoSnpPtlCleanShPerSep
  - WriteNoSnpFullCleanShPerSep
  - WriteNoSnpFullCleanInvPoPA
  - WriteNoSnpPtlCleanInvPoPA
- The Subordinate has two alternatives to send the completion response and the data request response to the Home.

  Alt 1a. Separate responses from Subordinate
  The Subordinate does both the following:
  - Returns a data request, DBIDResp, to the Home.
  - Returns a completion response, Comp, to the Home.
    It is permitted, but not required, to wait for write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, before returning Comp.

  Alt 1b. Combined response from Subordinate
  The Subordinate returns a combined data request and completion response, CompDBIDResp, to the Home.

- The Home sends write data, NonCopyBackWriteData, or a cancellation, WriteDataCancel, to the Subordinate. The Home must only send this after receiving DBIDResp or CompDBIDResp.
- There are two alternatives for the Subordinate to return the CMO response depending on whether or not a persist response, Persist, is required. It is permitted, but not required, for the Subordinate to wait for write data before returning CompCMO, Persist, or CompPersist.

  Alt 2a. Non-persist CMO
  When a persist response is not required, the Subordinate returns a CMO completion response, CompCMO, to the Home.

  Alt 2b. Persist CMO
  When a persist response is required, the Subordinate has two alternatives to send the CMO completion response and persist response.

  Alt 2b1. Separate response from Subordinate
  The Subordinate does both the following:

  – Returns a CMO completion response, CompCMO, to the Home.

  – Returns a persist response, Persist, to the Home.

  Alt 2b2. Combined response from Subordinate
  The Subordinate returns a combined CMO completion response and persist response, CompPersist, to the Home.

B2.3.9.5 Home to Subordinate Dataless transactions

Figure B2.19 shows the transaction flows for a Home to Subordinate Dataless transaction.

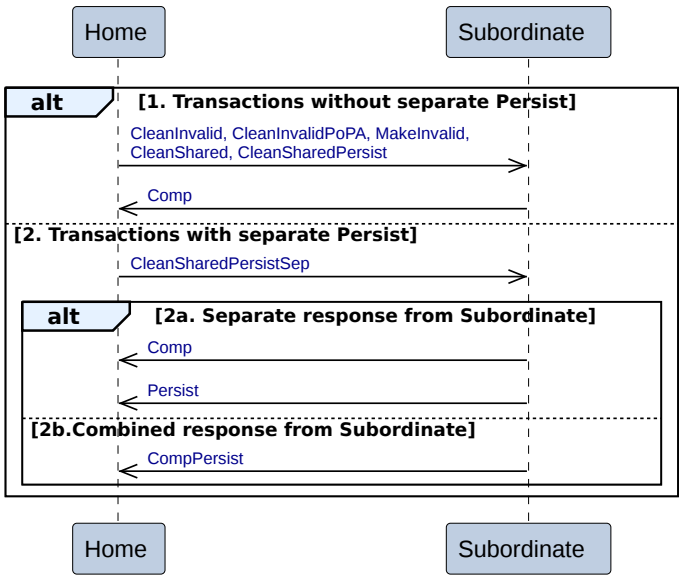

Figure B2.19: Home to Subordinate Dataless transactions

There are two possible sequences for Home to Subordinate Dataless transactions.

1. **Transactions without separate Persist**

   The Home to Subordinate Dataless transactions without separate Persist are:

   - CleanInvalid
   - CleanInvalidPoPA

- MakeInvalid
- CleanShared
- CleanSharedPersist

The Home sends the request to the Subordinate.

The Subordinate returns a completion response, Comp, to the Requester.

2. **Transactions with separate Persist**

   The Home to Subordinate Dataless transaction with separate Persist is:

   - CleanSharedPersistSep

   The Home sends the request to the Subordinate.

   The Subordinate has two alternatives to complete the transaction.

   Alt 2a. Separate response from Subordinate
   The Subordinate does both the following:

   – Returns a completion response, Comp, to the Home.

   – Returns a persist response, Persist, to the Home.
   Use of separate completion response, Comp, and persist response, Persist, allows a Completer to send an early Comp without waiting for Persist. Typically, Persist takes much longer.

   Alt 2b. Combined response from Subordinate
   The Subordinate returns a combined completion and persist response, CompPersist, to the Home.

B2.3.9.6 Home to Subordinate Atomic transactions

Figure B2.20 shows the possible transaction flows for Home to Subordinate Atomic transactions.

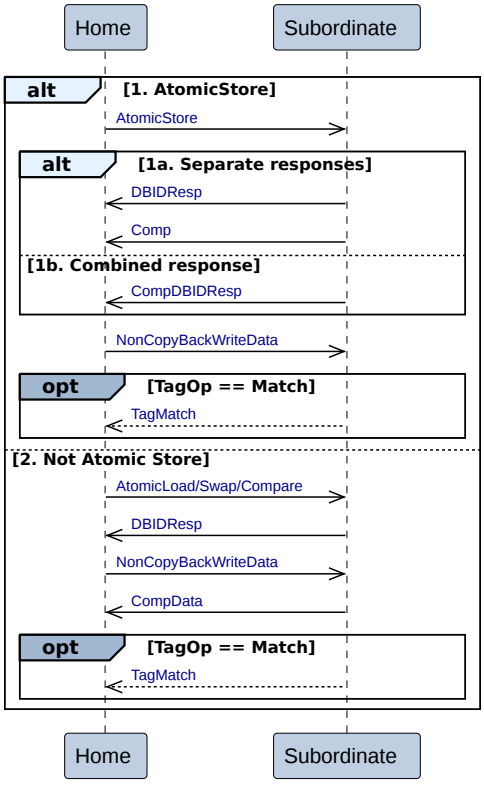

Figure B2.20: Home to Subordinate Atomic transactions

There are two alternatives for the Home Atomic transactions.

When the Subordinate supports the execution of atomic operations, the Home is permitted, but not required, to forward Atomic transactions to the Subordinate.

1. **AtomicStore**

- The Home sends an AtomicStore request to the Subordinate.
- The Subordinate has two alternatives to send the completion response and data request response to the Home.

  Alt 1a. Separate responses
  The Subordinate does both the following:

  – Returns a data request, DBIDResp, to the Home.

  – Returns a completion response, Comp, to the Home.
  It is permitted, but not required, to wait for write data before returning Comp.

  Alt 1b. Combined response
  The Subordinate returns a combined data request and completion response, CompDBIDResp, to the Home.

- The Home sends write data, NonCopyBackWriteData, to the Subordinate. The Home must only send this after receiving DBIDResp or CompDBIDResp. The Home must not wait for Comp before sending write data.
- Optionally, when the request requires a tag match response, the Subordinate returns a TagMatch response to the Home. It is permitted, but not required, to wait for write data before returning TagMatch.

2. **Not AtomicStore**

- The Home sends an AtomicLoad, AtomicSwap, or AtomicCompare request to the Subordinate.
- The Subordinate sends a data request response, DBIDResp, to the Home.
- The Home sends write data, NonCopyBackWriteData, to the Subordinate. The Home must only send this after receiving DBIDResp. The Home must not wait to receive CompData before write data is sent.
- The Subordinate returns a combined data and completion response, CompData, to the Home. It is permitted, but not required, to wait for write data before returning CompData.
- Optionally, when the request requires a TagMatch response, the Subordinate returns a tag match response, TagMatch, to the Home. It is permitted, but not required, to wait for write data before returning TagMatch.

B2.3.9.7 Home to Snoopee transactions

Figure B2.21 shows the possible transaction flows for a Home to Snoopee transaction.

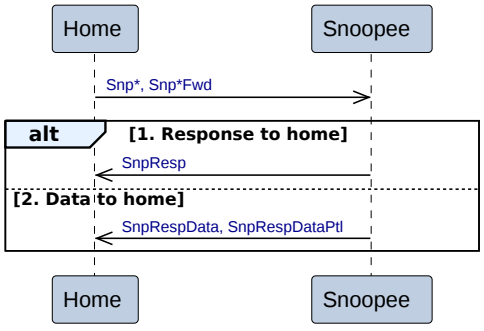

Figure B2.21: Home to Snoopee transactions

The following transactions must use this transaction flow.

- SnpOnce
- SnpClean
- SnpNotSharedDirty
- SnpShared
- SnpUnique
- SnpPreferUnique
- SnpCleanShared
- SnpCleanInvalid
- SnpMakeInvalid
- SnpQuery

The following transactions are also permitted, but not required, to use this transaction flow.

- SnpOnceFwd
- SnpCleanFwd
- SnpNotSharedDirtyFwd
- SnpSharedFwd
- SnpUniqueFwd
- SnpPreferUniqueFwd

The sequence the Home to Snoopee transaction is:

- The transaction starts with the Home issuing a Snoop request to the Snoopee.
- The Snoopee has two alternatives to complete the transaction:
  1. The Snoopee provides a snoop response, SnpResp, to the Home. This is the only permitted alternative for a SnpMakeInvalid transaction.
  2. The Snoopee provides a snoop response with data, SnpRespData or SnpRespDataPtl.

B2.3.9.8 Home to Snoopee DVM transactions

Figure B2.22 shows the transaction flow for a Home to Snoopee DVM transaction, SnpDVMOp.

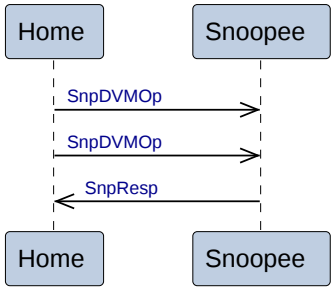

Figure B2.22: Home to Snoopee DVM transactions

The sequence for the Home to Snoopee DVM transaction is:

- The Home issues two Snoop DVM requests, SnpDVMOp, to the Snoopee.
- The Snoopee provides a single Snoop response, SnpResp. The Snoopee must only provide the Snoop response after receiving both Snoop DVM requests.

B2.4 Transaction identifier fields

Each transaction consists of a number of different packets that are transferred across the interconnect. A set of identifier fields, within a packet, are used to provide additional information about a packet.

The field that routes packets across the interconnect:

- B2.4.1 *Target Identifier, TgtID, and Source Identifier, SrcID*

The fields that relate all the packets associated with a single transaction are:

- B2.4.2 *Transaction Identifier, TxnID*
- B2.4.3 *Data Buffer Identifier, DBID*
- B2.4.4 *Return Transaction Identifier, ReturnTxnID*
- B2.4.5 *Forward Transaction Identifier, FwdTxnID*

The field that identifies the individual data packets within a transaction:

- B2.4.6 *Data Identifier, DataID, and Critical Chunk Identifier, CCID*

The fields that identify individual processing agents within a single Requester:

- B2.4.7 *Logical Processor Identifier, LPID*
- B2.4.8 *Stash Logical Processor Identifier, StashLPID*

The field that identifies the target node for a Stash transaction:

- B2.4.9 *Stash Node Identifier, StashNID*

The field that identifies the recipient node for the Data response, Persist response, or TagMatch response:

- B2.4.10 *Return Node Identifier, ReturnNID*

The field that identifies the recipient node for the Data response:

- B2.4.12 *Forward Node Identifier, FwdNID*

The field that is used to identify the recipient node for the CompAck response:

- B2.4.11 *Home Node Identifier, HomeNID*

The field that is used to identify different sets of transactions:

- B2.4.13 *Persistence Group Identifier, PGroupID*
- B2.4.14 *Stash Group Identifier, StashGroupID*
- B2.4.15 *Tag Group Identifier, TagGroupID*

B2.4.1 Target Identifier, TgtID, and Source Identifier, SrcID

A transaction request includes a TgtID that identifies the target node, and a SrcID that identifies the source node. These identifiers are used to route packets across the interconnect.

B2.4.2 Transaction Identifier, TxnID

A transaction request includes a TxnID that is used to identify the transaction from a given Requester. It is required that the TxnID, except for PrefetchTgt, must be unique for a given Requester. The Requester is identified by the SrcID. This ensures that any returning read data or response information can be associated with the correct transaction.

A 12-bit field is defined for the TxnID with the number of outstanding transactions limited to 1024. A Requester is permitted to reuse a TxnID value after receiving either:

- All responses associated with a previous transaction that have used the same value.
- A RetryAck response for a previous transaction that used the same value.

B2.5 *Transaction identifier field flows* gives more detailed rules for the different transaction types. The TxnID field is not applicable in a PrefetchTgt request and can take any value.

A value used in the TxnID field of a Request from Home to Subordinate can be reused by Home once all responses that are required to deallocate the request are received or a RetryAck response is received.

A transaction that is retried is not required to use the same TxnID. See B2.9 *Request Retry*.

B2.4.3 Data Buffer Identifier, DBID

The DBID field permits the Completer of a transaction to provide its own identifier for a transaction. The Completer sends a response that includes a DBID. The DBID value is used as the TxnID field value in the:

- WriteData response of Immediate Write, CopyBack Write, Combined Write, Atomic, and DVMOp transactions.
- CompData response of Stash transactions for Data Pull purposes.
- CompAck response of Read, Dataless, WriteNoSnp, WriteUnique and Immediate Combined Write transactions that include a CompAck response.

The DBID value used by a Completer in responses of a given transaction must be unique for a given Requester in the following cases:

- DBIDResp or DBIDRespOrd or CompDBIDResp for all Write transactions, except in WriteNoSnpZero and WriteUniqueZero.
- DBIDResp or DBIDRespOrd or CompDBIDResp for all Combined Write transactions.
- DBIDResp or DBIDRespOrd or CompDBIDResp for Atomic transactions.
- DBIDResp or DBIDRespOrd or CompDBIDResp for DVMOp transactions.
- CompData or RespSepData for Read transactions that include CompAck, except in the case when ReadOnce* and ReadNoSnp do not use the resultant CompAck for deallocation of the request at Home.
- Comp for Dataless transactions that include CompAck.

The DBID value is applicable in the DataSepResp response to Read requests that include CompAck, and it must be the same as the DBID value in the associated RespSepData response.

A Comp response message sent separate from a DBIDResp or DBIDRespOrd message for a Write or Combined Write transaction must include the same DBID field value in the Comp and DBIDResp or DBIDRespOrd message when the two messages originate from the same source.

A Comp response message sent separate from a DBIDResp or DBIDRespOrd message for a Atomic transaction is permitted, but is not required, to include the same DBID field value in the Comp and DBIDResp or DBIDRespOrd message.

A Completer is permitted, but not required, to use the same DBID value for two transactions with different Requesters. A Completer is permitted to reuse a DBID value after it has received all packets required to deallocate a previous transaction that has used the same value. B2.5 *Transaction identifier field flows* gives more detailed rules for the different transaction types.

The DBID value used by a Snoop Completer in response to a Stash snoop that includes a Data Pull must be unique with respect to:

- The DBID values in other Snoop responses to Stash snoops that use Data Pull.
- The TxnID of any outstanding request from that Snoop Completer.

The Completer is not required to utilize the DBID field, and is permitted to set the DBID to any value in:

- WriteNoSnpZero and WriteUniqueZero transactions.
- Read transactions without CompAck.
- Dataless transactions without CompAck.
- Snoop response to a Stash snoop that does not include a Data Pull.
- Snoop response to a Non-stash snoop.

Note

The advantage of using the DBID assigned by the Completer, instead of the TxnID assigned by the Requester, is that the Completer can use the DBID to index into its request structure instead of performing a lookup using TxnID and SrcID to determine which transaction write data or completion acknowledge is associated with which request.

If a Completer is using the same DBID value for different Requesters, which it must do if its operation requires more than 1024 DBID responses to be active at the same time, SrcID with DBID must be used to determine which request should be associated with a write data or response message.

The DBIDResp response is also used to provide certain ordering guarantees relating to the transaction. See B2.6.5 *Transaction ordering*.

B2.4.4 Return Transaction Identifier, ReturnTxnID

A transaction request from Home to Subordinate also includes a ReturnTxnID field to convey the value of TxnID in the data response and the DBIDResp response from the Subordinate.

Its value, when applicable, must be either:

- The TxnID generated by Home, when the ReturnNID is the node ID of the Home.
- The TxnID of the original Requester, when the ReturnNID is the node ID of the original Requester.

ReturnTxnID is only applicable in a ReadNoSnp, ReadNoSnpSep, WriteNoSnp, Combined Write, and Atomic requests from Home to Subordinate. The field is inapplicable and must be set to 0 in all other requests from Home to Subordinate.

ReturnTxnID is inapplicable and must be set to 0 in all requests from Requester to Home, and Requester to Subordinate.

The following are the expected and permitted values for ReturnTxnID in requests from the Home Node to the Subordinate Node.

In ReadNoSnp and ReadNoSnpSep:

- Expected value is the original Requester TxnID but permitted to be the Home TxnID.
- Used as the TxnID in the CompData and DataSepResp responses.

In Atomic with TagOp *Invalid* or *Match*:

- For AtomicStore, the ReturnTxnID can take any value, and the value is not used in any response.
- For Non-store Atomics, the ReturnTxnID must be the Home TxnID. The value is used as the TxnID in the CompData response.

In WriteNoSnp with any TagOp value:

- When DoDWT = 0, ReturnTxnID can take any value, and the value is not used in any response.
- When DoDWT = 1, the ReturnTxnID value is expected to be the original Requester TxnID but is permitted to be the Home TxnID. Used as the TxnID in the DBIDResp response.

In WriteNoSnpDef:

- When DoDWT = 0, ReturnTxnID can take any value, and the value is not used in any response.
- When DoDWT = 1, the ReturnTxnID value is expected to be the original Requester TxnID but is permitted to be the Home TxnID. Used as the TxnID in the DBIDResp response.

In Combined Write:

- When DoDWT = 0, ReturnTxnID can take any value, and the value is not used in any response.
- When DoDWT = 1, the ReturnTxnID value is expected to be the original Requester TxnID but is permitted to be the Home TxnID. Used as the TxnID in the DBIDResp response.

B2.4.5 Forward Transaction Identifier, FwdTxnID

A Snoop request from Home to RN-F also includes a FwdTxnID field to convey the value of TxnID in the Data response from the Snoopee. Its value must be the TxnID of the original Request.

The FwdTxnID field is only applicable in:

- SnpSharedFwd
- SnpCleanFwd
- SnpOnceFwd
- SnpNotSharedDirtyFwd
- SnpUniqueFwd
- SnpPreferUniqueFwd

The FwdTxnID field is inapplicable and must be set to 0 in all other snoops.

B2.4.6 Data Identifier, DataID, and Critical Chunk Identifier, CCID

These fields identify the individual data packets within a transaction.

See B2.8.4 *Data packetization* and B2.8.6 *Critical Chunk Identifier*.

B2.4.7 Logical Processor Identifier, LPID

This field is used when a single Requester contains more than one logically separate processing agent. The SrcID is used with LPID to uniquely identify the LP that generated the request.

The LPID must be set to the correct value for the following transactions:

- For any Non-snoopable Non-cacheable or Device access:
  - ReadNoSnp
  - WriteNoSnp
  - WriteNoSnpDef
- For Exclusive accesses, that can be one of the following transaction types:
  - ReadClean
  - ReadShared
  - ReadNotSharedDirty
  - ReadPreferUnique
  - MakeReadUnique
  - CleanUnique
  - ReadNoSnp
  - WriteNoSnp

See Chapter B6 *Exclusive accesses* for further details.

For other transactions, the LPID value is permitted, but not required, to indicate the original LP that caused a transaction to be issued.

In requests, when applicable, the same bits in the packet are used for TagGroupID, PGroupID, and StashGroupID.

B2.4.8 Stash Logical Processor Identifier, StashLPID

The StashLPID field can be used to specify a particular LP within the Request Node specified by StashNID, when the corresponding StashLPIDValid bit is asserted. See B13.10.11 *Stash Logical Processor Identifier, StashLPID*.

See B7.5.1 *Supporting REQ packet fields* for the permitted combinations of StashLPIDValid and StashNIDValid.

B2.4.9 Stash Node Identifier, StashNID

The StashNID field provides the Request Node that is the target of the Stash transaction when the corresponding StashNIDValid bit is asserted. See B13.10.9 *Stash Node Identifier, StashNID*.

B2.4.10 Return Node Identifier, ReturnNID

A transaction request from Home to Subordinate includes a ReturnNID that is used to determine the TgtID for the following responses from the Subordinate Node:

- Data response
- DBIDResp response
- Persist response
- TagMatch response

Its value must be either the node ID of Home or the node ID of the original Requester.

ReturnNID is only applicable in a ReadNoSnp, ReadNoSnpSep, CleanSharedPersistSep, WriteNoSnp, Combined Write, and Atomic requests from Home to Subordinate. The field is inapplicable and must be set to 0 in all other requests from Home to Subordinate.

ReturnNID is inapplicable and must be set to 0 in all requests from Requester to Home and Requester to Subordinate.

The following are the expected and permitted values for the ReturnNID in requests from the Home Node to the Subordinate Node.

In ReadNoSnp, ReadNoSnpSep, and CleanSharedPersistSep:

- Expected value is the original Requester Node ID but is permitted to be the Home Node ID.
- Used as the TgtID in CompData, DataSepResp and Persist responses.

In Atomic with TagOp *Invalid*:

- For AtomicStore, ReturnNID can take any value and the value is not used in any responses.
- For Non-store Atomics, the ReturnNID must be the Home Node ID. The value is used as the TgtID in CompData.

In Atomic with TagOp *Match*:

- ReturnNID must be the Home Node ID.
- The value is used as the TgtID in CompData and TagMatch responses.

In WriteNoSnp with TagOp not *Match*:

- When DoDWT = 0, the ReturnNID can take any value, and the value is not used in any responses.
- When DoDWT = 1, the ReturnNID value is expected to be the original Requester Node ID but is permitted to be the Home Node ID. Used as the TgtID in the DBIDResp response.

In WriteNoSnp with TagOp *Match*:

- Irrespective of the value of DoDWT, the ReturnNID value is expected to be the original Requester Node ID but is permitted to be the Home Node ID.
- When DoDWT = 0, the ReturnNID value is used as the TgtID in the TagMatch response only.
- When DoDWT = 1, the ReturnNID value is used as the TgtID in the DBIDResp and TagMatch responses.

In WriteNoSnpDef:

- When DoDWT = 0, ReturnNID can take any value, and the value is not used in any response.
- When DoDWT = 1, the ReturnNID value is expected to be the original Requester Node ID but is permitted to be the Home Node ID. Used as the TgtID in the DBIDResp response.

In Non-PCMO Combined Write:

- When DoDWT = 0, ReturnNID can take any value, and the value is not used in any responses.
- When DoDWT = 1, the ReturnNID value is expected to be the original Requester Node ID but is permitted to be the Home Node ID. Used as the TgtID in the DBIDResp response.

In WriteNoSnpFullClnShPer and WriteNoSnpPtlClnShPer:

- Irrespective of the value of DoDWT, the ReturnNID value is expected to be the original Requester Node ID but is permitted to be the Home Node ID.
- When DoDWT = 0, the ReturnNID value is used as the TgtID in the Persist response only.
- When DoDWT = 1, the ReturnNID value is used as the TgtID in the DBIDResp and Persist responses.

B2.4.11 Home Node Identifier, HomeNID

CompData includes the HomeNID field that is used by the Requester to identify the target of CompAck that the Requester could need to send in response to CompData. HomeNID is applicable in CompData and DataSepResp and is inapplicable and must be set to 0 for all other Data messages.

Note

There is no functional requirement for the HomeNID and DBID fields in the DataSepResp response because the values that are provided in the RespSepData response are identical and can always be used. However, it is required that these values are included to help with debugging and protocol checking.

B2.4.12 Forward Node Identifier, FwdNID

A Snoop request from Home to RN-F includes a FwdNID that is used to determine the TgtID for the Data response from the Snoopee. Its value must be the node ID of the original Requester.

The FwdNID field is only applicable in:

- SnpSharedFwd
- SnpCleanFwd
- SnpOnceFwd
- SnpNotSharedDirtyFwd
- SnpUniqueFwd

- SnpPreferUniqueFwd

The FwdNID field is inapplicable and must be set to 0 in all other snoops, except range-based *Translation Lookaside Buffer Invalidate* (TLBI) DVM operations. For range-based TLBI operations, the bits in the field are used for DVM payload.

B2.4.13 Persistence Group Identifier, PGroupID

A CleanSharedPersistSep and *Combined Write with Persistent CMO* (PCMO) request includes a PGroupID to identify the Persistence Group that the request belongs to. If a Requester has persistent CMO requests from different functional agents that it would like to identify for performant persistent CMO handling, it can assign a different PGroupID value to each group of Persist requests. Use of this 8-bit field is applicable in CleanSharedPersistSep and Combined Write with PCMO transactions. It is also applicable in Persist and CompPersist responses. It is inapplicable and must be set to zero in all other requests and responses. See B13.10.8 *Persistence Group Identifier, PGroupID*:

- PGroupID must be sent in the CleanSharedPersistSep request and a Combined Write request that includes a PCMO.
- The PGroupID value returned in the Persist response can be used by a Requester to separately track completions of Persist responses from each group.
- It is expected that a Requester that does not support multiple persistence groups sets the PGroupID value to 0.
- Typically, a Requester that is making use of PGroupID for passing a barrier does not reuse a PGroupID value until all the earlier sent CleanSharedPersistSep requests from that group have received Persist responses.
- The Completer is required to reflect back PGroupID in the Persist and CompPersist responses, and the responses of Combined Write requests that include a PCMO.
- The PGroupID field in the Comp and CompCMO response from both the Home and Subordinate is inapplicable and must be set to 0.

B2.4.14 Stash Group Identifier, StashGroupID

To identify the Stash Group that the request belongs to, a StashOnceSep request includes a StashGroupID. The same StashGroupID value from the request is later used by a StashDone response in the transaction flow. If a Requester has StashOnceSep requests from different functional agents that can be identified for performant stash handling, a different StashGroupID value to each group of Stash requests can be assigned.

Use of this 8-bit field is applicable in the StashOnceSep request and StashDone response. StashGroupID is inapplicable and must be set to 0 in all other requests and responses.

- StashGroupID must be sent in the StashOnceSep request.
- The StashGroupID value returned in the StashDone response can be used by a Requester to separately track completions of Stash transactions from each group.
- It is expected that a Requester that does not support multiple stash groups sets the StashGroupID value to 0.
- The Completer is required to reflect back StashGroupID in the StashDone response.

See B13.10.13 *Stash Group Identifier, StashGroupID*.

B2.4.15 Tag Group Identifier, TagGroupID

To identify the Tag group that the request belongs to, a Write request where TagOp is set to *Match* includes a TagGroupID. The same TagGroupID value from the request is later used by a TagMatch response in the transaction flow to notify the Requester if the TagMatch operation passed or failed.

Use of this 8-bit field is applicable in Write requests where TagOp is set to *Match*, and in a TagMatch response. TagGroupID is inapplicable and must be set to 0 in all other requests and responses.

- TagGroupID must be sent in a Write request where TagOp is set to *Match*.
- The precise contents of TagGroupID are IMPLEMENTATION DEFINED. Typically, TagGroupID is expected to contain an Exception Level, TTBR value, and PE identifier.

See B13.10.43 *Tag Group Identifier, TagGroupID*.

B2.5 Transaction identifier field flows

This section shows the transaction identifier field flows for the different transaction types:

- B2.5.1 *Read transactions*
- B2.5.2 *Dataless transactions*
- B2.5.3 *Write transactions*
- B2.5.4 *DVMOp transaction*
- B2.5.5 *Transaction requests with Retry*
- B2.5.6 *Protocol Credit Return transaction*

In the associated figures:

- The fields included in each packet are:
  - For a Request packet: TgtID, SrcID, TxnID, StashNID, StashLPID, ReturnNID, ReturnTxnID, PGroupID, StashGroupID, and TagGroupID.
  - For a Response packet: TgtID, SrcID, TxnID, DBID, PGroupID, StashGroupID, and TagGroupID.
  - For a Data packet: TgtID, SrcID, TxnID, HomeNID, and DBID.
  - For a Snoop packet: SrcID, TxnID, FwdNID, FwdTxnID, and StashLPID.
- All fields with the same color are the same value.
- The curved loop-back arrows show how the Requester and Completer use fields from earlier packets to generate fields for subsequent packets.
- A box containing an asterix [*] indicates when a field is first generated, that is, indicating the agent that determines the original value of the field.
- A field enclosed in parentheses indicates that the value is effectively a fixed value. Typically this is the case for the SrcID field when a packet is sent, and the TgtID field when a packet arrives at its destination.
- A field that is crossed-out indicates that the field is not valid.
- It is permitted for the TgtID of the original transaction to be remapped by the interconnect to a new value. This is shown by a box containing the letter R. This is explained in more detail in Chapter B3 *Network Layer*.

Note

An identifier field, in every packet sent, belongs to one of the following categories:

- New value. An asterisk indicates that a new value is generated.
- Generated from an earlier packet. A loop back arrow indicates the source.
- Fixed value. The value is enclosed in brackets.
- Not valid. The field is crossed-out.

In the following examples, any transaction identifiers that are not relevant for the example are sometimes omitted for clarity.

B2.5.1 Read transactions

This section shows the identifier field flows in Read transactions with and without Direct Data Transfer:

- B2.5.1.1 *ID value transfer with DMT*
- B2.5.1.2 *ID value transfer with DMT and separate Comp and Data*
- B2.5.1.3 *ID value transfer with DCT*
- B2.5.1.4 *ID value transfer without Direct Data Transfer*

B2.5.1.1 ID value transfer with DMT

Figure B2.23 shows how the Target and Transaction ID values in the DMT transaction messages are derived. For example, the value of SrcID in the ReadNoSnp request from the interconnect is assigned by the interconnect. Whereas the ReturnNID, used as TgtID in the Data response, is set to the value of SrcID of the received Read request.

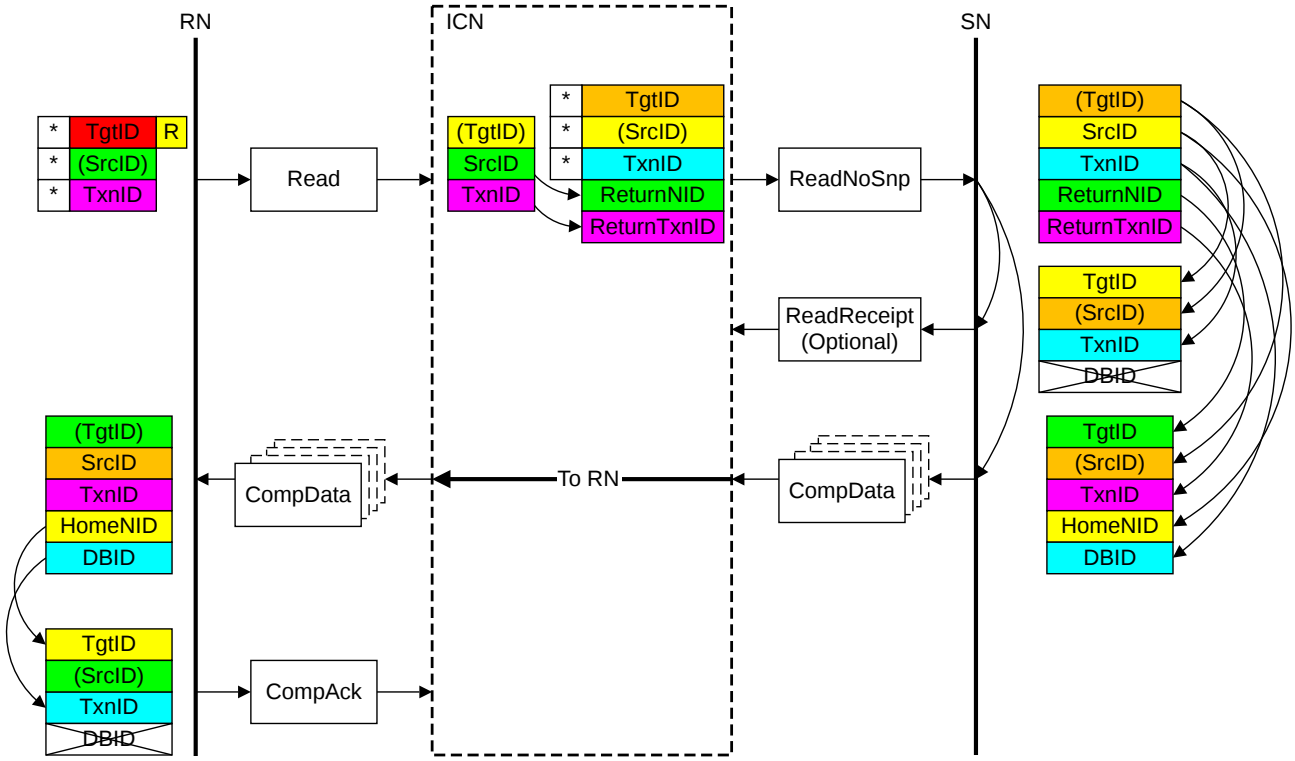

Figure B2.23: ID value transfer in a DMT transaction

The required steps in the flow that Figure B2.23 shows are:

1. The Requester starts the transaction by sending a Request packet.
   The identifier fields of the request are generated as follows:
   - The TgtID is determined by the destination of the Request.

     Note

     The TgtID field can be remapped to a different value by the interconnect.

   - The SrcID is a fixed value for the Requester.
   - The Requester generates a TxnID field that is unique for that Requester.
2. The recipient Home Node in the interconnect generates a Request to the Subordinate Node.
   The identifier fields of the request are generated as follows:
   - The TgtID is set to the value required for the Subordinate.
   - The SrcID is a fixed value for the Home.

- The TxnID is a unique value generated by the Home.
- The ReturnNID is set to the same value as the SrcID of the original request.
- The ReturnTxnID is set to the same value as the TxnID of the original request.

3. If the request to the Subordinate requires a ReadReceipt, the Subordinate provides the read receipt. The identifier fields of the ReadReceipt response are generated as follows:
   - The TgtID is set to the same value as the SrcID of the request.
   - The SrcID is a fixed value for the Subordinate. This also matches the TgtID received.
   - The TxnID is set to the same value as the TxnID of the request.
   - The DBID field is not valid.
4. The Subordinate provides the read data.
   The identifier fields of the Read data response are generated as follows:
   - The TgtID is set to the same value as the ReturnNID of the request.
   - The SrcID is a fixed value for the Subordinate. This also matches the TgtID received.
   - The TxnID is set to the same value as the ReturnTxnID of the request.
   - The HomeNID is set to the same value as the SrcID of the request.
   - The DBID is set to the same value as the TxnID of the request.
5. The Requester receives the read data and sends a completion acknowledge, CompAck, response. The identifier fields of the CompAck are generated as follows:
   - The TgtID is set to the same value as the HomeNID of the read data.
   - The SrcID is a fixed value for the Requester. This also matches the TgtID that was received.
   - The TxnID is set to the same value as the DBID of the read data.
   - The DBID field is not valid.

The CompAck response from Requester to Home is not required for all requests.

If the original request requires a ReadReceipt, the following additional step is included:

- The Home receives the Request packet and provides the read receipt. The identifier fields of the ReadReceipt response are generated as follows:
  - The TgtID is set to the same value as the SrcID of the request.
  - The SrcID is a fixed value for the Completer. This also matches the TgtID received.
  - The TxnID is set to the same value as the TxnID of the request.
  - The DBID field is not valid.

B2.4 *Transaction identifier fields* details when the TxnID value and DBID value can be reused.

B2.5.1.2 ID value transfer with DMT and separate Comp and Data

Figure B2.24 shows how the identifier field values are derived in DMT transaction messages that use separate Comp and Data.

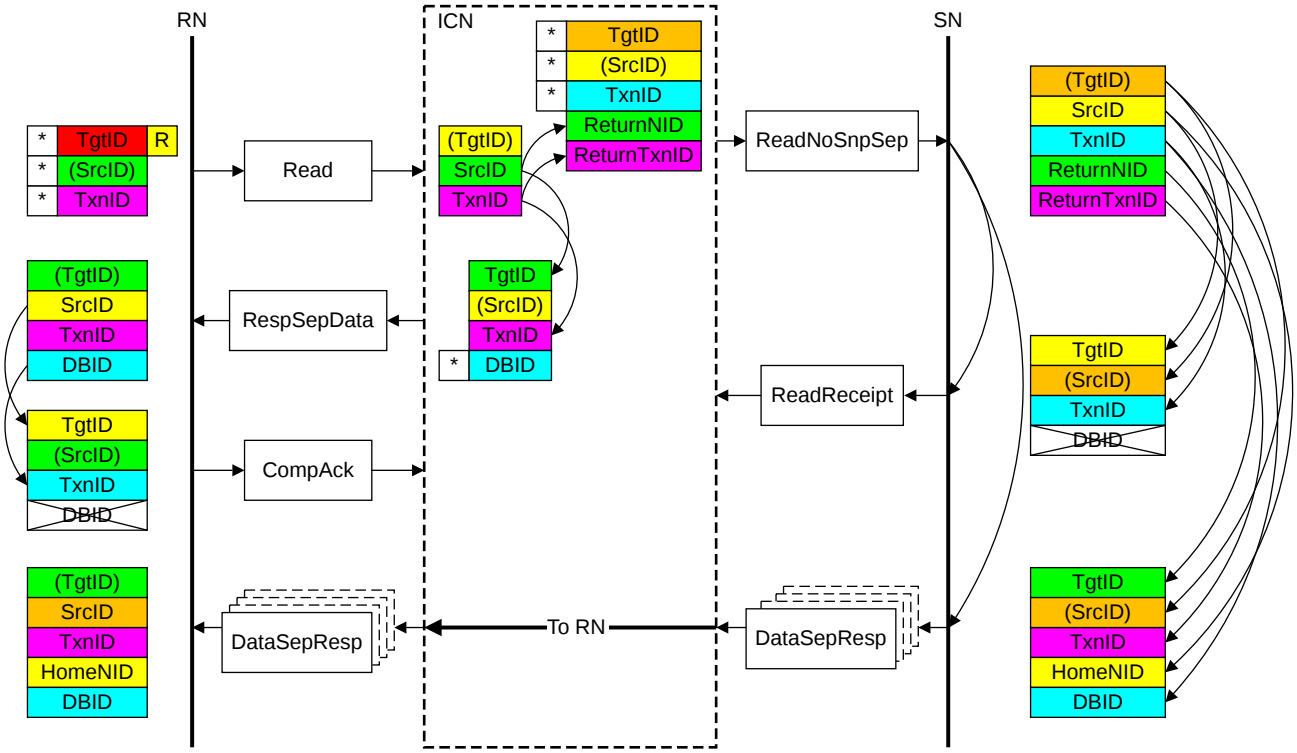

Figure B2.24: ID value transfer in a DMT transaction with separate Comp and Data

The required steps in the flow that Figure B2.24 shows are:

1. The Requester starts the transaction by sending a Request packet.
   The identifier fields of the request are generated as follows:
   - The TgtID is determined by the destination of the Request.

     Note

     The TgtID field can be remapped to a different value by the interconnect.

   - The SrcID is a fixed value for the Requester.
   - The Requester generates a TxnID field that is unique for that Requester.
2. The recipient Home Node in the interconnect generates a request to the Subordinate Node.
   The identifier fields of the request are generated as follows:
   - The TgtID is set to the value required for the Subordinate.
   - The SrcID is a fixed value for the Home.
   - The TxnID is a unique value generated by the Home.
   - The ReturnNID is set to the same value as the SrcID of the original request.
   - The ReturnTxnID is set to the same value as the TxnID of the original request.
3. The recipient Home Node in the interconnect provides the separate Read response.
   The identifier fields of the read response are generated as follows:

- The TgtID is set to the same value as the SrcID of the request.
- The SrcID is a fixed value for the Home.
- The TxnID is set to the same value as the TxnID of the original request.
- The DBID value is a unique value generated by the Home and is the same value as the TxnID in the request to the Subordinate.

4. The Requester receives the Read response and sends a completion acknowledge, CompAck, response. The identifier fields of the CompAck are generated as follows:
   - The TgtID is set to the same value as the SrcID of the read response.
   - The SrcID is a fixed value for the Requester.
   - The TxnID is set to the unique DBID value generated by the Home.
   - The DBID value is not valid.
5. The request to the Subordinate requires a ReadReceipt. The Subordinate provides the read receipt. The identifier fields of the ReadReceipt response are generated as follows:
   - The TgtID is set to the same value as the SrcID of the request.
   - The SrcID is a fixed value for the Subordinate. This also matches the TgtID received.
   - The TxnID is set to the same value as the TxnID of the request.
6. The Subordinate provides the separate read data.
   The identifier fields of the read data are generated as follows:
   - The TgtID is set to the same value as the ReturnNID of the request.
   - The SrcID is a fixed value for the Subordinate. This also matches the TgtID received.
   - The TxnID is set to the same value as the ReturnTxnID of the request.
   - The HomeNID is set to the same value as the SrcID of the request.
   - The DBID is set to the same value as the TxnID of the request.

B2.5.1.3 ID value transfer with DCT

Figure B2.25 shows how the identifier field values are derived in DCT transaction messages. In this example, the data is forwarded to a Request Node and a Snoop response is sent to HN-F with or without data.

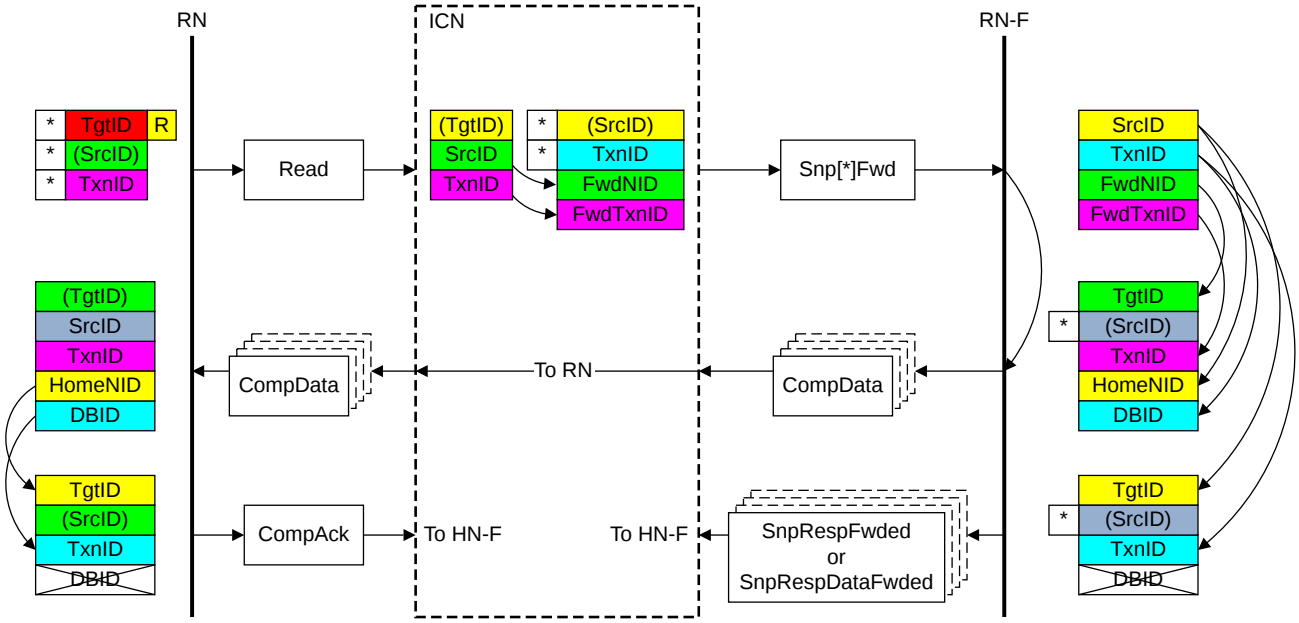

Figure B2.25: ID value transfer in a DCT transaction

The required steps in the flow that Figure B2.25 shows are:

1. The Requester starts the transaction by sending a Request packet.
   The identifier fields of the request are generated as follows:
   - The TgtID is determined by the destination of the Request.

     Note

     The TgtID field can be remapped to a different value by the interconnect.

   - The SrcID is a fixed value for the Requester.
   - The Requester generates a TxnID field that is unique for that Requester.
2. The recipient Home Node in the interconnect generates a Forwarding snoop to the RN-F node.
   The identifier fields of the snoop are generated as follows:
   - The SrcID is a fixed value for the Home.
   - The TxnID is a unique value generated by the Home.
   - The FwdNID is set to the same value as the SrcID of the original request.
   - The FwdTxnID is set to the same value as the TxnID of the original request.
3. The RN-F provides the read data.
   The identifier fields of the Read data response are generated as follows:
   - The TgtID is set to the same value as the FwdNID of the snoop.
   - The SrcID is a fixed value for the RN-F.
   - The TxnID is set to the same value as the FwdTxnID of the snoop.
   - The HomeNID is set to the same value as the SrcID of the snoop.

- The DBID is set to the same value as the TxnID of the snoop.

4. The RN-F also provides a response to Home, either with or without read data.
   The identifier fields of the response are generated as follows:
   - The TgtID is set to the same value as the SrcID of the snoop.
   - The SrcID is a fixed value for the RN-F.
   - The TxnID is set to the same value as the TxnID of the snoop.
   - The DBID field is not valid.
5. The Requester receives the read data and sends a completion acknowledge, CompAck, response.
   The identifier fields of the CompAck are generated as follows:
   - The TgtID is set to the same value as the HomeNID of the read data.
   - The SrcID is a fixed value for the Requester. This also matches the TgtID that was received.
   - The TxnID is set to the same value as the DBID of the read data.
   - The DBID field is not valid.

Note

An optional ReadReceipt from the interconnect to Requester can also be included.

B2.5.1.4 ID value transfer without Direct Data Transfer

This section gives an example of a Read identifier field flow without DMT or DCT and describes the use of the TxnID and DBID fields for Read transactions.

The Requester and Completer in this example are a Request Node and an HN-F respectively.

The identifier field flow includes an optional ReadReceipt response from the Completer, and an optional CompAck response from the Requester.

For Read transactions that include a CompAck response, the DBID is used by the Completer to associate the CompAck with the original transaction.

A Read transaction that does not include a CompAck response does not require a valid DBID field in the data response.

Figure B2.26 shows the ID value transfer.

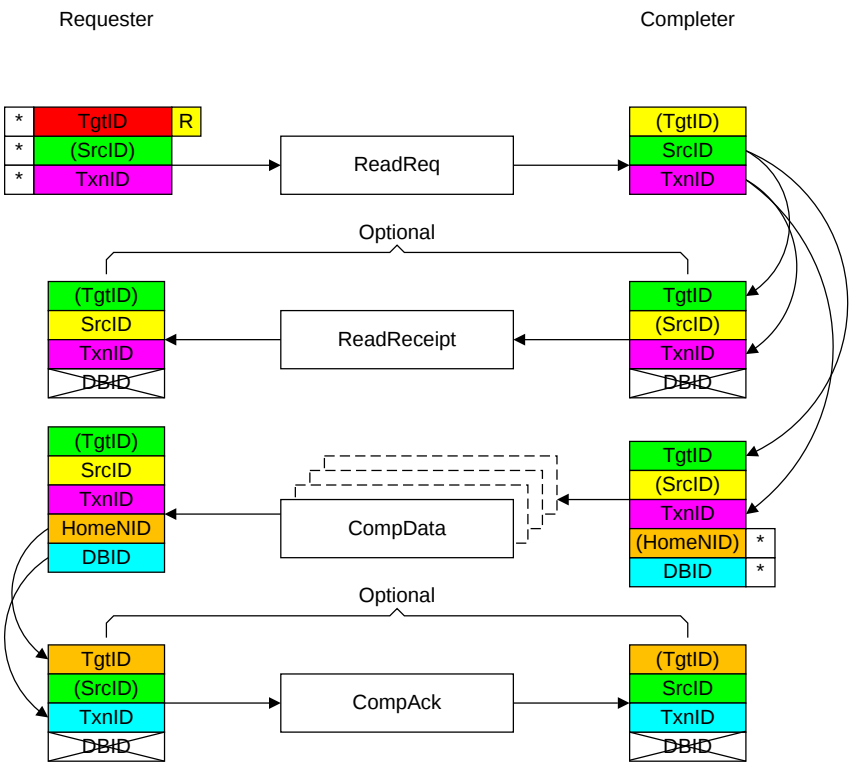

Figure B2.26: ID value transfer in a Read request with ReadReceipt and CompAck

The required steps in the flow that Figure 2-26 shows are:

1. The Requester starts the transaction by sending a Request packet.
   The identifier fields of the request are generated as follows:
   - The TgtID is determined by the destination of the Request.

     Note

     The TgtID field can be re-mapped to a different value by the interconnect.

   - The SrcID is a fixed value for the Requester.
   - The Requester generates a TxnID field that is unique for that Requester.
2. If the transaction includes a ReadReceipt, the Completer receives the Request packet and provides the read receipt.
   The identifier fields of the ReadReceipt response are generated as follows:
   - The TgtID is set to the same value as the SrcID of the request.
   - The SrcID is a fixed value for the Completer. This also matches the TgtID received.
   - The TxnID is set to the same value as the TxnID of the request.
   - The DBID field is not valid.
3. The Completer receives the Request packet and provides the read data.
   The identifier fields of the read data response are generated as follows:
   - The TgtID is set to the same value as the SrcID of the request.

- The SrcID is a fixed value for the Completer. This also matches the TgtID received.
- The TxnID is set to the same value as the TxnID of the request.
- The HomeNID is a fixed value for the Completer. This also matches the TgtID received.
- The Completer generates a unique DBID value if ExpCompAck in the request is asserted.

4. The Requester receives the read data and sends a completion acknowledge, CompAck, response. The identifier fields of the CompAck are generated as follows:
   - The TgtID is set to the same value as the HomeNID of the read data.
   - The SrcID is a fixed value for the Requester. This also matches the TgtID that was received.
   - The TxnID is set to the same value as the DBID of the read data.
   - The DBID field is not valid.

B2.5.2 Dataless transactions

For Dataless transactions, except for CleanSharedPersistSep and StashOnceSep, the use of identifier fields is similar to B2.5.1.4 *ID value transfer without Direct Data Transfer*. The only difference is that the response from the Completer to the Requester is sent as a single packet on the CRSP channel instead of multiple packets on the RDAT channel.

For StashOnceSep transactions, the StashGroupID value is sent in the request from the Request Node to the interconnect and the value is returned in the StashDone and CompStashDone responses. The TxnID value in the StashDone response is inapplicable and must be set to 0.

The description of ID value transfer in a CleanSharedPersistSep transaction follows.

B2.5.2.1 ID value transfer in a CleanSharedPersistSep transaction

Figure B2.27 shows how the identifier field values are derived in CleanSharedPersistSep transaction messages that use separate Comp and Persist responses. In Figure B2.27, PCMOSep represents CleanSharedPersistSep.

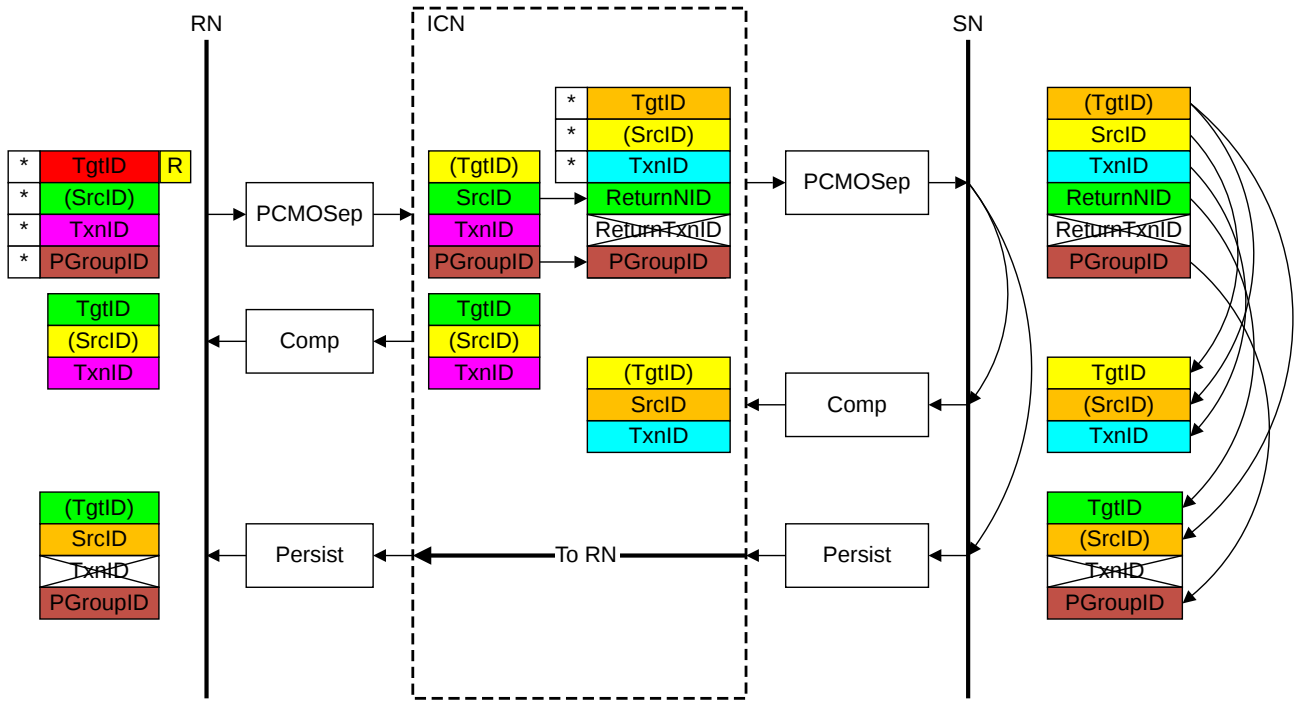

Figure B2.27: ID value transfer in a CleanSharedPersistSep transaction

The required steps in the flow that Figure B2.27 shows are:

1. The Requester starts the transaction by sending a Request packet.
   The identifier fields of the Request are generated as follows:
   - The TgtID is determined by the destination of the Request.

     Note

     The TgtID field can be remapped to a different value by the interconnect.

   - The SrcID is a fixed value for the Requester.
   - The Requester generates a TxnID value that is unique for that Requester.

   The TxnID value can be reused by the Requester after receiving the Comp response.

   - The Requester generates a new PGroupID value, or reuses a PGroupID value currently in use.
2. The recipient Home Node in the interconnect generates a request to the Subordinate.
   The identifier fields of the request to the Subordinate Node are generated as follows:
   - The TgtID is set to the value required for the Subordinate.
   - The SrcID is a fixed value for the Home.
   - The TxnID is a unique value generated by the Home.

   The TxnID value can be reused by the Home after receiving the Comp response.

   - The ReturnNID is set to the same value as the SrcID of the original request.
   - The ReturnTxnID is inapplicable and must be set to 0.

- The PGroupID is set to the same value as the PGroupID of the original request.

3. The recipient Home Node in the interconnect sends a Comp response to the Requester.
   The identifier fields of the Comp response to the Requester are generated as follows:
   - The TgtID is set to the same value as the SrcID of the original request.
   - The SrcID is a fixed value for the Home.
   - The TxnID is set to the same value as the TxnID of the original request.
4. The recipient Home Node can optionally send a Persist response to the Requester.
   The identifier fields of the optional Persist response from the Home Node to the Requester, not shown in Figure B2.27, are generated as follows:
   - The TgtID is set to the same value as the SrcID of the request.
   - The SrcID is a fixed value for the Home Node.
   - The TxnID is inapplicable and must be set to 0.
   - The PGroupID is set to the same value as the PGroupID of the request.

   The recipient Home Node can optionally send a combined CompPersist response to the Requester, instead of separate Comp and Persist responses.
   The identifier fields in the CompPersist response are generated as follows:
   - The TgtID is set to the same value as the SrcID of the original request.
   - The SrcID is a fixed value for the Home.
   - The TxnID is set to the same value as the TxnID of the original request.
   - The PGroupID is set to the same value as the PGroupID of the original request.
5. The Subordinate Node generates a Comp to the Home Node.
   The identifier fields of the Comp response from the Subordinate Node are generated as follows:
   - The TgtID is set to the same value as the SrcID of the request.
   - The SrcID is a fixed value for the Subordinate.
   - The TxnID is set to the same value as the TxnID of the request.
6. The Subordinate Node also generates a Persist response to either to the Requester or the Home.
   The identifier fields of the Persist response from the Subordinate Node are generated as follows:
   - The TgtID is set to the same value as the ReturnNID of the request.
   - The SrcID is a fixed value for the Subordinate.
   - The TxnID is inapplicable and must be set to 0.
   - The PGroupID is set to the same value as the PGroupID of the request.
7. The Subordinate Node can optionally send a combined CompPersist response to the Home Node, instead of separate Comp and Persist responses if the ReturnNID and SrcID of the request are the same value.
   The identifier fields of the CompPersist response from the Subordinate are generated as follows:
   - The TgtID is set to the same value as the SrcID of the request.
   - The SrcID is a fixed value for the Subordinate.
   - The TxnID is set to the same value as the TxnID of the request.
   - The PGroupID is set to the same value as the PGroupID of the request.

B2.5.3 Write transactions

This section describes the use of TxnID and DBID fields for Write transactions:

- B2.5.3.1 *CopyBack*
- B2.5.3.2 *WriteNoSnp transaction*
- B2.5.3.3 *WriteUnique transaction*
- B2.5.3.4 *StashOnce or StashOnceSep transaction*

B2.5.3.1 CopyBack

This section describes the use of the identifier fields for a CopyBack transaction.

Figure B2.28 shows the identifier value transfer.

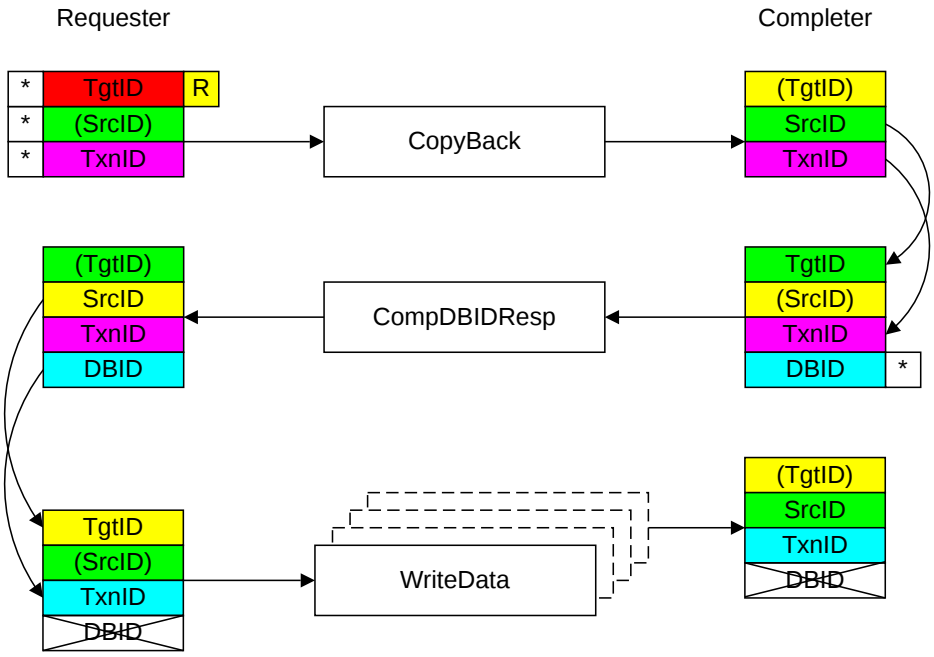

Figure B2.28: ID value transfer in a CopyBack

The required steps in the flow that Figure B2.28 shows are:

1. The Requester starts the transaction by sending a Request packet.
   The identifier fields of the request are generated as follows:
   - The TgtID is determined by the destination of the Request.

     Note

     The TgtID field can be remapped to a different value by the interconnect.

   - The SrcID is a fixed value for the Requester.
   - The Requester generates a unique TxnID field.
2. The Completer receives the Request packet and generates a CompDBIDResp response.
   The identifier fields of the response are generated as follows:

- The TgtID is set to the same value as the SrcID of the request.
- The SrcID is a fixed value for the Completer. This also matches the TgtID received.
- The TxnID is set to the same value as the TxnID of the request.
- The Completer generates a unique DBID value.

3. The Requester receives the CompDBIDResp response and sends the write data.
   The identifier fields of the write data are generated as follows:
   - The TgtID is set to the same value as the SrcID of the CompDBIDResp response. This can be different from the original TgtID of the request if the value was remapped by the interconnect.
   - The SrcID is a fixed value for the Requester.
   - The TxnID is set to the same value as the DBID value provided in the CompDBIDResp response.
   - The DBID field in the write data is not used.
   - The TgtID, SrcID, and TxnID fields must be the same for all write data packets.

   After receiving the CompDBIDResp response, the Requester can reuse the same TxnID value used in the request packet for another transaction.
4. The Completer receives the write data and uses the TxnID field, which now contains the DBID value that the Completer generated. This helps to determine which transaction to associate the write data.

   After receiving all write data packets, the Completer can reuse the same DBID value for another transaction.

B2.5.3.2 WriteNoSnp transaction

This section describes the use of the identifier fields for a WriteNoSnp transaction.

Figure B2.29 does not show the Memory Tagging supported TagGroupID flow. For TagGroupID transfer details, see B12.5 *Write transactions*.

Figure B2.29 shows the identifier value transfer for separate Comp and DBIDResp. The Completer can opportunistically combine the Comp and DBIDResp into a single CompDBIDResp response. The Requester can opportunistically combine NonCopyBackWriteData with CompAck.

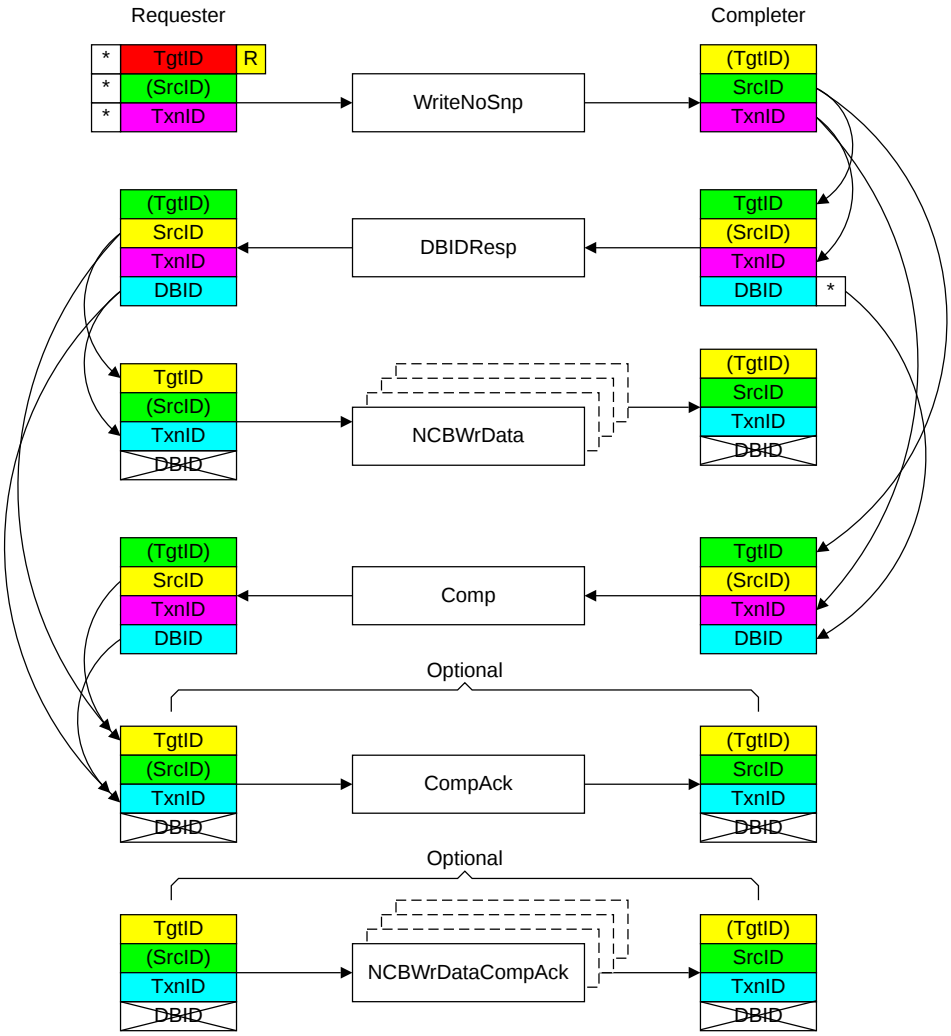

Figure B2.29: ID value transfer in a WriteNoSnp

The uses of the identifier fields are the same as for a transaction with a combined response with the additional requirements that:

- The identifier fields used for the separate DBIDResp and Comp responses must be identical.
- The TxnID value must only be reused by a Requester when both the DBIDResp and Comp responses have been received.

The required steps in the flow that Figure B2.29 shows are:

1. The Requester starts the transaction by sending a Request packet.
   The identifier fields of the request are generated as follows:
   - The TgtID is determined by the destination of the Request.

     Note

     The TgtID field can be remapped to a different value by the interconnect.

   - The SrcID is a fixed value for the Requester.

- The Requester generates a unique TxnID field.

2. The Completer receives the Request packet and generates a DBIDResp response.
   The identifier fields of the response are generated as follows:
   - The TgtID is set to the same value as the SrcID of the request.
   - The SrcID is a fixed value for the Completer. This also matches the TgtID received.
   - The TxnID is set to the same value as the TxnID of the request.
   - The Completer generates a unique DBID value.
3. The Requester receives the DBIDResp response and sends the write data.
   The identifier fields of the write data are generated as follows:
   - The TgtID is set to the same value as the SrcID of the DBIDResp response. This can be different from the original TgtID of the request if the value was remapped by the interconnect.
   - The SrcID is a fixed value for the Requester.
   - The TxnID is set to the same value as the DBID value provided in the DBIDResp response.
   - The DBID field in the write data is not used.
   - The TgtID, SrcID, and TxnID fields must be the same for all write data packets.
4. The Completer receives the write data and uses the TxnID field, which now contains the DBID value that the Completer generated, to determine which transaction the write data is associated with.
5. The Completer generates a Comp response when the transaction is complete.
   The identifier fields of the Comp response must be the same as the DBIDResp response and are generated as follows:
   - The TgtID is set to the same value as the SrcID of the request.
   - The SrcID is a fixed value for the Completer. This also matches the TgtID received.
   - The TxnID is set to the same value as the TxnID of the request.
   - The Completer uses the same DBID value as is used in the DBIDResp response.
6. The Requester sends a CompAck message, if required by the transaction, after receiving DBIDResp or Comp.
   The identifier fields of the CompAck are generated as follows:
   - The TgtID is set to the same value as the SrcID of the DBIDResp or Comp response.
   - The SrcID is a fixed value for the Requester. This also matches the TgtID that was received.
   - The TxnID is set to the same value as the DBID of the DBIDResp or Comp response.
   - The DBID field is not valid.

After receiving both the Comp and DBIDResp response, the Requester can reuse the same TxnID value for another transaction.

After receiving all the write data packets, the Completer can reuse the same DBID value for another transaction.

Note

There is no ordering requirement between the separate DBIDResp and Comp responses. It is required that the values used are identical when the two messages originate from the same source.

B2.5.3.3 WriteUnique transaction

This section describes the use of the identifier fields for a WriteUnique transaction.

Figure B2.30 does not show the Memory Tagging supported TagGroupID flow. For TagGroupID transfer details, see B12.5 *Write transactions*.

Under certain circumstances, the WriteUnique transaction can also include a CompAck response from the Requester to the Completer. In this case, the additional rules for the use of the identifier fields are:

- The TgtID, SrcID, and TxnID identifier fields of the CompAck response from the Requester to the Completer must be the same as the fields used for the write data, that is:
  - The TgtID is set to the same value as the SrcID of the CompDBIDResp response. If separate Comp and DBIDResp responses are given, the TgtID is set to the same value as the SrcID of either the Comp or DBIDResp response because the SrcID value in both must be identical. However, this can be different from the original TgtID of the request if the value has been remapped by the interconnect.
  - The SrcID is a fixed value for the Requester.
  - The TxnID is set to the same value as the DBID value provided in the CompDBIDResp response. If separate Comp and DBIDResp responses are given, the TxnID is set to the same value as the DBID of either the Comp or DBIDResp response because the DBID value in both must be identical.
  - The DBID field in the WriteData and in the CompAck is not used.
  - If a combined WriteData and CompAck response is sent, the TgtID is set to the same value as the SrcID in the Comp, DBIDResp, or CompDBIDResp, and the TxnID in the combined response is set to the same value as the DBID in Comp, DBIDResp, or CompDBIDResp.
- The Completer must receive all items of write data and the CompAck response before reusing the same DBID value for another transaction.

Figure B2.30 shows the identifier value transfer with a combined CompDBIDResp response.

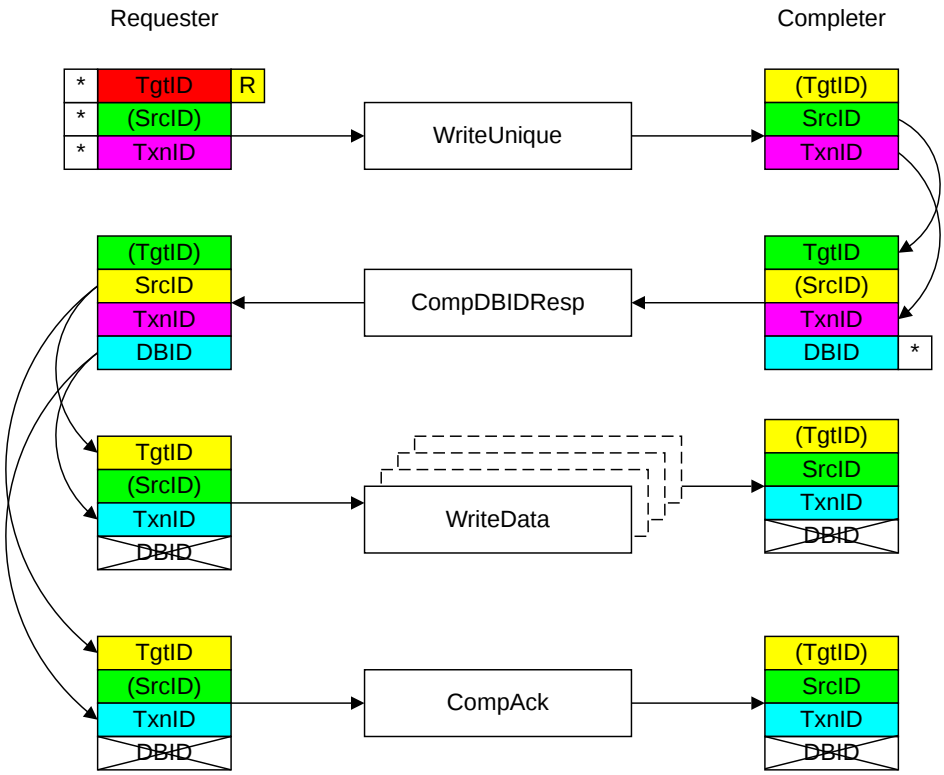

Figure B2.30: ID value transfer with a combined CompDBIDResp response

Figure B2.31 shows the identifier value transfer with a combined WriteData and CompAck response.

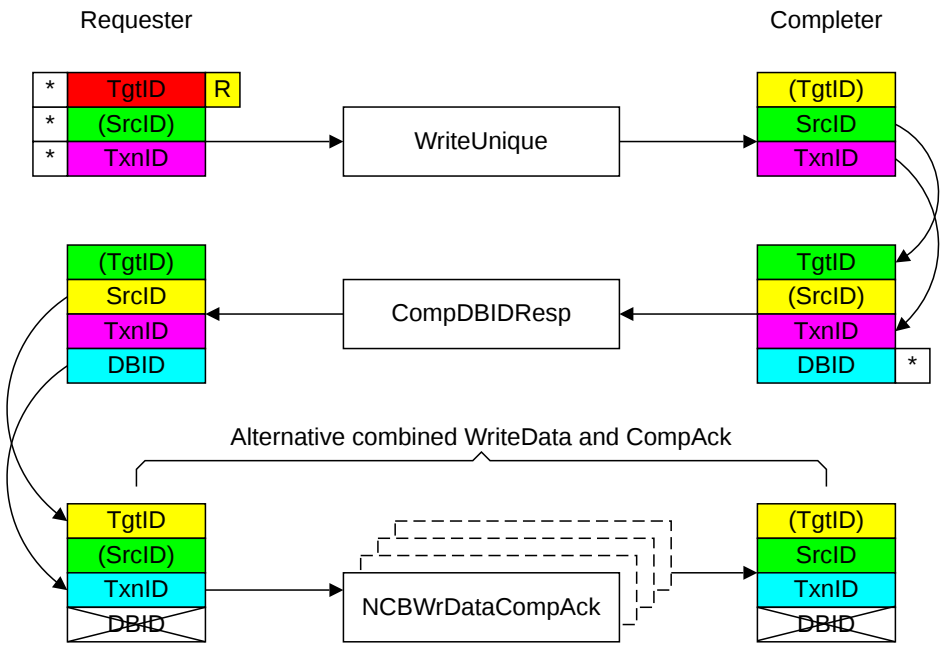

Figure B2.31: ID value transfer with a combined WriteData and CompAck response

B2.5.3.4 StashOnce or StashOnceSep transaction

This section describes the use of the identifier fields for a StashOnce or StashOnceSep transaction with DataPull. Figure B2.32 shows the identifier value transfer.

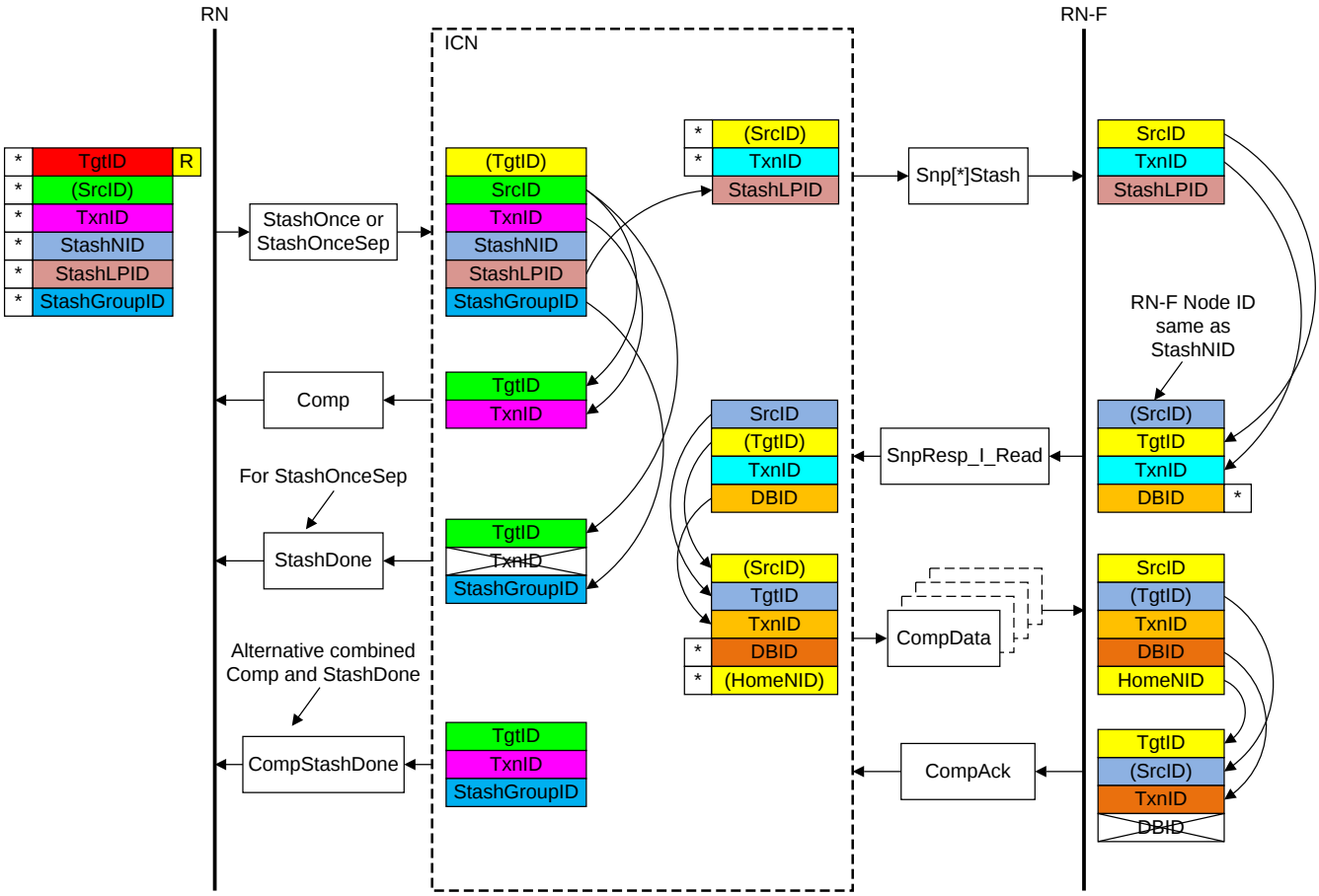

Figure B2.32: ID value transfer in a Stash transaction

The required steps in the flow that Figure B2.32 shows are:

1. The Requester starts the transaction by sending a Stash request packet.
   The identifier fields of the request are generated as follows:
   - The TgtID is determined by the destination of the Request.

     Note

     The TgtID field can be remapped to a different value by the interconnect.

   - The SrcID is a fixed value for the Requester.
   - The Requester generates a TxnID field that is unique for that Requester.
   - The Requester includes the StashNID field to indicate to which RN-F to send the Stash.
   - The Requester includes the StashLPID field to indicate the LP within the RN-F.
2. The Home Node in the interconnect receives the Stash request packet and sends the Comp response to the Request Node.
   The identifier fields of the Comp response are generated as follows:
   - The TgtID is set to the same value as the SrcID of the request.
   - The TxnID is set to the same value as the TxnID of the request.

3. The Home Node in the interconnect, for the StashOnceSep request, sends the StashDone response to the Request Node.
   The identifier fields of the StashDone response are generated as follows:
   - The TgtID is set to the same value as the SrcID of the request.
   - The TxnID is not valid.
   - StashGroupID is set to the same value as the StashGroupID of the request.

   Alternatively for the StashOnceSep request, the Home Node in the interconnect sends the combined CompStashDone response instead of separate Comp and StashDone responses to the Request Node.

   The identifier fields of the CompStashDone response are generated as follows:
   - The TgtID is set to the same value as the SrcID of the request.
   - The TxnID is set to the same value as the TxnID of the request.
   - The StashGroupID is set to the same value as the StashGroupID of the request.
4. The Home Node in the interconnect generates a snoop with Stash to the appropriate RN-F.
   The identifier fields of the request are generated as follows:
   - The SrcID is a fixed value for the Home.
   - The TxnID is a unique value generated by the Home.
   - The StashLPID is set to the same value as the StashLPID of the original request.

   Note

   A Snoop request does not include a TgtID field.
5. The snooped RN-F generates a Snoop response. In this example, a Data Pull indication is included.
   The identifier fields of the Snoop response are generated as follows:
   - The TgtID is set to the same value as the SrcID of the request.
   - The SrcID is a fixed value for the RN-F.
   - The TxnID is set to the same value as the TxnID of the request.
   - The DBID field is a unique value generated by the RN-F.
6. The Home provides the read data.
   The identifier fields of the read Data response are generated as follows:
   - The TgtID is set to the same value as the SrcID of the Snoop response.
   - The SrcID is a fixed value for the Home.

   Note

   In this example, the read data is being provided by the Home.
   - The TxnID is set to the same value as the DBID of the Snoop response.
   - The DBID field is a unique value generated by Home.
   - The HomeNID is a fixed value for the Home.
7. The RN-F receives the read data and sends a completion acknowledge, CompAck, response.
   The identifier fields of the CompAck are generated as follows:
   - The TgtID is set to the same value as the HomeNID of the read data.

- The SrcID is a fixed value for the RN-F. This also matches the TgtID received.
- The TxnID is set to the same value as the DBID of the read data.
- The DBID field is not valid.

B2.5.4 DVMOp transaction

The use of TgtID, SrcID, TxnID, and DBID identifier fields for a DVMOp transaction is identical to those in the B2.5.3.2 *WriteNoSnp transaction*.

B2.5.5 Transaction requests with Retry

For transactions that receive a RetryAck response, there are specific rules on how the identifier fields are used.

See B2.9 *Request Retry*, for more details on the Retry mechanism, and B2.5.6 *Protocol Credit Return transaction* for rules about the return of unused credits.

Figure B2.33 shows the identifier value transfer.

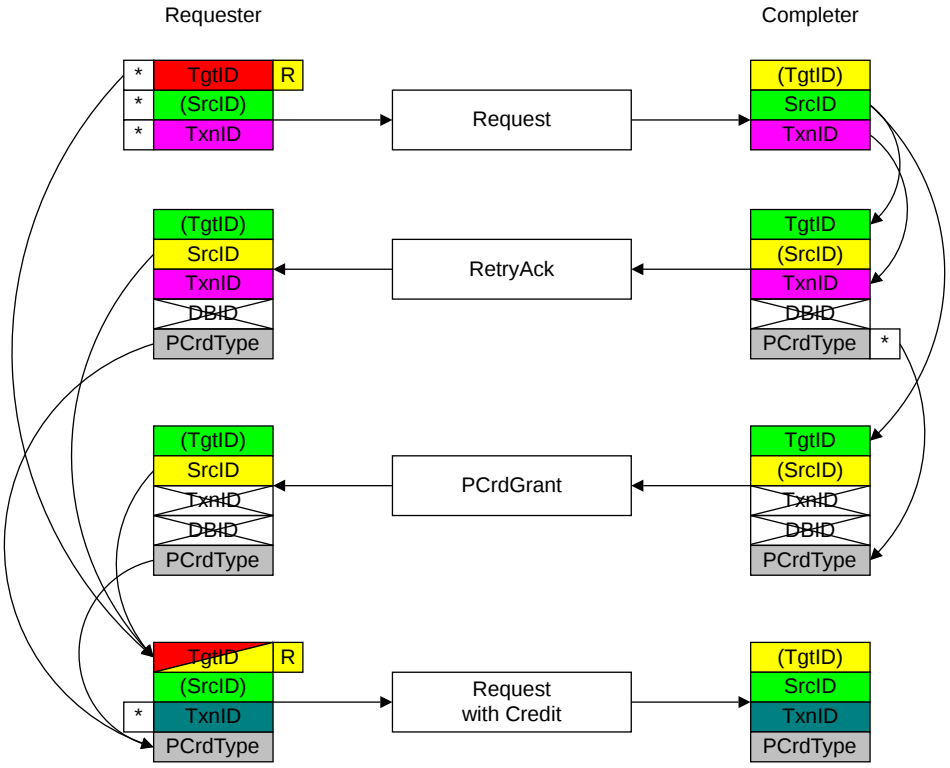

Figure B2.33: ID value transfer in a transaction request with retry

The required steps in the flow that Figure B2.33 shows are:

1. The Requester starts the transaction by sending a Request packet.
   The identifier fields of the request are generated as follows:
   - The TgtID is determined by the destination of the Request.

> **Note**
>
> The TgtID field can be remapped to a different value by the interconnect.

- The SrcID is a fixed value for the Requester.
- The Requester generates a unique TxnID field.

2. The Completer receives the Request packet and determines that a RetryAck response will be sent. The identifier fields of the RetryAck response are generated as follows:
   - The TgtID is set to the same value as the SrcID of the request.
   - The SrcID is a fixed value for the Completer. This also matches the TgtID received.
   - The TxnID is set to the same value as the TxnID of the request.
   - The DBID field is not valid.
   - The Completer uses a PCrdType value that indicates the type of credit required to retry the transaction.
3. When the Completer is able to accept the retried transaction of a given PCrdType, a credit to the Requester is sent using the PCrdGrant response.
   The identifier fields of the PCrdGrant response are generated as follows:
   - The TgtID is set to the same value as the SrcID of the request.
   - The SrcID is a fixed value for the Completer. This also matches the TgtID of the request.
   - The TxnID field is not used and must be set to 0.
   - The DBID field is not used and must be set to 0.
   - The PCrdType value is set to the type required to issue the original transaction again.
4. The Requester receives the credit grant and reissues the original transaction by sending a Request packet. The identifier fields of the request are generated as follows:
   - The TgtID is set to either the same value as the SrcID of the RetryAck response, which is also the same as the SrcID of the PCrdGrant response, or the value used in the original request.
   - The SrcID is a fixed value for the Requester.
   - The Requester generates a unique TxnID field. This is permitted, but not required, to be different from the original request that received a RetryAck response.
   - The PCrdType value is set to the PCrdType value in the RetryAck response to the original request, which is also the same as the PCrdType of the PCrdGrant response.

B2.5.6 Protocol Credit Return transaction

A P-Credit Return transaction uses the PCrdReturn Request to return a granted, but no longer required, credit. The TgtID, SrcID, and TxnID requirements are:

- The Requester sends the Protocol Credit Return transaction by sending a PCrdReturn Request packet. The identifier fields of the request are generated as follows:
  - The TgtID must match the SrcID of the credit that was obtained.
  - The SrcID is a fixed value for the Requester.
  - The TxnID field is not used and must be set to 0.

The PCrdType must match the value of the PCrdType in the original PCrdGrant that was required to issue the original transaction again.

There is no response or use made of the DBID field associated with Protocol Credit Return transactions.

B2.6 Ordering

This section describes the mechanisms that the protocol includes to support system ordering requirements. It contains the following sections:

- B2.6.1 *Multi-copy atomicity*
- B2.6.2 *Completion response and ordering*
- B2.6.3 *Completion acknowledgment*
- B2.6.4 *Ordering semantics of RespSepData and DataSepResp*
- B2.6.5 *Transaction ordering*

For the meaning of the terms EWA, Device, and Cacheable, see B2.7.3 *Memory Attributes*.

B2.6.1 Multi-copy atomicity

The memory model used in this specification requires multi-copy atomicity. All compliant components must ensure that all write requests are multi-copy atomic. A write is defined as multi-copy atomic if both of the following conditions are true:

- All writes to the same location are serialized, therefore observed in the same order by all Requesters. Some Requesters could not observe all of the writes.
- A read of a location does not return the value of a write until all Requesters observe that write.

In this specification, two addresses are considered to be the same with respect to coherence,observability, and hazarding if their cache line addresses and *Physical Address Space* (PAS) attributes are the same.

B2.6.2 Completion response and ordering

Table B2.7 shows the various transaction responses and any ordering guarantees they provide with respect to later transactions, either from the same agent or from another agent.

Table B2.7: Completion response and ordering

Transaction	Location	Response	Outcome
Read	Cacheable	CompData or DataSepResp	The transaction is observable to a later transaction from any agent to the same location.
	Cacheable	RespSepData	No earlier transaction will send a snoop to this Requester. All later transactions send a snoop only if required after the Home receives the CompAck response for this transaction.
	Non-cacheable or Device	RespSepData or CompData	The transaction is observable to a later transaction from any agent to the same endpoint address range.
Write or Atomic	Cacheable	Comp or CompData	The transaction is observable to a later transaction from any agent to the same location.
	Non-cacheable or Device	Comp or CompData	The transaction is observable to a later transaction from any agent to the same endpoint range.
Dataless except for StashOnceSep and CleanInvalidPoPA		Comp	The transaction is observable to a later transaction from any agent to the same memory location.

Continued on next page

Table B2.7 – *Continued from previous page*

Transaction	Location	Response	Outcome
CleanSharedPersist		Comp	Any data written earlier to the same memory location is made persistent.
CleanInvalidPoPA		Comp	The transaction is observable to a later transaction from any agent to the same memory location in any PAS. Additional Cache Maintenance Operations could be required in the other Physical Address Spaces to ensure any data written earlier is fully visible to those Physical Address Spaces.
Combined Write		CompCMO (non-PoPA CMO)	CMO, PCMO, and write operations are observable to a later transaction from any agent to the same memory location.
		CompCMO (PoPA CMO)	CMO and write operations are observable to a later transaction from any agent to the same memory location in any PAS. Additional Cache Maintenance Operations could be required in the other Physical Address Spaces to ensure that any data written earlier is fully visible to those Physical Address Spaces.
		Comp	See outcome of Comp for Write transaction within this table.
CleanSharedPersistSep		Persist	The data written earlier to the same memory location is made persistent.
Combined Write with PCMO		Persist	The data write in the Combined Write is made persistent. All earlier writes on the same line are also made persistent.
StashOnceSep		Comp	The Completer accepts the request and does not send a RetryAck response.
		StashDone	The transaction is observable to a later transaction from any agent to the same memory location.

Note

The size of the endpoint address range is IMPLEMENTATION DEFINED.

In a combined *Ordered Write Observation* (OWO) Write request, if the write is canceled and the Requester sends a WriteDataCancel, the required cache maintenance on the write data is not carried out. The Requester must resend both the Write request and the CMO:

- If a Write with PCMO had its write canceled:
  - The Persist response for the combined request does not indicate that the write data is made persistent.
  - The Requester must resend the PCMO either combined with the resent Write request or after the resent Write request succeeds.
- If a Write with CMO had its write canceled:
  - The Requester must resend the CMO, either combined with the re-sent Write request or after the resent Write request succeeds.

Note

The size of an endpoint address range is IMPLEMENTATION DEFINED. Typically, this is:

- The size of a peripheral device, for a region used for peripherals.
- The size of a cache line, for a region used for memory.

A Cacheable location can be determined by the assertion of the MemAttr[2] Cacheable bit in the request. A Non-cacheable or Device location can be determined by the deassertion of the MemAttr[2] Cacheable bit in the request.

A component must only give a Comp or CompDBIDResp response when all observers are guaranteed see the result of the atomic operation.

A Comp response in a canceled write only implies that the transaction loop is completed and makes no statement regarding the completion of coherency action initiated by the write. Therefore, the Completer is permitted to send a Comp as soon as receiving a WriteDataCancel response without dependency on either the processing of the write request or the completion of any snoops sent due to the write.

B2.6.3 Completion acknowledgment

The relative ordering of transactions issued by a Requester, and Snoop transactions caused by transactions from different Requesters, is controlled by the use of a completion acknowledge, CompAck, response. This ensures that a Snoop transaction that is ordered after the transaction from the Requester is guaranteed to be received after the transaction response.

The sequencing of the completion of a Read transaction and the sending of CompAck is as follows:

1. An RN-F sends a CompAck after receiving Comp, RespSepData, or CompData, or both RespSepData and DataSepResp.
2. An HN-F, except in the case of ReadNoSnp and ReadOnce*, waits for CompAck before sending a subsequent snoop to the same address. For CopyBack transactions, WriteData acts as an implicit CompAck and an HN-F must wait for WriteData before sending a snoop to the same address.

This sequence guarantees that an RN-F receives completion for a transaction and a snoop to the same cache line in the same order as they are sent from an HN-F. This ensures transactions to the same cache line are observed in the correct order.

When an RN-F has a transaction in progress that uses CompAck, except for ReadNoSnp and ReadOnce*, a Snoop request is guaranteed to not be received to the same address between the point that Comp is received and the point that CompAck is sent.

For WriteNoSnp, WriteUnique, and their Combined Write variants, that require a CompAck message, a Request Node sends the CompAck after receiving the Comp, DBIDResp, or CompDBIDResp response.

The use of CompAck for a transaction is determined by the Requester setting the ExpCompAck field in the original request. The rules for a Request Node setting the ExpCompAck field and generating a CompAck response are as follows:

- An RN-F must include a CompAck response in all Read transactions except ReadNoSnp and ReadOnce*.
- An RN-F is permitted, but not required, to include a CompAck response in ReadNoSnp and ReadOnce* transactions.
- An RN-F must not include a CompAck response in StashOnce*, CMO, Atomic, or Evict transactions.
- An RN-I or RN-D is permitted, but not required, to include a CompAck response in Read transactions.
- An RN-I or RN-D must not include a CompAck response in Dataless or Atomic transactions.

- A Request Node that wants to make use of DMT must include a CompAck response in ordered ReadNoSnp and ReadOnce* transactions.
- For Write transactions, CompAck can only be used for:
  - WriteUnique, WriteNoSnp, and their Combined Write variants, when they require OWO guarantees. See B2.6.5.3 *Streaming Ordered Write transactions*.
  - CopyBack write transactions where Home has provided a Comp response, indicating that the Requester must not send CBWrData. When Home provides a Comp response, a CompAck must be sent by the Requester regardless of the original ExpCompAck value. See B2.3.2.3 *CopyBack Write*.

For transactions between a Request Node and a Home Node, where the Home Node is the Completer, the Home Node must support the use of CompAck for all transactions that are required or permitted to use CompAck.

A Subordinate Node is not required to support the use of CompAck.

A Requester, such as an HN-F or HN-I that communicates with an SN-F or SN-I respectively, must not send a CompAck response.

Table B2.8 shows the Request types that require a CompAck response, and the corresponding Requester types that are required to provide that response. The following key is used:

Y Yes, required

N No, not required

H Dependent on transaction flow chosen by Home in response to the CopyBack Write request

O Optional

- Not applicable

Table B2.8: Requester CompAck requirement

Request type	CompAck required	
	RN-F	RN-D, RN-I
ReadNoSnp	O	O
ReadOnce*	O	O
ReadClean	Y	-
ReadNotSharedDirty	Y	-
ReadShared	Y	-
ReadUnique	Y	-
ReadPreferUnique	Y	-
MakeReadUnique	Y	-
CleanUnique	Y	-
MakeUnique	Y	-
CleanShared	N	N
CleanSharedPersist*	N	N
CleanInvalid	N	N
CleanInvalidPoPA	N	N

Continued on next page

Table B2.8 – *Continued from previous page*

Request type	CompAck required	
	RN-F	RN-D, RN-I
MakeInvalid	N	N
WriteBack	H	-
WriteClean	H	-
WriteUnique	O	O
WriteUniqueZero	N	N
Evict	N	-
WriteEvictFull	H	-
WriteEvictOrEvict	H	-
WriteNoSnp	O	O
WriteNoSnpDef	N	N
WriteNoSnpZero	N	N
Atomics	N	N
StashOnce*	N	N

In a Combined Write transaction, the CompAck requirement is the same as the CompAck requirement for the type of Write in the Combined Write transaction.

B2.6.4 Ordering semantics of RespSepData and DataSepResp

When a Requester receives the first DataSepResp, the Read transaction can be considered to be globally observed. This is because there is no action which can modify the read data received.

When a Requester receives a RespSepData response from Home, the relevent request has been ordered at Home. The Requester does not receive any snoops for transactions that are scheduled before the RespSepData response. Before sending RespSepData response to the Requester, the Home must ensure that no Snoop transactions are outstanding to that Requester to the same address. Receiving of RespSepData does not guarantee that Home has completed snooping of other agents in the system.

When the Requester gives a completion acknowledge, CompAck, response, this Requester is indicating to accept responsibility to hazard snoops for any transaction that is scheduled after the CompAck response. The following rules apply:

- For all transactions, except as described immediately below, the CompAck must be sent after the RespSepData response is received. It is permitted, but not required, to wait for the DataSepResp response before the CompAck is given.
- For ReadOnce and ReadNoSnp transactions with an ordering requirement, that is, Order field is set to `0b10` or `0b11` and ExpCompAck field is asserted, it is required that the CompAck is given only after both DataSepResp and RespSepData responses are received.

Note

It is required that CompAck must not be given when only DataSepResp is received.

B2.6.5 Transaction ordering

In addition to using a Comp response to order a sequence of requests from a Requester, this specification also defines mechanisms for ordering of requests between a Request Node, Home Node pair and an HN-I, SN-I pair. Between an HN-F, SN-F pair and HN-I, SN-I pair, the order field is used to obtain a Request Accepted acknowledgment.

Requester Order between an RN, HN pair and an HN-I, SN-I pair is supported by the Order field in a request. The Order field indicates that the transaction requires one of the following forms of ordering:

Request Order This guarantees the order of multiple transactions, from the same agent, to the same address location.

Endpoint Order This guarantees the order of multiple transactions, from the same agent, to the same endpoint address range.

Ordered Write Observation, OWO This guarantees the observation order by other agents in the system, for a sequence of Write transactions from a single agent.

Request Accepted This guarantees that the Completer sends a positive acknowledgment only when accepting the Read request.

Table B2.9 shows the Order field encodings.

Table B2.9: Order value encodings

Order[1:0]	Description	Permitted between
0b00	No ordering required	All
0b01	Request accepted	HN-F to SN-F, and HN-I to SN-I
	Reserved	RN to HN
0b10	Request Order/OWO	RN to HN[a]
	Request Order	HN-I to SN-I
	Reserved	HN-F to SN-F
0b11	Endpoint Order	RN to HN, and HN-I to SN-I
	Reserved	HN-F to SN-F

[a] Request Order when ExpCompAck = 0. OWO when ExpCompAck = 1.

B2.6.5.1 Ordering requirements

A Requester that changes the ordering requirements of a transaction to a stronger ordering requirement, is required to be consistent in changing the ordering requirement of Request Order to Endpoint Order on all its transactions.

The Order field must only be set to a non-0 value for the following transactions:

- ReadNoSnp
- ReadNoSnpSep
- ReadOnce*
- WriteNoSnp
- WriteNoSnpDef
- WriteNoSnp*CMO

- WriteNoSnpZero
- WriteUnique
- WriteUnique*CMO
- WriteUniqueZero
- Atomic

When a ReadNoSnp or ReadOnce* transaction requires Request Order or Endpoint Order:

- The Requester requires a ReadReceipt to determine when it can send the next ordered request.
- At the Completer a ReadReceipt means the request has reached the next ordering point that maintains requests in the order they were received:
  - Requests that require Request Order maintain order between requests to the same address from the same source.
  - Requests that require Endpoint Order maintain order between requests to the same endpoint address range from the same source.
- A Completer that is capable of sending separate Non-data and Data-only responses can send RespSepData response instead of ReadReceipt and achieve the same functional behavior.

When a WriteNoSnp, WriteNoSnpDef, WriteNoSnpZero, or a Non-snoopable Atomic transaction requires Request Order or Endpoint Order:

- The Requester requires a DBIDResp or DBIDRespOrd to determine when sending the next ordered request.
- The Completer sending a DBIDResp or DBIDRespOrd response means that a data buffer is available, and that the Write request has reached a PoS that maintains requests in the order they were received:
  - For requests that require Request Order, the Completer maintains order between requests to the same address from the same source.
  - For requests that require Endpoint Order, the Completer maintains order between requests to the same endpoint address range from the same source.

When a WriteUnique transaction without ExpCompAck asserted, or a WriteUniqueZero or a Snoopable Atomic transaction require Request Order:

- The Requester requires a DBIDResp or DBIDRespOrd to determine when sending the next ordered request.
- The Completer sending a DBIDResp or DBIDRespOrd response means that maintains order between requests to the same address from the same source.

Additionally, when a Completer sends DBIDRespOrd for a request with a no order or Request Order requirement, the Completer guarantees to order all subsequent no order or Request Order received requests to the same address from the same source against this request, where these later received requests are of any transaction type, not necessarily Write transactions. When the write includes a CMO, the order is guaranteed against both the write and the CMO.

When a WriteUnique, WriteNoSnp, or one of their Combined Write variants requires OWO:

- CompAck is required. The Request Node must assert ExpCompAck.
- The Request Node requires a DBIDResp or DBIDRespOrd.
- The Completer is a PoS. A PoS sending DBIDResp or DBIDRespOrd means:
  - A data buffer is available.
  - The PoS guarantees that the completion of the coherence action on this write does not depend on completion of the coherence action on a subsequent write that requires OWO.
  - The write is not made visible until CompAck is received.

All architectural mechanisms applicable to increasing streaming efficiency and corresponding constraints are defined in B2.6.5.3 *Streaming Ordered Write transactions*.

When a ReadNoSnp or ReadNoSnpSep has the Order field set to `0b01`, a ReadReceipt response from the Completer guarantees that the Completer has accepted the request and does not send a RetryAck response.

B2.6.5.1.1 Read Request order example

Figure B2.34 shows the request ordering of three Read requests.

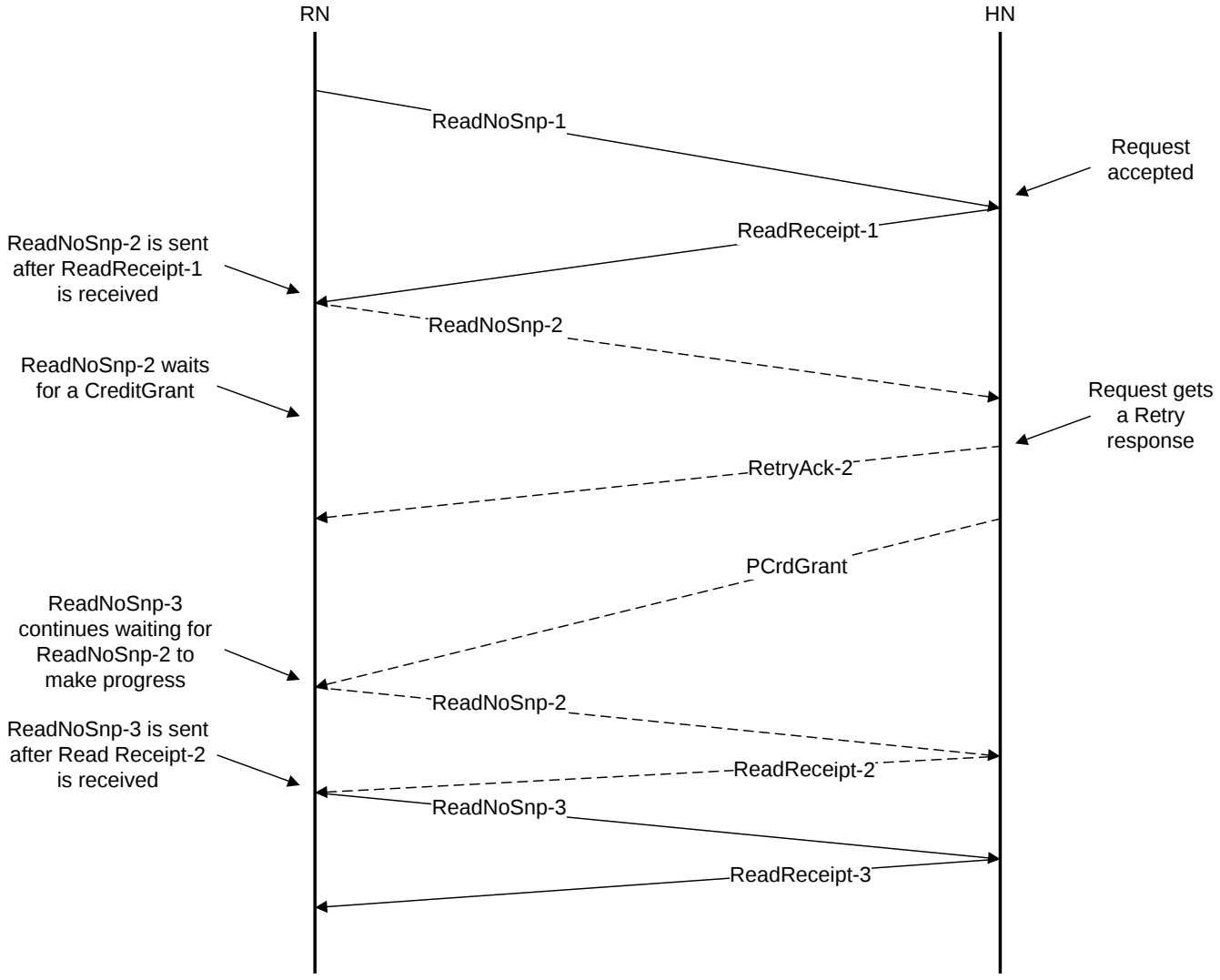

Figure B2.34: Series of ordered Read requests

Three ordered requests are sent from Request Node to Home Node in Figure B2.34 as follows:

1. The Request Node sends the ReadNoSnp-1 request to the Home Node.
2. The Home Node accepts the request and returns the ReadReceipt-1 response to the Request Node.
3. After the ReadReceipt-1 response is received, the Request Node sends the ReadNoSnp-2 request to the Home Node.

4. The Home Node cannot immediately accept the ReadNoSnp-2 request and returns the RetryAck-2 response to the Request Node.
5. The Request Node must now wait for a PCrdGrant to be sent from the Home Node before resending the ReadNoSnp-2 request. The Request Node does not send ReadNoSnp-3 at this point, to order ReadNoSnp-3 behind ReadNoSnp-2. This ordering requires that ReadNoSnp-2 must be accepted at the Home Node before ReadNoSnp-3 is sent to the Home Node.
6. After receipt of an appropriate PCrdGrant, the Request Node resends the ReadNoSnp-2 request.
7. The Home Node accepts the request and returns a ReadReceipt-2 response to the Request Node.
8. After receipt of the ReadReceipt-2 response, the Request Node sends the ReadNoSnp-3 request to the Home Node.
9. The Home Node accepts the request and returns the ReadReceipt-3 response to the Request Node.
10. Each of the read transactions is completed with the Requester receiving a completion and data. This is not shown in Figure B2.34.

Note

Figure B2.34 shows a single ordered stream of three reads from the Request Node. However, a Request Node can have multiple streams of reads, so requests must be ordered within a stream. However, ordering dependency does not exist between streams. For example, when the streams are from different threads within the Request Node. In this case, the Request Node waits for the ReadReceipt of the previous request from the same thread only before sending out the next ordered request from that stream.

B2.6.5.2 CopyBack Request order

An RN-F must wait for the CompDBIDResp or Comp response to be received for an outstanding CopyBack transaction before issuing another request to the same cache line.

- It is permitted for an Atomic transaction with SnoopMe asserted to be issued before the CompDBIDResp or Comp response is received for an outstanding CopyBack to the same cache line.
- It is permitted for a CopyBack transaction to be issued before the CompDBDResp or Comp response is received for an outstandng Atomic transaction, with SnoopMe asserted, to the same cache line.

B2.6.5.3 Streaming Ordered Write transactions

The architectural mechanisms applicable to increasing Ordered Write Observation (OWO) Write streaming efficiency, and the corresponding constraints, are applicable to WriteUnique and WriteNoSnp transactions only.

If a Requester requires a sequence of Write transactions to be observed in the same order as they are issued, the Requester can wait for completion for a write before issuing the next write in the sequence. Such an observation ordering is typically termed OWO. This specification provides a mechanism termed Streaming Ordered Writes to more efficiently stream such ordered Write transactions.

The Streaming Ordered Write mechanism relies on the use of the OWO ordering requirement and CompAck. Responsibilities of Requesters and HN-F when utilizing the Streaming Ordered Write solution are:

- The Requester must set the Order field to `0b10` and set ExpCompAck on the Write request.
- The OWO requirement in a Write request indicates to the HN-F that the completion of coherence action on this write must not depend on completion of coherence action on a subsequent write.
- The Requester must wait for DBIDResp, DBIDRespOrd, CompDBIDResp, or Comp for a Write transaction before sending the next Write request.

- The Requester must send a CompAck response after receiving DBIDResp, DBIDRespOrd, CompDBIDResp, or Comp responses for the corresponding writes and Comp or CompDBIDResp responses for all earlier related ordered writes. If write data is to be sent, the Requester is permitted, but not required, to combine the CompAck response with the WriteData response into a NonCopyBackWriteDataCompAck response. When the Requester uses the combined CompAck and WriteData response for a transaction, a combined response must be sent for all WriteData transfers in that transaction. The method by which a Requester determines if a group of ordered writes are related is IMPLEMENTATION DEFINED.

Note

Waiting to send CompAck response until all prior, related ordered writes have received their Comp responses ensures the operations at their respective HN-Fs have completed. Any Requester observing the write associated with the CompAck response also observes all prior, related ordered writes.

- The Requester that receives DBIDResp* and is ready to send CompAck must not wait for Comp to send CompAck.
- HN-F must wait for a CompAck response from the Request Node before deallocating a Write transaction and making the write visible to other observers.

B2.6.5.3.1 Optimized Streaming Ordered Write transactions

The writes in this section refer to WriteUnique or WriteNoSnp only. The Streaming Ordered Writes mechanism can be further optimized. If a previously sent write is to a different target, the Requester does not need to wait for the DBIDResp* for the request before sending the next ordered write. However, if the interconnect can remap the TgtID, the Requester must presume that all Write transactions are targeting the same HN-F and must not use the optimized version of the Streaming Ordered Writes flow.

An implementation using an optimized or non-optimized Streaming Ordered Writes solution must avoid deadlock and livelock situations.

Note

A technique for avoiding resource-related deadlock or livelock issues is to limit Streaming Ordered Writes optimization to one Requester in the system. All other Requesters in the system can use the Streaming Ordered Writes solution without the optimization.

In a typical system, the optimized Streaming Ordered Writes solution is most beneficial to an RN-I that is a conduit for PCIe style, non-relaxed order, Snoopable writes. In most systems, one RN-I hosting this type of PCIe traffic is adequate.

OWO Writes can be used by more than one Requester by making use of WriteDataCancel messages to avoid resource related deadlocks and livelocks.

Figure B2.35 shows a typical transaction flow in which an RN-I uses Streaming Ordered WriteUnique transactions. This flow prevents a read acquiring the new value of Write-B before Write-A has completed.

Note

For clarity, the Write-B DBIDResp* and the NCBWrData flow is omitted from Figure B2.35.

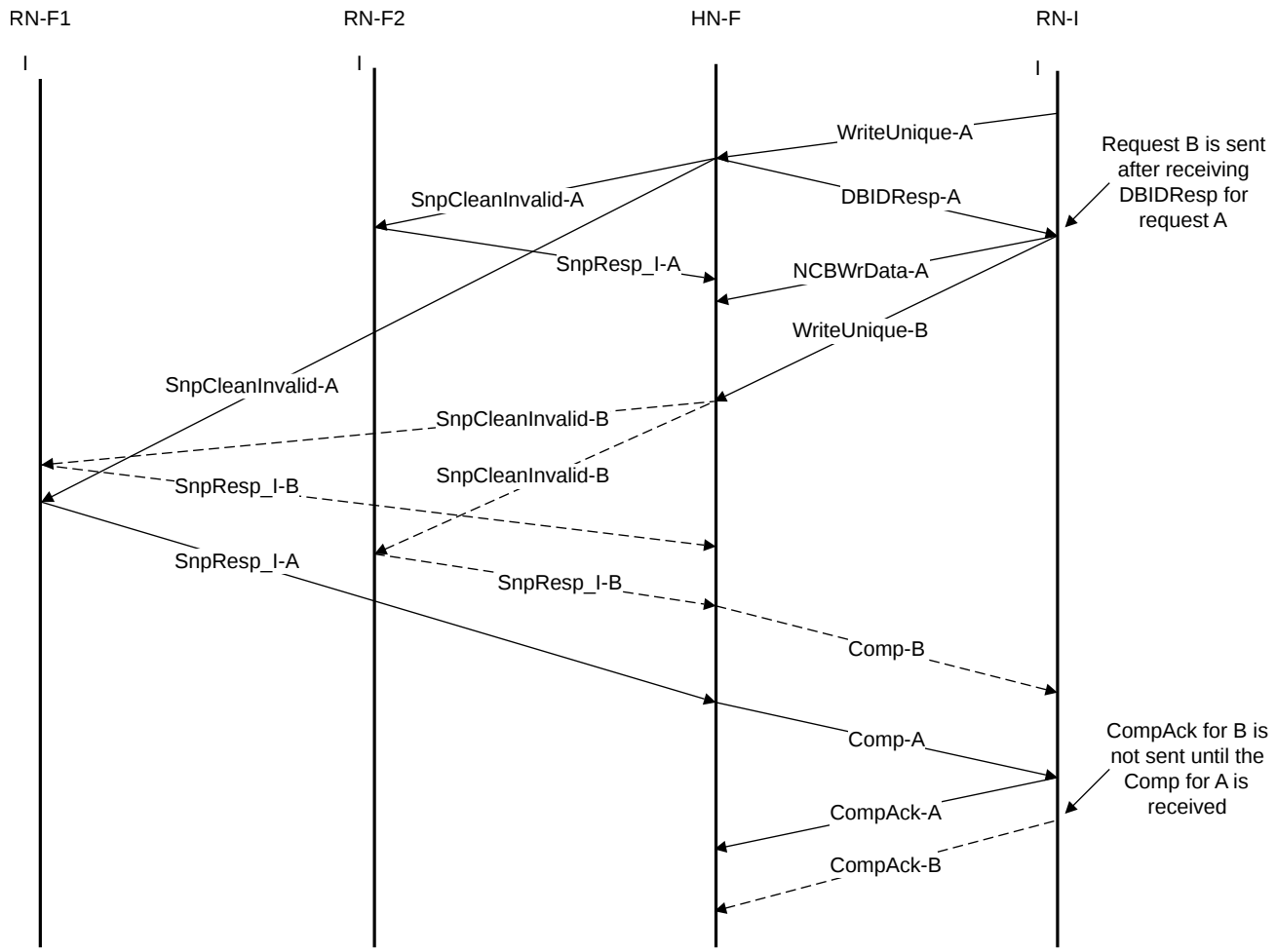

Figure B2.35: Streaming Ordered WriteUnique transactions flow

The Streaming Ordered WriteUnique transaction flow is as follows:

1. RN-I issues WriteUnique-A to the Home.
2. The Home responds with DBIDResp and issues SnpCleanInvalid-A to RN-F1 and RN-F2.
3. RN-I sends the write data associated with WriteUnique-A and issues the next ordered WriteUnique request to the Home, shown in Figure B2.35 as WriteUnique-B.
4. WriteUnique-A and WriteUnique-B are to different addresses. The Home sends out SnpCleanInvalid-B snoops to RN-F1 and RN-F2 for WriteUnique-B without waiting for responses for WriteUnique-A transaction snoops.
5. The Snoop responses for the WriteUnique-B transaction are received by the Home before the Snoop responses for the WriteUnique-A transaction are received. Home sends Comp to RN-I for the WriteUnique-B transaction.
6. The Home waits for Snoop responses for the WriteUnique-A transaction to be received. Once received, the Home can send Comp to RN-I for the WriteUnique-A transaction.
7. The Requester, RN-I, receives Comp-B and waits to receive Comp-A before proceeding with the next response.

8. The RN-I sends a CompAck response for the WriteUnique-A transaction when it is ready to make the write observable.
9. The RN-I sends a CompAck response for the WriteUnique-B transaction when it is ready to make the write observable.

B2.7 Address, Control, and Data

A transaction includes attributes defining the manner in which the transaction is handled by the interconnect. These include the address, memory attributes, snoop attributes, and data formatting. Each attribute is defined in this section.

In this section, unless explicitly stated otherwise, a reference to a Write transaction includes both the individual Write transaction and the corresponding Combined Write transaction.

B2.7.1 Address

The CHI protocol supports:

- *Physical Address* (PA) of 44 bits to 52 bits, in 1 bit increments.
- *Virtual Address* (VA) of 49 bits to 53 bits.

The REQ and SNP packet Addr fields are specified as follows:

- REQ channel: Address[(MPA-1):0]
- SNP channel: Address[(MPA-1):3]

MPA is the Maximum PA supported.

Table B2.10 shows the relationship between the physical address field width and the supported virtual address.

Table B2.10: Addr field width and supported Physical Address and Virtual Address size

REQ Addr field width (bits)	Maximum supported (bits)	
	Physical Address	Virtual Address
44	44	49
45	45	51
46 to 52	46 to 52	53

See B8.3 *DVMOp field value restrictions* for DVM payload mapping in the REQ and SNP fields with different Addr field widths.

The Req_Addr_Width parameter is used to specify the maximum PA in bits that is supported by a component. Valid values for this parameter are 44 to 52, when not specified, the parameter takes the default value of 44.

B2.7.2 Physical Address Space, PAS

The PAS of an access is determined using a combination of the NSE and NS fields. Table B2.11 shows the encoding for the PAS.

Table B2.11: Physical Address Space encodings

[NSE, NS]	Description
00	Secure
01	Non-secure

Continued on next page

Table B2.11 – *Continued from previous page*

[NSE, NS]	Description
10	Root
11	Realm

For Snoopable transactions, the PAS can be considered additional address information that defines four address spaces. Any aliasing between the different Physical Address Spaces must be handled correctly.

Note

Hardware coherency does not manage coherency between the four address spaces. See B2.6.1 *Multi-copy atomicity*.

The PAS encoding requirements are:

- Can take any value in any Read, Dataless, Write, and Atomic transactions.
- Can take any value in PrefetchTgt transaction.
- Is not applicable in the DVMOp or PCrdReturn transaction, and must be set to 0.

B2.7.3 Memory Attributes

The *Memory Attributes* (MemAttr) consist of *Early Write Acknowledgment* (EWA), Device, Cacheable, and Allocate.

B2.7.3.1 EWA

EWA indicates whether the Write completion response for a transaction:

- Can come from an intermediate point in the interconnect, such as a Home Node.
- Must come from the final endpoint at the target destination.

If EWA is asserted, the Write completion response for the transaction can come from an intermediate point or from the endpoint. A completion that comes from an intermediate point must provide the same guarantees required by a Comp as described in B2.6.2 *Completion response and ordering*.

If EWA is deasserted, the write completion response for the transaction must come from the endpoint.

Note

It is permitted, but not required, for an implementation not to use the EWA attribute. In this instance, completion must be given from the endpoint.

The EWA assertion requirements are:

- Can take any value in:
  - A ReadNoSnp and ReadNoSnpSep transaction
  - A WriteNoSnp and WriteNoSnpDef transaction.

Note

For a WriteNoSnpDef transaction, it is expected that the response comes from the final target endpoint, regardless of EWA value.

An early response from an intermediate component can stop the original Requester observing a valid Defer response from the final endpoint, reducing the effectiveness of the deferrable write transaction.

- CMO transactions.
- Atomic transactions.
- Must be asserted in any:
  - Read transaction that is not a ReadNoSnp, ReadNoSnpSep, or CMO transaction.
  - Dataless transaction that is not a CMO transaction.
  - Write transaction that is not a WriteNoSnp or WriteNoSnpDef transaction.
- Is inapplicable and:
  - Must be set to 0 in the DVMOp or PCrdReturn transactions.
  - Can take any value in the PrefetchTgt transaction.

B2.7.3.2 Device

Device attribute indicates if the memory type is either Device or Normal.

B2.7.3.2.1 Device memory type

Device memory type must be used for locations that exhibit side-effects. Use of Device memory type for locations that do not exhibit side-effects is permitted.

The requirements for a transaction to a Device type memory location are:

- A Read transaction must not read more data than requested.
- Prefetching from a Device memory location is not permitted.
- A read must get its data from the endpoint. A read must not be forwarded data from a write to the same address location that completed at an intermediate point.
- Combining requests to different locations into one request, or combining different requests to the same location into one request, is not permitted.
- Writes must not be merged.
- Writes to Device memory that obtain completion from an intermediate point must make the write data visible to the endpoint in a timely manner.

Accesses to Device memory must use the following types, exclusive variants are permitted:

- Read accesses to a Device memory location must use ReadNoSnp.
- Write accesses to a Device memory location must use WriteNoSnpPtl, WriteNoSnpFull, WriteNoSnpZero, or WriteNoSnpDef.
- CMO transactions are permitted to Device memory locations.
- Atomic transactions are permitted to Device memory locations.
- The PrefetchTgt transaction is not permitted to Device memory locations. The MemAttr field is inapplicable and can take any value in the PrefetchTgt transaction.

B2.7.3.2.2 Normal memory type

Normal memory type is appropriate for memory locations that do not exhibit side-effects.

Accesses to Normal memory do not have the same restrictions regarding prefetching or forwarding as Device type memory:

- A Read transaction that has EWA asserted can obtain read data from a Write transaction that has sent its completion from an intermediate point and is to the same address location.

- Writes can be merged.

Any Read, Dataless, Write, PrefetchTgt, or Atomic transaction type can be used to access a Normal memory location. The transaction type used is determined by the memory operation to be accomplished, and the Snoopable attributes.

B2.7.3.3 Cacheable

The Cacheable attribute indicates if a transaction must perform a cache lookup:

- When Cacheable is asserted, the transaction must perform a cache lookup.
- When Cacheable is deasserted, the transaction must access the final destination.

The Cacheable attribute value requirements are:

- Must be asserted for any:
  - Read transaction except ReadNoSnp and ReadNoSnpSep.
  - Dataless transaction except CleanShared, CleanSharedPersist*, CleanInvalid, CleanInvalidPoPA, and MakeInvalid.
  - Write transaction, except WriteNoSnpFull, WriteNoSnpPtl, and WriteNoSnpDef.
- Must not be asserted for any:
  - Device memory transaction
  - WriteNoSnpDef transaction
- Can take any value for:
  - ReadNoSnp or ReadNoSnpSep transaction to a Normal memory location.
  - WriteNoSnpFull and WriteNoSnpPtl transactions to a Normal memory location.
  - CleanShared, CleanSharedPersist*, CleanInvalid, CleanInvalidPoPA, and MakeInvalid.
  - An Atomic transaction.
- Is inapplicable and:
  - Must be set to 0 DVMOp and PCrdReturn.
  - Can take any value in the PrefetchTgt.

Note

In a transaction that can take any Cacheable value, the value is typically determined from the page table attributes.

B2.7.3.4 Allocate

The Allocate attribute is an allocation hint and indicates the recommended allocation policy for a transaction:

- If Allocate is asserted, it is recommended that the transaction is allocated into the cache for performance reasons. However, the cache is permitted to not allocate the transaction.
- If Allocate is deasserted, it is recommended that the transaction is not allocated into the cache for performance reasons. However, the cache is permitted to allocate the transaction.

The Allocate attribute value requirements are:

- Can be asserted for transactions that have the Cacheable attribute asserted, except for ReadOnceMakeInvalid.

- Must be asserted for the WriteEvictFull transaction.

Note

A Requester can convert a WriteEvictFull with the Allocate bit not asserted to an Evict transaction.

- Must not be asserted for:
  - Device memory transactions.
  - Normal Non-cacheable memory transactions.
- Is inapplicable and:
  - Must be set to 0 in DVMOp, PCrdReturn, and Evict transactions.
  - Can take any value in the PrefetchTgt transaction.

B2.7.3.5 Propagation of Attr

A request from the Home Node to Subordinate Node sent in response to a request to the Home Node must preserve the MemAttr bits EWA, Device, Cacheable, and Allocate. The only exception to this rule is when the downstream memory is known to be Normal, then the Device field value can be set to `0b0` to indicate Normal.

In a Combined Write transaction, when the write and CMO transactions are separated, the Write transaction inherits the MemAttr and SnpAttr values of the original combined request. The separated CMO transaction SnpAttr and Cacheable bits must be set to the most pervasive to affect all caches at RN-F nodes and caches downstream.

For a ReadNoSnp or WriteNoSnp generated within the interconnect due to a Prefetch from Home or an eviction from the System cache:

- MemAttr bits EWA, Cacheable, and Allocate must all be set to `0b1`.
- Device field value must be set to `0b0` to indicate Normal.
- SnpAttr field value must be set to `0b0` to indicate Non-snoopable.

B2.7.4 Transaction attribute combinations

Table B2.12 lists the legal combinations of MemAttr, SnpAttr, and Order field values and the equivalent ARM memory type. The Order field is described in B2.6 *Ordering*.

Table B2.12: Legal combinations of MemAttr, SnpAttr, and Order field values

MemAttr[3:0]						Order[1:0][a]		
[1]	[3]	[2]	[0]			[1]	[0]	ARM Memory type
Device	Allocate	Cacheable	EWA	SnpAttr	LikelyShared			
1	0	0	0	0	0	1	1	Device nRnE
	0	0	1	0	0	1	1	Device nRE
	0	0	1	0	0	0/1[a]	0	Device RE

Continued on next page

Table B2.12 – *Continued from previous page*

MemAttr[3:0]						Order[1:0][a]		
[1]	[3]	[2]	[0]			[1]	[0]	**ARM Memory type**
Device	**Allocate**	**Cacheable**	**EWA**	**SnpAttr**	**LikelyShared**			
	All other values[b]							Not valid
0	0	0	0	0	0	0/1[a]	0	Non-cacheable Non-bufferable[c]
	0	0	1	0	0	0/1[a]	0	Non-cacheable Bufferable
	0	1	1	0	0	0/1[a]	0	Non-snoopable WriteBack No-allocate
	1	1	1	0	0	0/1[a]	0	Non-snoopable WriteBack Allocate
	0	1	1	1	0/1[d]	0/1[a]	0	Snoopable WriteBack No-allocate
	1	1	1	1	0/1[d]	0/1[a]	0	Snoopable WriteBack Allocate
	All other values[b]							Not valid

[a] Order = `0b10` is permitted in ReadOnce*, WriteUnique, ReadNoSnp, WriteNoSnp, WriteNoSnpDef, and Atomic transactions only.

[b] Order = `0b01` is not used for the ordering of transactions. See B2.6.5 *Transaction ordering*.

[c] Non-cacheable Non-bufferable is an AXI memory type, not an ARM memory type.

[d] See B2.7.5 *Likely Shared*.

B2.7.4.1 Memory type

This section specifies the required behavior for each of the memory types shown in Table B2.12.

B2.7.4.1.1 Device nRnE

The required behavior for Device nRnE memory type is:

- The write response must be obtained from the final destination.
- Read data must be obtained from the final destination.
- A read must not fetch more data than is required.
- A read must not be prefetched.
- Writes must not be merged.
- A write must not write to a larger address range than the original transaction.
- All Read and Write transactions from the same source to the same endpoint must remain ordered.

B2.7.4.1.2 Device nRE

The required behavior for the Device nRE memory type is the same as for the Device nRnE memory type, except that the write response can be obtained from an intermediate point.

B2.7.4.1.3 Device RE

The required behavior for the Device RE memory type is same as for the Device nRE memory type, except:

- Read and Write transactions from the same source to the same endpoint need not remain ordered.
- Read and Write transactions from the same source to addresses that overlap must remain ordered.

B2.7.4.1.4 Normal Non-cacheable Non-bufferable

The required behavior for the Normal Non-cacheable Non-bufferable memory type is:

- The write response must be obtained from the final destination.
- Read data must be obtained from the final destination.
- Writes can be merged.
- Read and Write transactions from the same source to addresses that overlap must remain ordered.

B2.7.4.1.5 Normal Non-cacheable Bufferable

The required behavior for the Normal Non-cacheable Bufferable memory type is:

- The write response can be obtained from an intermediate point.
- Write transactions must be made visible at the final destination in a timely manner.

  Note

  There is no mechanism to determine when a Write transaction is visible at its final destination.

- Read data must be obtained either from:
  - The final destination.
  - A Write transaction that is progressing to its final destination.

    If read data is obtained from a Write transaction:

    * The data must be obtained from the most recent version of the write.
    * The data must not be cached to service a later read.
- Writes can be merged.
- Read and Write transactions from the same source to addresses that overlap must remain ordered.

Note

For a Normal Non-cacheable Bufferable read, data can be obtained from a Write transaction that is still progressing to its final destination. This is indistinguishable from the Read and Write transactions propagating to arrive at the final destination at the same time. Read data returned in this manner does not indicate that the Write transaction is visible at the final destination.

B2.7.4.1.6 Write-Back No-allocate

The required behavior for the Write-Back No-allocate memory type is:

- The Write response can be obtained from an intermediate point.
- Write transactions are not required to be made visible at the final destination.
- Read data can be obtained from an intermediate cached copy.
- Reads can be prefetched.
- Writes can be merged.
- A cache lookup is required for Read and Write transactions.

- Read and Write transactions from the same source to addresses that overlap must remain ordered.
- The No-allocate attribute is an allocation hint and only recommends to the memory system that the transaction is not allocated because of performance reasons. However, the allocation of the transaction is not prohibited.

B2.7.4.1.7 Write-Back Allocate

The required behavior for the Write-Back Allocate memory type is the same as for Write-Back No-allocate memory. However, in this case, the allocation hint is a recommendation to the memory system that, for performance reasons, the transaction is allocated.

B2.7.5 Likely Shared

The LikelyShared attribute is a cache allocation hint. When asserted this attribute indicates that the requested data is likely to be shared by other Request Nodes within the system. This acts as a hint to shared system level caches that the allocation of the cache line is recommended for performance reasons.

There is no required behavior associated with this transaction attribute.

The LikelyShared assertion requirements:

- Can be asserted in:
  - ReadClean
  - ReadNotSharedDirty
  - ReadShared
  - StashOnceUnique, StashOnceSepUnique
  - StashOnceShared, StashOnceSepShared
  - WriteUniquePtl
  - WriteUniqueFull
  - WriteUniqueZero
  - WriteUniquePtlStash
  - WriteUniqueFullStash
  - WriteBackFull
  - WriteCleanFull
  - WriteEvictFull
  - WriteEvictOrEvict
- Must not be asserted in any:
  - Combined Write transactions
  - Dataless or Atomic transactions
  - Other Read or Write transactions
- Is not applicable in:
  - DVMOp or PCrdReturn transaction, and must be set to 0.
  - PrefetchTgt transaction and can take any value.

B2.7.6 Snoop attribute

The SnpAttr field indicates if a transaction could require snooping.

See Table B13.19 for SnpAttr field encodings.

Table B2.13 shows the snoop attributes for the different transaction types. The following key is used:

Y Yes, permitted

- No, not permitted

n/a Not applicable

Table B2.13: Snoop attributes for different transaction types

Transaction	Non-snoopable	Snoopable
ReadNoSnp	Y	-
ReadNoSnpSep		
ReadOnce*	-	Y
ReadClean		
ReadShared		
ReadNotSharedDirty		
ReadUnique		
ReadPreferUnique		
MakeReadUnique		
CleanUnique	-	Y
MakeUnique		
StashOnce		
CleanShared	Y	Y
CleanSharedPersist*		
CleanInvalid		
CleanInvalidPoPA		
MakeInvalid		
Evict	-	Y
WriteNoSnp	Y	-
WriteNoSnpDef	Y	-
WriteBack	-	Y
WriteClean		
WriteEvictFull		
WriteEvictOrEvict		
WriteUnique	-	Y
Atomic transactions	Y	Y
DVMOp	n/a[a]	n/a[a]
PrefetchTgt	n/a[b]	n/a[b]

Continued on next page

Table B2.13 – *Continued from previous page*

Transaction	Non-snoopable	Snoopable

[a] Not applicable, SnpAttr bit is used as a Domain Identifier in DVM transactions, see B8.4.1 *DVM domain*.
[b] Not applicable, can take any value.

The SnpAttr field value in a CMO, in an Atomic, and in ReadNoSnp and ReadNoSnpSep from Home to Subordinate, must be set to 0, irrespective of the field value in the Request from the original Requester to Home. In Write and Combined Write transactions from Home to Subordinate, the bit position for the SnpAttr field is used for the DoDWT field. See B2.3.9.2 *Home to Subordinate Write transactions* and B2.3.9.4 *Home to Subordinate Combined Write and CMO transactions*.

Note

For transactions that can take more than one value of SnpAttr, the value is typically determined from page table attributes.

B2.7.7 Mismatched Memory attributes

Two different accesses are permitted to occur to the same location using mismatched MemAttr or SnpAttr values. The mismatched MemAttr or SnpAttr values can be expected or unexpected. It is required that the system does not deadlock in either instance and the transaction always makes forward progress.

To assist with the expected mismatched memory attributes, an interconnect:

- Must ensure a Clean line, either UC or SC, in a fully coherent cache is:
  - Not made visible to a Requester using a transaction with SnpAttr = 0 to access that location, if that location has been updated by another transaction with SnpAttr = 0 after the cache line was allocated into the fully coherent cache.
  - Not written to main memory.
  - Not used to overwrite a Dirty copy in a downstream cache.
- Is permitted, but not required, to make a Dirty copy of a cache line that was written by a Requester using a transaction with SnpAttr = 0 visible to a fully coherent Requester.
  Visibility of the Dirty line must not be removed from the Requester using a transaction with SnpAttr = 0 to access that location.
- Is permitted, but not required, to make a Dirty line, either UD or SD, in a fully coherent cache visible to another Requester using a transaction with SnpAttr = 0 to access that location.
  Once a Dirty line in a fully coherent cache is made visible to a Requester using a transaction with SnpAttr = 0 request to access that location, the cache line must not be removed from the visibility of that Requester.

Note

There are multiple implementation options that could be used to ensure that this removal of visibility does not take place. These include, but are not limited to:

- Writing back the dirty copy of the line to main memory.
- Keeping a copy of the line in the system cache, to be observed by requests that have SnpAttr = 0.
- Snooping fully coherent caches to complete requests that have SnpAttr = 0.

- Must ensure that a CMO must not be terminated on hitting a Unique copy of a line.
  The continuation of the CMO is required because the Unique copy could be stale and other dirty copies of the same location exist elsewhere in the cache hierarchy of the system.

B2.7.7.1 Reasons for mismatched attributes

Mismatched attributes could happen for various reasons:

- B2.7.7.1.1 *Upgrading of Non-shareable Cacheable accesses*
- B2.7.7.1.2 *Software protocol errors*

B2.7.7.1.1 Upgrading of Non-shareable Cacheable accesses

Fully coherent Requesters that have Nonshareable_Cache_Maint property set to True upgrade all accesses to locations that are marked in page tables as Non-shareable Cacheable to be Shareable Cacheable.

Fully coherent Requesters that have Nonshareable_Cache_Maint property set to False will use transactions with SnpAttr = 0 to access these locations. See B16.1.17 *Nonshareable_Cache_Maint* for more details.

With this upgrade:

- A fully coherent Requester uses transactions with SnpAttr = 1 when accessing Non-shareable Cacheable locations.
- All snoops received to locations that were marked Non-shareable Cacheable must behave in the same manner as if the location was originally marked as Shareable Cacheable in the page tables. Additionally, a CMO issued by any Requester in the system subsequently operates on copies that are cached by a fully coherent Requester or interconnect.
- Cache maintenance is enabled on locations marked as Non-shareable in the page tables to be performed by a PE that is different to the one that caused allocation into the cache. This is required to support features such as Realm Management Extensions, see Chapter B10 *Realm Management Extension*.

Leaving these locations as Non-shareable Cacheable is preferred to high bandwidth IO coherent Requesters, such as GPUs, who do not encounter any snoop filter look-up on all transactions.

When an IO Coherent Requester has been performing stores to the location using Non-snoopable transactions, such as WriteNoSnp, an upgraded version is of that line is possible in an RN-F or interconnect becomes stale.

The interconnect is responsible to ensure stale copies of the cache lines are not made visible to agents that must not observe them and ensure that the most up-to-date copies of the cache lines are not removed from the visibility of agents that must observe them.

B2.7.7.1.2 Software protocol errors

Memory accesses from different agents made with mismatched snoopability or cacheability attributes that are not expected can be considered as software protocol errors. A software protocol error can cause loss of coherency and result in corruption of data values. See B9.4.1 *Software-based error*.

A software protocol error for an access in one 4KB memory region must not cause data corruption in a different 4KB memory region.

For locations held in Normal memory, the use of appropriate software cache maintenance can be used to return memory locations to a defined state.

The use of mismatched memory attributes can result in an RN-F observing a Snoop transaction to the same address where the RN-F would use, or previously have used, Non-snoopable transactions to access. If an RN-F receives a Snoop to a location that is believed to be Non-snoopable, the RN-F must not respond with data and instead send SnpResp_I. When mismatched attributes are used, there is no defined relationship between the Snoop transaction and the transaction that the RN-F has issued.

B2.7.8 CopyAtHome attribute

CopyAtHome, CAH, is a cache line attribute that can be used by Home to optimize away redundant data transfers. It is useful in topologies where system caches are neither fully-inclusive or fully-exclusive, but which adapt their inclusion policy depending on the behavior of the system.

In CompData and DataSepResp response to the Requester, the CAH value indicates if the Home keeps a copy of the line. The CAH value is cached with the line at the Requester. If the line is updated locally, the cached CAH attribute must be reset.

When the Requester performs a CopyBack Write or Combined CopyBack Write for the line, the CAH value is sent along with the request. The Home can use the CAH value in the request to determine if it should check for a local copy of the line. If Home still retains a copy of the line, the CopyBack Write transaction is requested to complete without a CopyBackWriteData transfer.

B2.7.8.1 CAH usage at Home

In CompData or DataSepResp responses from Home, the CAH attribute is set to:

1 If the Home indicates a copy of the line is being kept. If the Home has provided a Requester with a Unique copy of the line, the copy at the Home is a hidden copy and is not observable to any agents in the system. If the Hidden copy is Clean, Home is permitted to remove the copy of the cache line at any time, except for during the window where a Comp has been issued and the associated CompAck is still outstanding. Home is not permitted to remove the hidden copy of the line if the line is Dirty.

0 If the Home intends to not keep a copy of the line nor supports the optimized CopyBack Write flow.

When the Home receives a later CopyBack Write request, it can inspect the CAH attribute to establish the transaction flow variation for use to complete the transaction. Table B2.14 shows the CAH attribute usage at Home for CopyBack Write transactions.

Table B2.14: CAH usage at Home for CopyBack Write transactions

Request CAH value	CopyBack Write transaction	Description
1	WriteBack, WriteClean, or WriteEvictFull	The Home is permitted to do one of the following: • Respond with CompDBIDResp to request the write data for the transaction. • Check if Home still has a copy of the line: **If Home has a copy of the line** — The Home is expected, but not required, to respond with Comp to request that the transaction is completed without a data transfer. If the Home does not send Comp, it must send CompDBIDResp, to request the write data for the transaction. **If Home does not have a copy of the line** — The Home must respond with CompDBIDResp to request the write data for the transaction.

Continued on next page

Table B2.14 – *Continued from previous page*

Request CAH value	CopyBack Write transaction	Description
	WriteEvictOrEvict	The Home is permitted to do one of the following: • Respond with CompDBIDResp to request the write data for the transaction. • Respond with Comp to request that the transaction is completed without a data transfer. • Check if Home still has a copy of the line: **If Home has a copy of the line**: The Home is expected, but not required, to respond with Comp to request that the transaction is completed without a data transfer. If the Home does not send Comp, it must send CompDBIDResp, to request the write data for the transaction. **If Home does not have a copy of the line**: The Home must respond with CompDBIDResp to request the write data for the transaction.
0 or Not checked	WriteBack, WriteClean, or WriteEvictFull	The Home must respond with CompDBIDResp to request the write data for the transaction.
	WriteEvictOrEvict	The Home is permitted to do one of the following: • Respond with CompDBIDResp to request the write data for the transaction. • Respond with Comp to request that the transaction is completed without a data transfer.

If Home issues a Comp instead of CompDBIDResp, a following CompAck response must be observed from the Requester to complete the CopyBack Write transaction. See B4.5.4 *Miscellaneous response* for details on how Home can interpret the possible CompAck responses.

B2.7.8.2 CAH usage at Request Node

The Requester or Snoopee handles the CAH attribute in the following way:

- If the CAH attribute has a value of 1 in a CompData or DataSepResp response, a Requester is permitted, but not required, to cache a value of 1 for CAH.
- If the CAH attribute has a value of 0 in a CompData or DataSepResp response, a Requester must not cache a value of 1 for CAH.
- A Requester is not required to cache an incoming CAH value. If the CAH value is not cached, the value must be assumed to be 0.
- When the line or MTE tags are updated, a cache must clear the CAH attribute to 0.
  - The CAH attribute can be cleared to 0 in a cache at any point.

> **Note**
>
> However, there is a loss of the ability to reduce the write data bandwidth for CopyBack Write transactions if the cached CAH attribute is cleared unnecessarily.

  - A Requester is not required to clear the CAH attribute to 0 when a WriteClean request is sent.

- When a Requester has a cache line with a CAH value of 0, a CopyBack transaction must not be sent with the CAH attribute in the request set to 1.
- When a Requester has a cache line with a CAH value of 1, it is expected, but not required, to send a CopyBack transaction with the CAH attribute as 1.
  - Once a CopyBack request has been sent with the CAH attribute set to 1, the Requester must not further modify the cache line until the transaction completes.
  - Once a WriteCleanFull request has been sent from a UD state with the CAH attribute set to 1, the Requester must ensure that any later updates to the line are not lost if the Home completes the transaction without requesting a data transfer.
- When supplying CompData in response to a forwarding snoop, a Snoopee is expected, but not required, to forward the CAH value to the Requester. If the Snoopee does not forward the CAH value, the value must be set to 0 in the CompData response.
- When supplying SnpRespData or SnpRespDataFwded to Home in response to a snoop, a Snoopee is expected, but not required, to send the CAH value. If the Snoopee does not send the CAH value, it then must be set to 0 in the data response.

For CopyBack Write transactions, the Resp field value in a CompAck response is applicable and must indicate the state of the cache at the point the CompAck response is sent. If a snoop request is received while the CopyBack Write is outstanding, the cache line state can differ from the cache line state when the original transaction was sent. See B4.7.3 *Write request transactions* for more information.

The cache line state information in the CompAck response must be used by the Home to determine whether any hidden copy of the line the Home has can be exposed. See Table B2.15 Home can take for each CompAck response.

Table B2.15: Permitted CompAck responses for CopyBack Write transactions

Response	Resp[2:0]	Cache line state when the response was sent	Action when a hidden copy of the cache line exists at Home	Notes
CompAck_I	0b000	Imprecise and must be ignored	Must not yet be exposed	Indicates a CopyBack request has been canceled
CompAck_UC	0b010	UC	Expose, unless a Unique copy exists elsewhere[a]. This is expected, but not required.	
CompAck_SC	0b001	SC	Expose. This is expected, but not required.	
CompAck_UD_PD	0b110	UD	Expose, unless a Unique copy exists elsewhere[a]. This is expected, but not required.	Responsibility for updating memory is passed to the Home

Continued on next page

Table B2.15 – *Continued from previous page*

Response	Resp[2:0]	Cache line state when the response was sent	Action when a hidden copy of the cache line exists at Home	Notes
CompAck_SD_PD	0b111	SD	Expose. This is expected, but not required.	Responsibility for updating memory is passed to the Home

[a] A Unique copy can still exist at the Requester when the CopyBack Write transaction was a WriteClean.

If Home has kept a hidden copy of a cache line and later determines that no other copies of the line exist:

- If the hidden copy is Clean, the Home is expected, but not required, to expose that copy.
- If the hidden copy is Dirty, the Home is required to expose that copy.

B2.8 Data transfer

A data payload is included in Read transactions, Write transactions, Combined Write transactions, Atomic transactions, and Snoop responses with data. This section defines the data alignment rules and the data bytes that are accessed for different combinations of address, transaction size, and memory type.

In this section, unless explicitly stated otherwise, a reference to a Write transaction includes both the individual Write transaction and the corresponding Combined Write transaction.

B2.8.1 Data size

The Size field in a packet is used, in combination with other fields, to determine the number of bytes transferred.

Table B2.16 shows the Size field value encodings. Snoop transactions do not include a Size field. All snoop data transfers are 64-byte.

Table B2.16: Size field value encodings

Size[2:0]	Bytes
0b000	1
0b001	2
0b010	4
0b011	8
0b100	16
0b101	32
0b110	64
0b111	Reserved

B2.8.2 Bytes access in memory

The MemAttr[1] bit field determines if the memory type is Device or Normal. See B2.7.3 *Memory Attributes*. The bytes that are accessed are determined by the memory type as follows:

Normal memory Transactions with a Normal memory type access the number of bytes defined by the Size field. Data access is from the `Aligned_Address`, that is, the transaction address rounded down to the nearest Size boundary and ends at the byte before the next Size boundary.
This is calculated as:
`Start_Address` = Addr field value.
`Number_Bytes` = 2^{Size} field value.
`INT(x)` = Rounded down integer value of x.
`Aligned_Address = (INT(Start_Address / Number_Bytes)) x Number_Bytes`.
The bytes accessed are from `(Aligned_Address)` to `(Aligned_Address + Number_Bytes) - 1`.

Device memory Transactions with a Device memory type access the number of bytes from the transaction address up to the byte before the next Size boundary.
The bytes accessed are from `(Start_Address)` to `(Aligned_Address + Number_Bytes) - 1`.

For Write transactions to Device locations, BE bits must only be asserted for the bytes that are accessed. See B2.8.3 *Byte Enables*.

B2.8.3 Byte Enables

Byte Enables (BE) are used alongside Write transactions and Snoop responses with Data.

In the following section, unless explicitly stated otherwise, a reference to a Write transaction includes both the individual Write transaction and the corresponding Combined Write transaction.

In WriteData and Snoop responses, a data BE value of 0 must set the associated data byte value to 0.

In CompData and DataSepResp messages, the BE bits are inapplicable and can take any value.

B2.8.3.1 Write transactions

For all Write transactions, BE bits that are not within the data window, specified by Addr and Size, must be deasserted.

An asserted BE in Write transactions indicates that the associated data byte is valid and must be updated in memory or cache. A deasserted BE indicates that the associated data byte is not valid and must not be updated in memory or cache.

A Requester must deassert all BE values in CopyBackWriteData_I and WriteDataCancel packets.

During data transfers in the following write transactions:

- All BE bits must be asserted, except when write data is a CopyBackWriteData_I or WriteDataCancel packet:
  - WriteNoSnpFull
  - WriteNoSnpDef
  - WriteBackFull
  - WriteCleanFull
  - WriteEvictFull
  - WriteUniqueFull
  - WriteUniqueFullStash
- Any combination of BE bits is permitted, including asserting all and asserting none:
  - WriteBackPtl
  - WriteUniquePtl
  - WriteUniquePtlStash

For a WriteNoSnpPtl transaction, the following rules apply:

- For a transaction to Normal memory, any combination of BE bits can be asserted during the data transfers. This includes asserting all and asserting none.
- For a transaction to Device memory, BE bits must only be asserted for bytes at or above the address specified in the transaction. Any combination of BE bits can be asserted that meets this requirement. This includes asserting all and asserting none.

B2.8.3.2 Atomic transactions

For Atomic transactions, all BE bits within the data window must be asserted.

For Atomic transactions, BE bits that are not within the data window, as specified below by Addr and Size, must be deasserted:

- If Addr is aligned to Size, the Data window is `[Addr:(Addr+Size-1)]`.
- If Addr is not aligned to Size, the Data window is `[(Addr-Size/2):(Addr+Size/2-1)]`.

B2.8.3.3 Snoop transactions

For Snoop responses with data that use the SnpRespData or SnpRespDataFwded opcode, all BE bits must be asserted.

For Snoop responses with data that use the SnpRespDataPtl opcode, any combination of BE bits can be asserted alongside the data transfers. This includes asserting all and asserting none.

B2.8.4 Data packetization

For each transaction that involves data, the data bytes can be transferred in multiple packets.

The number of packets required is determined by:

- Number of bytes
- Data bus width

The number of bytes transferred in each packet is determined by:

- Data bus width

The CHI protocol supports the following data bus widths:

- 128-bit
- 256-bit
- 512-bit

The *Data Identifier* (DataID) and *Critical Chunk Identifier* (CCID) fields are used to identify data packets within a transaction.

A transaction size of up to 16-byte is always contained in a single packet. The DataID can be calculated for each packet based on the data bus width:

- 128-bit: bits [5:4] of the effective memory address of any byte in the packet
- 256-bit: bit [5] of the effective memory address of any byte in the packet, concatenated with `0b0`
- 512-bit: always `0b00`

For different data bus widths, Table B2.17 shows the relationship between the DataID field and the bytes that are contained within the packet.

Table B2.17: DataID and the bytes within a packet for different data widths

DataID	Data Width		
	128-bit	256-bit	512-bit
`0b00`	Data[127:0]	Data[255:0]	Data[511:0]
`0b01`	Data[255:128]	Reserved	Reserved
`0b10`	Data[383:256]	Data[511:256]	Reserved
`0b11`	Data[511:384]	Reserved	Reserved

Within a data packet, all bytes are located at their natural byte positions. This is true even if fewer data bytes are transferred than the width of the data bus.

The number of data packets used for transactions to Device memory is independent of the address of the transaction. The number of data packets required is determined only by the Size field and the data bus width.

Note

For some transactions to Device memory, some data packets can be determined from the address at the start of the transaction to not contain valid data and are redundant. However, it is required that these data packets are transferred.

When a DAT message is split into multiple packets:

- The fields that must be consistent are:
  - TgtID
  - SrcID
  - TxnID
  - HomeNID
  - PBHA
  - Opcode
  - Resp
  - FwdState
  - DataPull
  - DataSource
  - DBID, when the field is applicable
  - CCID
  - TagOp
  - CAH
- The fields that are expected to be consistent, but can vary, are:
  - QoS
  - RespErr
  - DBID, when the field is inapplicable and can take any value
  - TraceTag
  - CBusy
- The fields that are expected to vary are:
  - Data
  - DataID
  - Tag
  - RSVDC
  - BE
  - DataCheck
  - Poison

B2.8.5 Size, Address, and Data alignment in Atomic transactions

This section describes the data size and alignment requirements for Atomic transactions. It contains the following sections:

- B2.8.5.1 *Size*
- B2.8.5.2 *Address and Data alignment*
- B2.8.5.3 *Endianness*

B2.8.5.1 Size

The Size field of the packet specifies the total data size of the Atomic transaction.

For the AtomicCompare transaction, the data size is the sum of the Compare and Swap data values.

Table B2.18 shows the permitted data sizes, and the relationship between inbound and outbound valid data size for each Atomic transaction type. The size of the data value returned in response to an AtomicCompare transaction is half the number of bytes specified in the Size field in the associated Request packet.

Table B2.18: Atomic transaction outbound and inbound data sizes

Atomic transaction	Outbound (bytes)	Inbound
AtomicStore	1, 2, 4, or 8	-
AtomicLoad	1, 2, 4, or 8	Same as outbound
AtomicSwap	1, 2, 4, or 8	Same as outbound
AtomicCompare	2, 4, 8, 16, or 32	Half size of outbound

B2.8.5.2 Address and Data alignment

In the AtomicStore, AtomicLoad, and AtomicSwap transactions:

- The byte address is aligned to the outbound data size.
- The position of data bytes in the NonCopyBackWriteData or any CompData packet matches the endianness of the operation, as specified in the Endian field of the request.
- The big-endian data is byte invariant.

The write data associated with an AtomicCompare transaction is provided in the same way as a transaction that is aligned to the outbound data size.

In the AtomicCompare transaction:

- The byte address must be aligned in the Data packet to the inbound data size, which is equivalent to half the outbound data size.

The two data values in an AtomicCompare transaction are placed in the data field in the following manner:

- The Compare and Swap data values are concatenated and the resulting data payload is aligned in the Data packet to the outbound data size.
- The Compare data is always at the addressed byte location.
- The Swap data is always in the remaining half of the valid data.

For any given Compare data address, the Swap data address can be determined by inverting bit[n] in the Compare data address where:

n = $\log_2$(Compare data size in bytes)

B2.8.5.2.1 Alignment example

Figure B2.36 shows the examples of data placement with different addresses and different Data sizes.

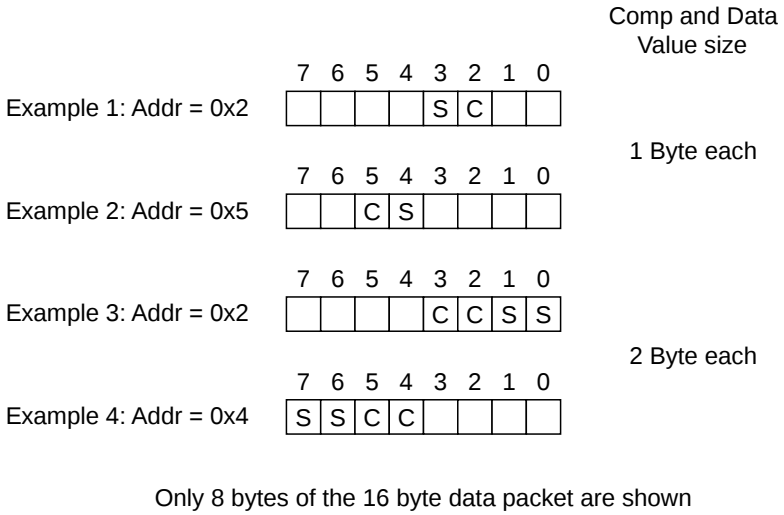

Figure B2.36: Data value packing for AtomicCompare transaction

In Figure B2.36, Example 1 shows the addressed byte location is `0x2` and the total size of data is 2 bytes. In this case, the Compare and Swap data must be placed in an address location aligned to a 2-byte boundary that includes the addressed location, that is, addresses `0x2` to `0x3`. Compare data is placed in location `0x2` and Swap data is placed in location `0x3`.

Note

The address of the Swap data can be determined by inverting bit[0] of the Compare data address. Bit[0] is inverted because the size of the Compare data and the size of the Swap data is 1 byte.

In Figure B2.36, Example 3 shows the addressed location is `0x2` and the total size of data is 4 bytes. In this case, the Compare and Swap data must be placed in an address location aligned to a 4-byte boundary that includes the addressed location, that is, addresses `0x0` to `0x3`. Compare data is placed in location `0x2` and Swap data is placed in location `0x0`.

Note

The address of the Swap data can be determined by inverting bit[1] of the Compare data address. Bit[1] is inverted because the size of the Compare data and the size of the Swap data is 2 bytes.

B2.8.5.3 Endianness

The data on which an atomic operation executes can be in either little-endian or big-endian format. For arithmetic operations, such as ADD, MAX, and MIN, the component performing the operation needs to know the format of the data.

The endian format of the data is defined by the Endian bit in the Atomic transaction Request packet. See B13.10.32 *Endian*.

B2.8.6 Critical Chunk Identifier

The CCID field is used to identify the data bytes that are the most critical in the transaction request.

The CCID field must match the value of Addr[5:4] of the original request. Transactions which contain multiple data packets must use the same CCID value for all data packets.

When read data or write data is reordered by the interconnect, the CCID field permits quick identification of the most critical bytes within a transaction by comparing the CCID value with the DataID value. When the two values match, the data bytes being transferred are the critical bytes.

The bits to match is dependent on the data bus width:

- For a data bus width of 128 bits, the CCID and DataID bits must match for the critical chunk.
- For a data bus width of 256 bits only the most significant CCID and DataID bits must match for the critical chunk.

B2.8.7 Critical Chunk First Wrap order

The Sender of Data is permitted, but not required, to send individual Data packets of a transaction in critical chunk first wrap order.

The interface property, CCF_Wrap_Order defines the capabilities of a Sender, and the guarantees provided by the Receiver:

- CCF_Wrap_Order at the Sender:

  True The Sender signals the Data packets can be sent in critical chunk first wrap order.

  False The Sender signals the Data packets cannot be sent in critical chunk first wrap order.

- CCF_Wrap_Order at the interconnect:

  True The interconnect signals the Data packets are guaranteed to maintain the order that transactions are received.

  False The interconnect signals the Data packets are not guaranteed to maintain the order that transactions are received.

- CCF_Wrap_Order at the Receiver that is not an interconnect:

  True The Receiver requires the Data packets to be received in critical chunk first wrap order.

  False The Receiver does not require the Data packets to be received in critical chunk first wrap order.

If some components in the system do not support sending Data packets in critical chunk first wrap order, the receiver of Data must not rely on Data being received in critical chunk first wrap order.

Note

At design time, the CCF_Wrap_Order parameter can help a component to identify if Data packets need to be sent in critical chunk first wrap order. For example, if the component is aware to be connected to an out-of-order interconnect, its Data packet path could be simplified by not returning the Data packets in critical chunk first wrap order.

If the interconnect has the CCF_Wrap_Order property set to True, a component interfacing to that interconnect can send Data packets in critical chunk first wrap order, if capable. The Receiver can subsequently make use of possible latency optimization, due to receiving the critical chunk first.

B2.8.8 Data Beat ordering

The reordering of data packets within a transaction is permitted when passing across an interconnect. However, the original source of data packets is permitted, but not required, to provide the packets in a critical chunk first, wrap order. See B2.8.7 *Critical Chunk First Wrap order*.

Note

Critical chunk first wrap order ensures that interfacing to protocols that do not support data reordering, such as AXI, can be done in the most efficient manner when an ordered interconnect is used.

Wrap order is defined as follows:

`Start_Address` = Addr

`Number_Bytes` = 2^{Size}

`INT(x)` = Rounded down integer value of x

`Aligned_Address = (INT(Start_Address / Number_Bytes)) x Number_Bytes`

`Lower_Wrap_Boundary = Aligned_Address`

`Upper_Wrap_Boundary = Aligned_Address + Number_Bytes - 1`

To maintain wrap order, the order must be:

1. The first data packet must correspond to the data bytes specified by the Start_Address of the transaction.
2. Subsequent packets must correspond to incrementing byte addresses up to the Upper_Wrap_Boundary.
3. Subsequent packets must correspond to the Lower_Wrap_Boundary.
4. Subsequent packets must correspond to incrementing byte addresses up to the Start_Address.

Note

Some of the steps to maintain wrap order could overlap and not be required if the required bytes are included in a previous step.

B2.8.9 Data transfer examples

This section gives a number of examples of the data transfer requirements.

In most of the examples, the size of the transaction is 64 bytes and the data bus width is 128 bits. This requires 4 data packets for each transaction.

In the following examples, the accompanying text highlights some interesting aspects. The examples are not intended to describe all aspects.

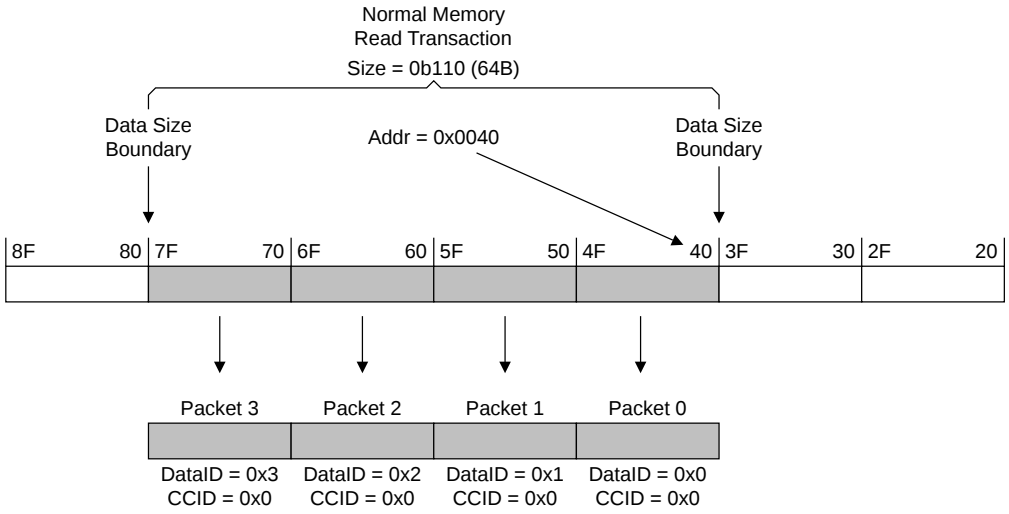

Figure B2.37: Normal memory 64-byte Read transaction from an aligned address

In Figure B2.37:

- The order of the data packets, as indicated by Packet 0, Packet 1, Packet 2, and Packet 3, is such that they follow wrap order.
- The DataID changes for each packet, while the CCID field remains constant.
- The packet containing the data bytes specified by the address of the transaction has the same value for the CCID and DataID fields.

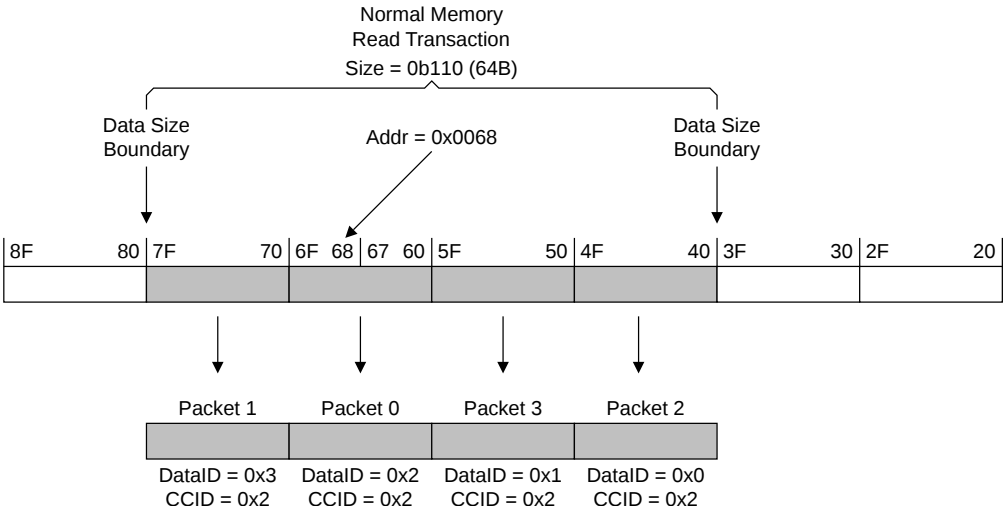

Figure B2.38: Normal memory 64-byte Read transaction from an unaligned address

In Figure B2.38:

- The order of the data packets, as indicated by Packet 0, Packet 1, Packet 2, and Packet 3, is such that they follow wrap order.
- The DataID changes for each packet, while the CCID field remains constant.
- The packet containing the data bytes specified by the address of the transaction has the same value for the CCID and DataID fields.

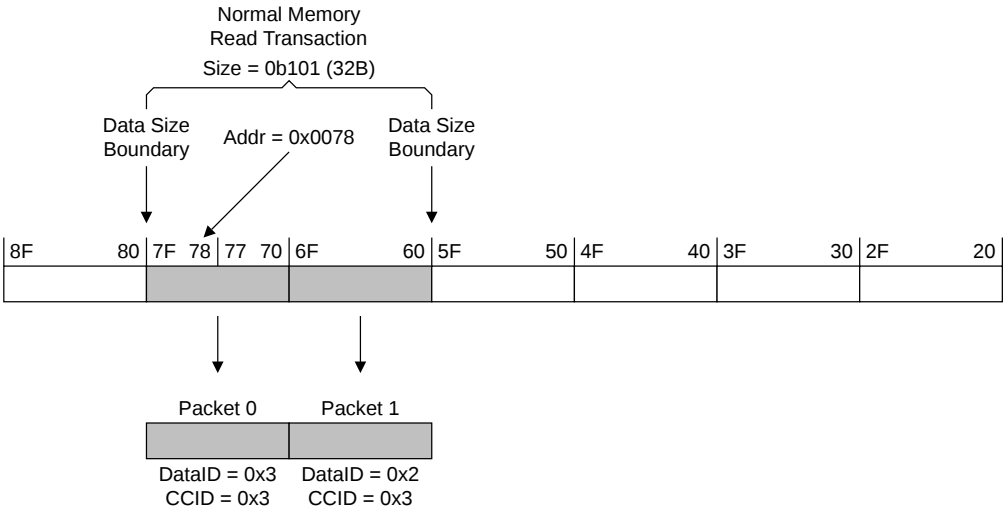

Figure B2.39: Normal memory 32-byte Read transaction from an unaligned address

In Figure B2.39:

- The size of the transaction is 32-byte and the data bus width is 128-bit, resulting in 2 data packets.
- The order of the data packets, as indicated by Packet 0 and Packet 1, is such that they follow wrap order.

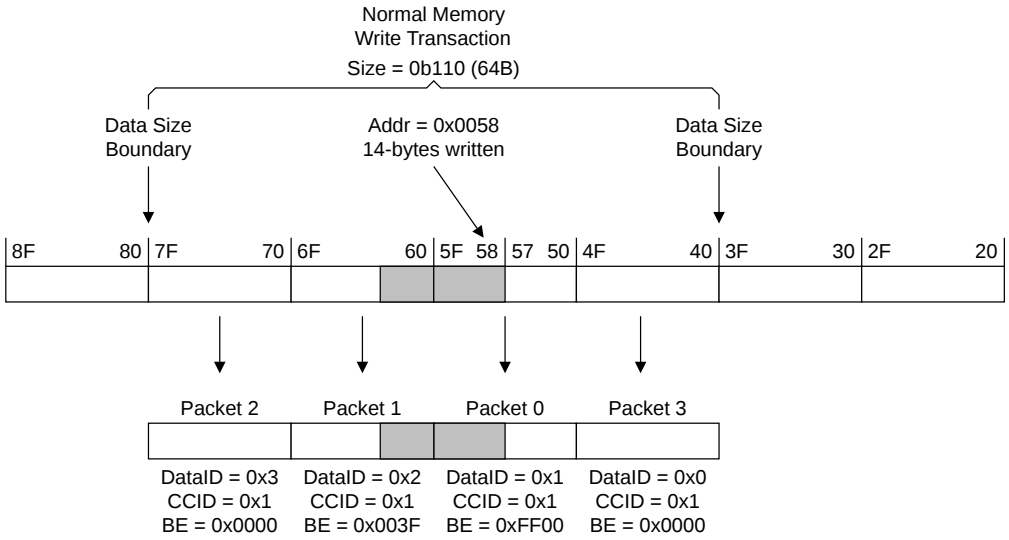

Figure B2.40: Normal memory 14-byte Read transaction from an unaligned address

In Figure B2.40:

- The order of the data packets, as indicated by Packet 0, Packet 1, Packet 2, and Packet 3, is such that they follow wrap order.
- The DataID changes for each packet, while the CCID field remains constant.
- The packet containing the data bytes specified by the address of the transaction has the same value for the CCID and DataID fields.
- Fourteen consecutive bytes in memory are written, as indicated by the BE bits. However, other combinations of BE bits are permitted. See B2.8.3 *Byte Enables*.

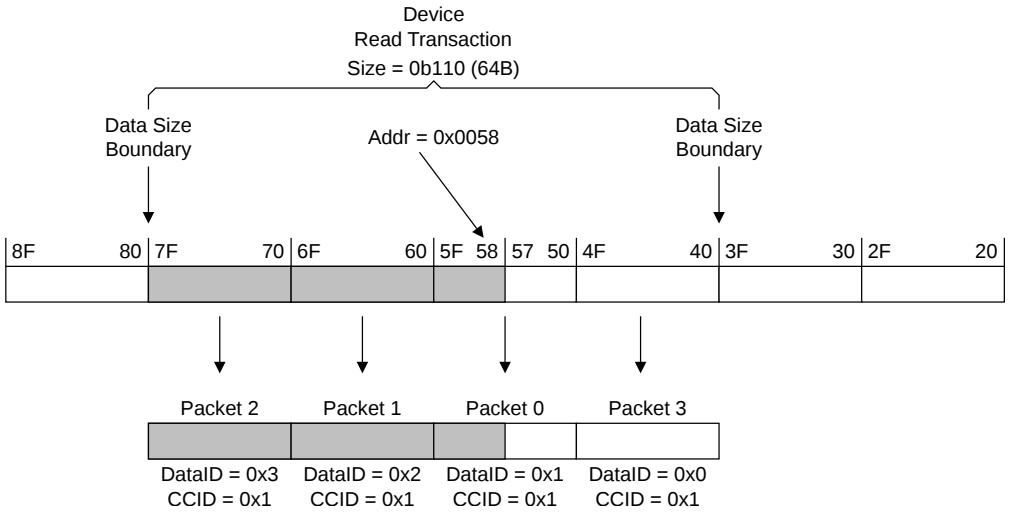

Figure B2.41: Device Read transaction from an unaligned address

In Figure B2.41:

- The shaded area indicates the valid bytes in the transaction. The valid bytes extend from the transaction address up to the next Size boundary.
- The transaction includes the transfer of a packet that contains no valid data.

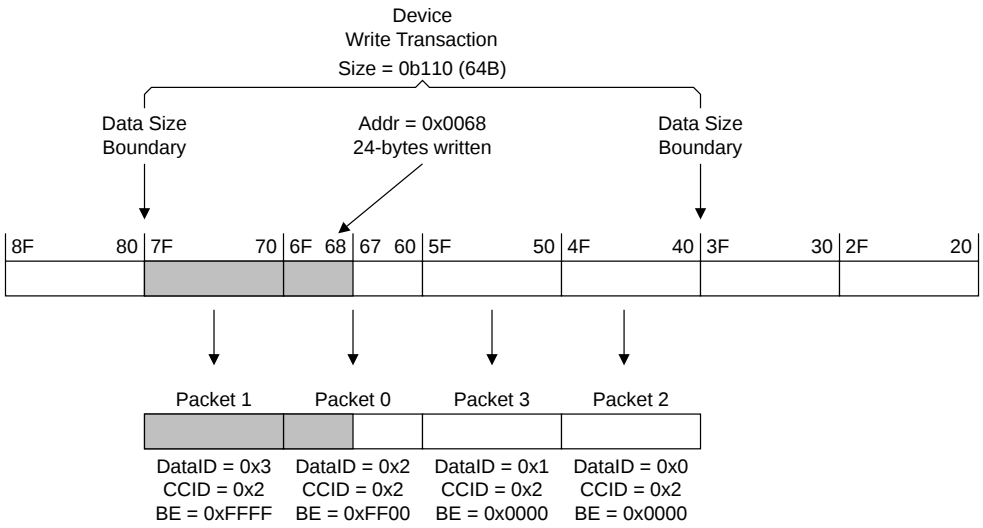

Figure B2.42: Device Write transaction to an unaligned address

In Figure B2.42:

- BE bits are only permitted to be asserted for the bytes from the transaction address up to the next Size boundary. It is not required that all BE bits meeting this criteria are asserted.
- BE bits for bytes below the start address must not be asserted.

B2.9 Request Retry

A Request Retry mechanism ensures that when a request reaches a Completer, the request can be either be accepted or be given a RetryAck response. Request Retry prevents blocking the REQ channel. There is no mechanism to Retry messages on the DAT, RSP, or SNP channels. For example, a DataPull response to a stash snoop cannot be retried.

Request Retry is not applicable to the PrefetchTgt transaction. The PrefetchTgt transaction cannot be retried because there is no response associated with this request.

A Requester is required to hold all the details of the request until a response is received indicating that the request has either been accepted or must be sent again at a later point in time. To meet this requirement, except for PrefetchTgt, the AllowRetry field must be asserted the first time a transaction is sent.

A Completer that is receiving requests is able to give a RetryAck response to a request that cannot be accepted. Typically, a Completer with limited resources and insufficient storage to hold the current request until some earlier transactions have completed will provide a RetryAck response.

The Completer is responsible for recording where the request came from RetryAck response is given, as determined by the SrcID of the request. The Completer is also responsible for determining and recording the type of *Protocol Credit* (P-Credit) required to process the request. The PCrdType field in the RetryAck encodes the type of P-Credit that is granted by the Completer. When required resources become available, at a later point in time, the Completer must subsequently send a P-Credit to the Requester, using a PCrdGrant response. The PCrdGrant response indicates to the Requester that the transaction can be retried.

Note

There is no explicit mechanism to request a credit. A transaction that is given a RetryAck response implicitly requests a credit.

Responses could be reordered so PCrdGrant is received by the Requester before the RetryAck response for the transaction is received. In this case, the Requester must record the received credit, including the credit type. The credit can appropriately be assigned by the Requester when the RetryAck response is received.

Note

Typically, the delay between a RetryAck and PCrdGrant response is much longer than any delay caused by interconnect reordering. PCrdGrant is expected to rarely be reordered with respect to RetryAck.

When the Requester receives a credit, the request can be resent in a second attempt with a credit allocation indication. The credit allocation indication is performed by deasserting the AllowRetry field. The second attempt to carry out the transaction is guaranteed to be accepted.

The transaction that is resent must have the same field values as the original request, except when the field is inapplicable or is one of the following:

- QoS
- TgtID, see B3.3.1 *TgtID determination for Request messages*.
- TxnID
- ReturnTxnID for:
  - ReadNoSnp and ReadNoSnpSep from the Home Node to the Subordinate Node, and a DMT flow is not being used.
  - WriteNoSnp or Combined Write from the Home Node to the Subordinate Node, and a DWT flow is not being used.
- SLCRepHint
- AllowRetry, which must be deasserted
- PCrdType, which must be set to the value in the Retry response for the original transaction.

- TraceTag
- RSVDC
- CAH

When the Requester receives a RetryAck response for a CopyBack Write request whose data is invalidated by a snoop, the Requester can either:

- Drop the Write request and return the received credit.
- Send the Write request with AllowRetry set to 0, knowing that the subsequent data response is CopyBackWriteData_I.

There is no fixed relationship between credits and particular transactions. If a Requester has received multiple RetryAck responses for different transactions and then receives a credit, there is no fixed credit allocation. The Requester is free to choose the most appropriate transaction from the list of transactions that receives a RetryAck response with that particular Protocol Credit Type.

The Retry mechanism supports up to 16 different credit types. This lets the Completer use different credit types for different resources. For example, a Completer could use one credit type for the resources associated with Read transactions, and another credit type for Write transactions. Using different credit types gives the Completer the ability to efficiently manage its resources by controlling which of the retried requests can be sent again.

The transaction must only be retried by the Requester when a PCrdGrant is received with the correct PCrdType.

Note

If a Completer is only using one credit type, it is recommended that the PCrdType value of `0b0000` is used. See B2.9.2.2 *PCRdType*.

A Completer must be able to record all the RetryAck responses to ensure credits are correctly distributed. If the Completer is using more than one credit type, the RetryAck responses that have been given for each credit type must be recorded.

A Requester must limit the number of issued transactions so that the Completer is never required to track more than 1024 transactions that require a PCrdGrant response. This is achieved by limiting the maximum number of outstanding transactions to 1024 for each Requester.

A transaction is outstanding from the cycle that the request is first issued until either:

- The transaction is fully completed, as determined by the return of all the following responses that are expected for the transaction:
  - ReadReceipt
  - CompData
  - RespSepData
  - DataSepResp
  - DBIDResp*
  - Comp, CompCMO, CompPersist, and CompStashDone
  - CompDBIDResp
- RetryAck and PCrdGrant is received and either the request is:
  - Retried using a credit of the appropriate PCrdType and subsequently is fully completed as determined by the return of all responses.
  - Canceled and returns the received credit using the PCrdReturn message.

A Requester can reuse the TxnID value used by a request either:

- As soon as a RetryAck response is received for that request.
- As soon as all the required responses are received for that request if the received responses are Non-RetryAck responses.

Each transaction request includes a QoS value which can be used by the Completer to influence the allocation of credits as resources become available. See B11.1 *Quality of Service (QoS) mechanism* for further details.

B2.9.1 Credit Return

It is possible for a Requester to be given more credits than required.

This specification does not define when this can occur, but two typical scenarios are:

- A transaction is canceled between the first attempt and the point at which the request can be resent with P-Credit.
- A transaction is requested multiple times with increasing QoS values. However, only a single completion of the transaction is required.

Note

If a Requester makes a second request before the first request has received a RetryAck response, both transactions must be acceptable to occur. However, as an example, this behavior would typically not be acceptable for accesses to a peripheral device.

A Requester returns a credit by the use of the PCrdReturn transaction. This is effectively a No Operation transaction that uses the credit that is not required. This transaction is used to inform the Completer that the allocated resources are no longer required for the given PCrdType.

Any credits that are not required must be returned in a timely manner.

Note

Any unused pre-allocated credit must be returned to avoid components holding on to credits in expectation of using them later. Such behavior is likely to result in an inefficient use of resources and to make analysis of the system performance difficult.

B2.9.2 Transaction Retry mechanism

The following sections describe the Request transaction fields used by the Retry mechanism.

The transaction retry mechanism is not applicable to the PrefetchTgt transaction.

B2.9.2.1 AllowRetry

The AllowRetry field indicates if the Request transaction can be given a RetryAck response. See Table B13.25 for the AllowRetry value encodings. The AllowRetry field must be asserted the first time a transaction is sent.

The AllowRetry field must be deasserted when either:

- The transaction is using a pre-allocated P-Credit.
- The transaction is PrefetchTgt.

B2.9.2.2 PCRdType

The PCrdType field indicates the credit type associated with the request and is determined as follows:

- For a Request transaction:
  - If the AllowRetry field is asserted, the PCrdType field must be set to `0b0000`.

  - If the AllowRetry field is deasserted, the PCrdType field must be set to the value that was returned in the RetryAck response from the Completer when the transaction was first attempted.

- A PCrdReturn transaction must have the credit type set to the value of the credit type that is being returned. See B13.10.39 *Protocol Credit Type, PCrdType* for the PCrdType value encodings.
- For destinations that have a single credit class, or do not implement credit type classification, it is recommended that the PCrdType field is set to `0b0000`.

Note

The value a Completer assigns to PCrdType is IMPLEMENTATION DEFINED.

The Completer must implement a starvation prevention mechanism to ensure that all transactions, irrespective of QoS value or credit type required, eventually make forward progress, even if over a significantly long time period. This is done by ensuring that credits are eventually given to every transaction that has received a RetryAck response. See B11.1 *Quality of Service (QoS) mechanism* for more details on the distribution of credits for the purposes of QoS.

B2.9.3 Transaction Retry flow

Figure B2.43 shows a typical Transaction Retry flow.

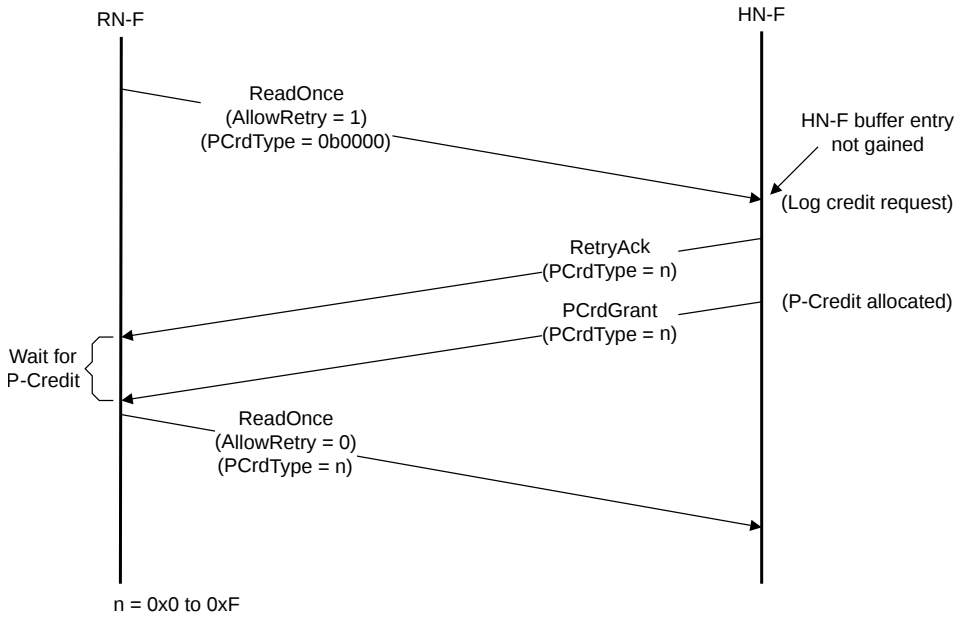

Figure B2.43: Transaction Retry flow

The steps that Figure B2.43 shows are:

1. The RN-F sends a ReadOnce request to the HN-F.
   - This is done without a credit, so AllowRetry is asserted. This means PCrdType must be set to `0b0000`.
2. The HN-F receives the request and sends a RetryAck response because the request is not able to gain buffer entry at the HN-F.
   - The request is logged and a PCrdType is determined at the HN-F.

3. The HN-F sends a P-Credit, using the PCrdGrant response, when a resource has been allocated for the transaction.

- The PCrdGrant includes the PCrdType allocated for the original request.

4. The RN-F re-sends the ReadOnce transaction with AllowRetry deasserted.

- The request uses the P-Credit and sets the PCrdType field to the value allocated for the original request.

It is permitted, but not expected, for a Completer to send a PCrdGrant before the associated RetryAck response is sent.

Note

The Requester could receive PCrdGrant before RetryAck.

The transaction must not be resent until both a RetryAck response and an appropriate P-Credit is received for the transaction.

Chapter B3
Network Layer

This chapter describes the network layer that is responsible for determining the node ID of a destination node. It contains the following sections:

B3.1 System Address Map, SAM

Requesters must have a *System Address Map* (SAM) to determine the TgtID of a request. Requesters could be Request Nodes or Home Nodes. The scope of the SAM could be as simple as providing a fixed node ID value to all the outgoing requests.

The exact format and structure of the SAM is IMPLEMENTATION DEFINED and is outside the scope of this specification.

The SAM must provide a complete decode of the entire address space. It is recommended that any address that does not correspond to a physical component is sent to an agent that can provide an appropriate error response.

B3.2 Node ID

Each component connected to a Port on the interconnect is assigned a node ID that is used to identify the source and destination of packets communicated over the interconnect. A Port can be assigned multiple node IDs. A node ID value can be assigned only to a single Port.

A variable NodeID field width of 7 bits to 11 bits is supported.

The width can be configured to any fixed value within this range for a given implementation and this value must be consistent across all NodeID fields.

Defining and assigning a node ID for each node in a system is IMPLEMENTATION DEFINED and is outside the scope of this specification.

B3.3 TgtID determination

This section describes how the TgtID is determined for the different message types. It contains the following sections:

- B3.3.1 *TgtID determination for Request messages*
- B3.3.2 *TgtID determination for Response messages*
- B3.3.3 *TgtID determination for snoop request messages*

B3.3.1 TgtID determination for Request messages

For mapping of TgtID in requests from the Request Node, it is required that the SAM logic is present in the Request Node or in the interconnect. In the case of the interconnect, the TgtID could be remapped in the Request packet provided by the Request Node.

The TgtID of a Request message is determined in the following manner using the SAM logic.

Except for PCrdReturn:

- If the request does not use a pre-allocated credit, the TgtID is determined by:
  - Opcode for DVMOp.
  - Address to node ID mapping for all other requests.
    PrefetchTgt uses a different Address to node ID mapper than other requests. Two requests from a Request Node to the same Address, where one is a PrefetchTgt, target different nodes. PrefetchTgt always targets a Subordinate Node. All other requests from a Request Node that use an Address to node ID mapper target a Home Node.
- If the request uses pre-allocated credit, the TgtID of the Request must be obtained from either the SrcID of the RetryAck, provided as a response to the original Request message, or the TgtID of the original request.

For PCrdReturn:

- The TgtID provided by the Request Node must match the SrcID included in the prior PCrdGrant which provided the credit being returned.

A Request Node must expect the interconnect to remap the TgtID of a request.

For transactions from a Request Node, except for PrefetchTgt which is targeted to an SN-F, it is expected a Snoopable transaction to be targeted to HN-F and a Non-snoopable transaction to target HN-I or HN-F. It is legal for a Snoopable transaction to be targeted at an HN-I, for example, due to a software programming error. In this case, the HN-I is required to respond to the transaction in a protocol-compliant manner, but coherency is not guaranteed.

A Home Node could also use address map logic to determine the target Subordinate Node ID for each request.

B3.3.2 TgtID determination for Response messages

Response packets are issued as a result of a received message. The TgtID in Response packets must match either the SrcID, HomeNID, ReturnNID, or FwdNID in the received message that resulted in the response being sent.

Table B3.1 shows the source of the Response packet TgtID for each Response message type and the field in the received message that determines the TgtID.

Table B3.1: Source of Response packet TgtID

Response message	TgtID obtained from		
	Home Node	Subordinate Node	Request Node
RetryAck	Request.SrcID	Request.SrcID	-
PCrdGrant	Request.SrcID	Request.SrcID	-
ReadReceipt	Request.SrcID	Request.SrcID	-
RespSepData	Request.SrcID or SnpResp*.SrcID[a]	-	-
Comp	Request.SrcID	Request.SrcID	-
CompCMO	Request.SrcID	Request.SrcID	-
DataSepResp	Request.SrcID or SnpResp*.SrcID[a]	Request.ReturnNID	-
CompData	Request.SrcID or SnpResp*.SrcID[a]	Request.ReturnNID	Snoop.FwdNID
CompAck (reads)	-	-	Comp.SrcID or RespSepData.SrcID or CompData.HomeNID
CompAck (writes)	-	-	Comp.SrcID or DBIDResp*.SrcID or CompDBIDResp.SrcID
CompDBIDResp	Request.SrcID	Request.SrcID[b]	
DBIDResp	Request.SrcID	If Request.DoDWT == 0 then: Request.SrcID or If Request.DoDWT == 1 then: Request.ReturnNID	-
DBIDRespOrd	Request.SrcID	-	-
WriteData	-	-	DBIDResp*.SrcID or CompDBIDResp.SrcID
NonCopyBackWriteDataCompAck	-	-	Comp.SrcID or DBIDResp*.SrcID or CompDBIDResp.SrcID
Persist	Request.SrcID	Request.ReturnNID	-
CompPersist	Request.SrcID	Request.SrcID[b]	-
StashDone	Request.SrcID	-	-
CompStashDone	Request.SrcID	-	-
TagMatch	Request.SrcID	Request.ReturnNID	-
SnpResp*[c]	-	-	Snoop.SrcID

[a] For Data Pull requests where Snoop response can be SnpResp or SnpRespData or SnpRespDataPtl.

[b] This response is only permitted when Request.SrcID = Request.ReturnNID, including when Request.DoDWT = 1.

[c] SnpResp, SnpRespData, SnpRespDataPtl, SnpRespFwded, and SnpRespDataFwded.

B3.3.3 TgtID determination for snoop request messages

A snoop request does not include a TgtID. The protocol does not define an architectural mechanism to address and send a Snoop request to a target. It is expected that the mechanism is IMPLEMENTATION DEFINED and therefore outside the scope of this specification.

B3.4 Network layer flow examples

This section shows transaction flows at the network layer. It contains the following sections:

- B3.4.1 *Simple flow*
- B3.4.2 *Flow with interconnect-based SAM*
- B3.4.3 *Flow with interconnect-based SAM and Retry request*

The following figures contain the term ‘Dec’ that represents the Decode.

B3.4.1 Simple flow

Figure B3.1 is an example of a simple transaction flow and shows how the TgtID is determined for the requests and responses.

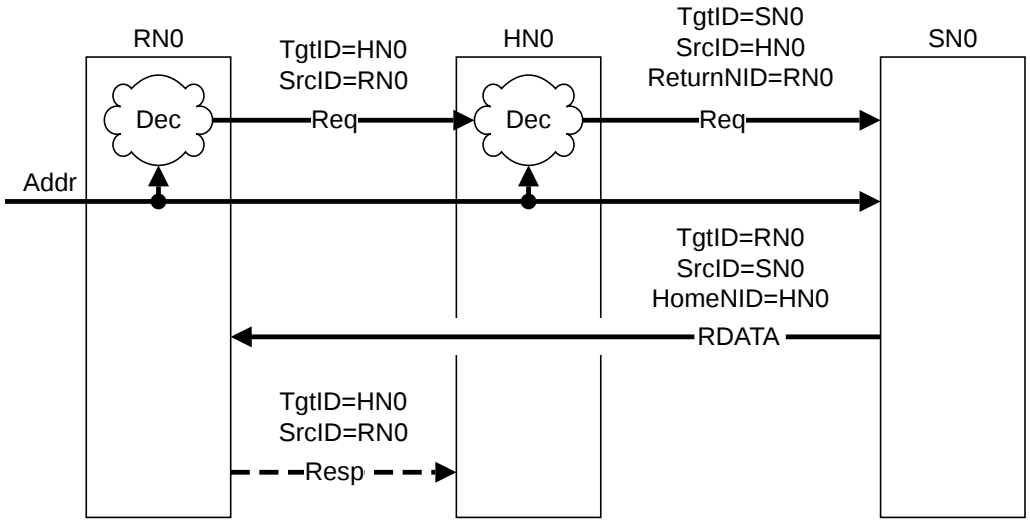

Figure B3.1: TgtID assignment without remapping

The steps for TgtID assignment without remapping in Figure B3.1 are:

1. RN0 sends a request with TgtID of HN0 using the SAM internal to RN0.
   - The interconnect does not remap the node ID.
2. HN0 looks up an internal SAM to determine the target Subordinate Node.
3. SN0 receives the request and sends a data response.
   - The data response packet has the TgtID derived from the requests ReturnNID.
4. RN0 receives the data response from SN0.
5. RN0 sends, if necessary, a CompAck response with TgtID of HN0 derived from the HomeNID in the data response packet to complete the transaction.

B3.4.2 Flow with interconnect-based SAM

Figure B3.2 shows a case where remapping of the TgtID occurs in the interconnect.

Note

Only the TgtID of the request from the Request Node is remapped. The TgtID in all other packets in the transaction flow is determined in a similar manner to B3.4.1 *Simple flow*.

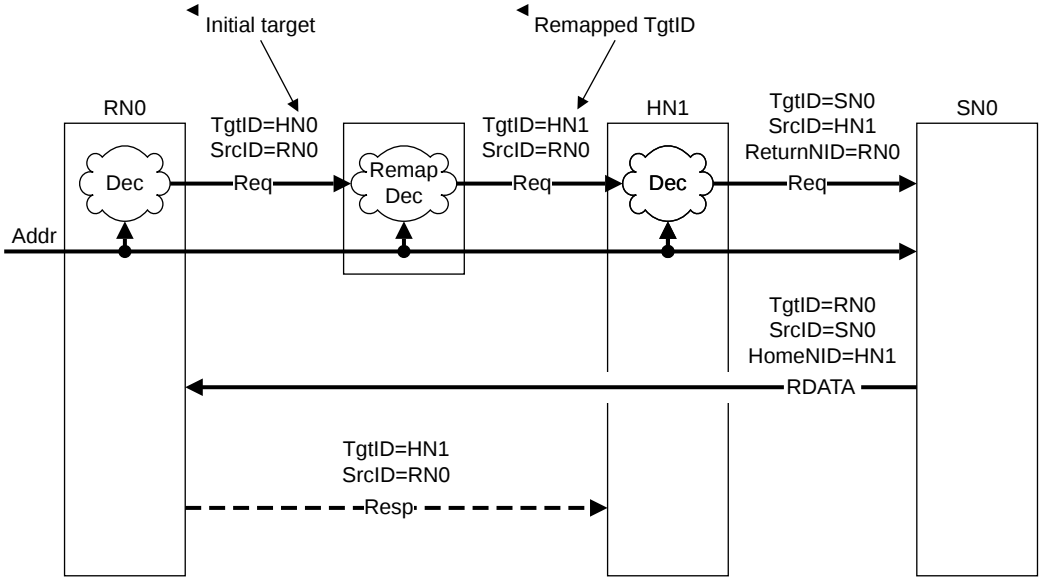

Figure B3.2: TgtID assignment with remapping logic

The steps for TgtID assignment with remapping logic in Figure B3.2 are as follows:

1. RN0 sends a Request with TgtID of HN0 using the SAM internal to RN0.
2. The interconnect remaps the request TgtID from HN0 to HN1. The SrcID remains set to the original requester, RN0.
3. HN1 looks up an internal SAM to determine the target Subordinate Node. The ReturnNID is set to match the original Requester, RN0.
4. SN0 receives the request and sends a data response.
   - The data response packet has the TgtID derived from the requests ReturnNID, with HomeNID set to match the remapped Home Node.
5. RN0 receives the data response from SN0.
6. If necessary, the RN0 sends a CompAck response with TgtID of HN1 derived from the HomeNID in the data response packet to complete the transaction.

B3.4.3 Flow with interconnect-based SAM and Retry request

Figure B3.3 shows a case of a request getting retried.

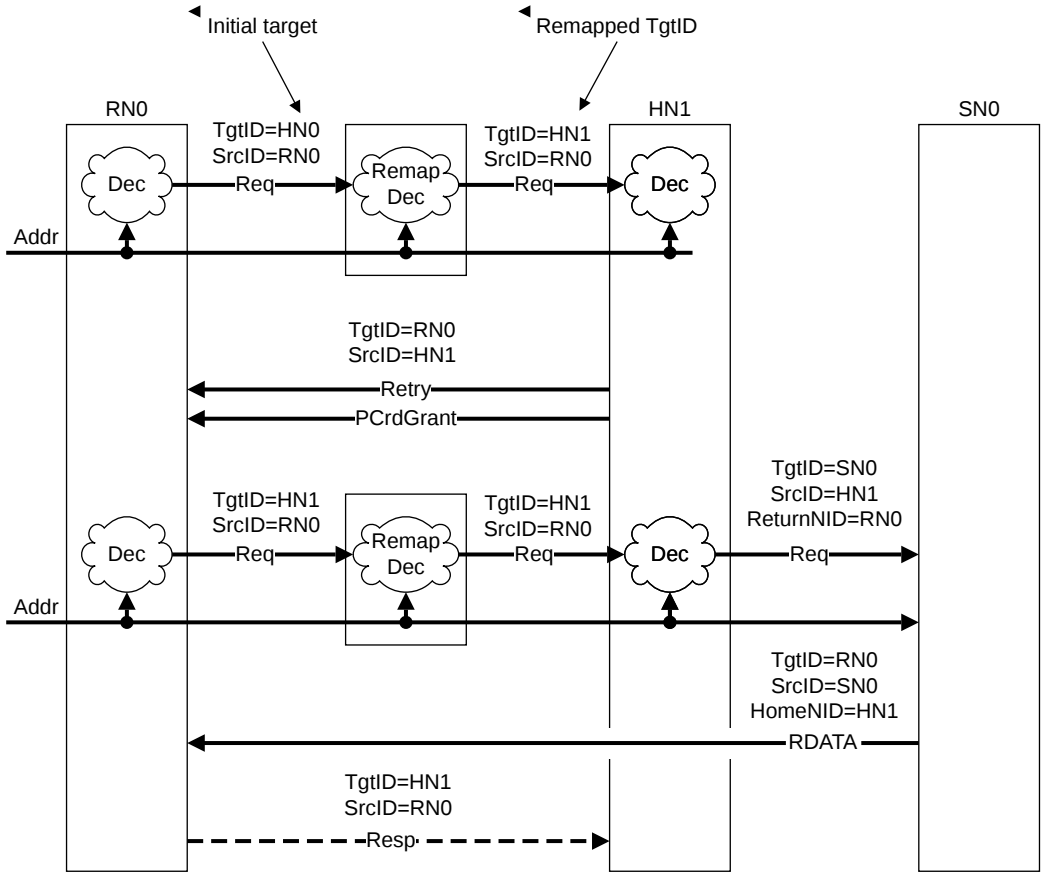

Figure B3.3: Remapping of TgtID and retried request

The steps to remap TgtID and retry the request in Figure B3.3 are as follows:

1. The interconnect remaps the TgtID provided by RN0 to HN1.
2. The request receives a RetryAck response.
   - The RetryAck and PCrdGrant responses get the TgtID information from the SrcID in the received request.
3. RN0 resends the request once both RetryAck and PCrdGrant responses are received.
   - The TgtID in the retried request is the same as the SrcID in the received RetryAck or the TgtID in the original request. The TgtID must pass through the remapping logic again.
4. The packets in the rest of the transaction flow get the TgtID in a similar manner to B3.4.2 *Flow with interconnect-based SAM*.

Chapter B4
Coherence Protocol

This chapter describes the coherence protocol and contains the following sections:

B4.1 Cache line states

The action required when a protocol node accesses a cache line is determined by the cache line state. The protocol defines the following cache line states:

I Invalid:
The cache line is not present in the cache.

UC Unique Clean:

- The cache line is present only in this cache.
- The cache line has not been modified with respect to memory.
- The cache line can be modified without notifying other caches.
- In response to a snoop that requests data, the cache line is permitted, but not required, to be:
  - Returned to Home when requested.
  - Forwarded directly to the Requester when instructed by the snoop.

UCE Unique Clean Empty:

- The cache line is present only in this cache.
- The cache line is in a unique state but none of the data bytes are valid.
- The cache line can be modified without notifying other caches.
- In response to a snoop that requests data, the cache line must not be:
  - Returned to Home even when requested.
  - Forwarded directly to the Requester even when instructed by the snoop.

UD Unique Dirty:

- The cache line is present only in this cache.
- The cache line has been modified with respect to memory.
- The cache line must be written back to next level cache or memory on eviction.
- The cache line can be modified without notifying other caches.
- In response to a snoop that requests data, the cache line:
  - Must be returned to Home when requested.
  - Is expected to be forwarded directly to the Requester when instructed by the snoop.

UDP Unique Dirty Partial:

- The cache line is present only in this cache.
- The cache line is unique. The cache line could have some bytes valid, where some includes none or all bytes.
- The cache line has been modified with respect to memory.
- When the cache line is evicted, data from next level cache or memory must be merged with the evicted cache line to form the complete valid cache line.
- The cache line can be modified without notifying other caches.
- In response to a snoop that requests data, the cache line must:
  - Be returned to Home.

- Not be forwarded directly to the Requester even when instructed by the snoop.

SC Shared Clean:

- Other caches could have a shared copy of the cache line.
- The cache line could have been modified with respect to memory.
- The cache is not responsible to write the cache line back to memory on eviction.
- The cache line cannot be modified without invalidating any shared copies and obtaining unique ownership of the cache line.
- In response to a snoop that requests data, the cache line:
  - Is required to not return data if RetToSrc bit is not set.
  - Is expected to return data if RetToSrc bit is set.
  - Is expected to be forwarded directly to the Requester when instructed by the snoop

SD Shared Dirty:

- Other caches could have a shared copy of the cache line.
- The cache line has been modified with respect to memory.
- The cache line must be written back to next level cache or memory on eviction.
- The cache line cannot be modified without invalidating any shared copies and obtaining unique ownership of the cache line.
- In response to a snoop that requests data, the cache line:
  - Must be returned to Home when requested.
  - Is expected to be forwarded directly to the Requester when instructed by the snoop.

A cache is permitted to implement a subset of these states.

B4.1.1 Empty cache line ownership

An empty cache line is a cache line that is held in a Unique state to prevent other copies of the cache line existing. None of the data bytes are valid in an empty cache line. This cache line state is UCE or UDP.

The following are examples of when empty cache line ownership can occur:

- A Requester can deliberately obtain an empty cache line before starting a write, to save system bandwidth. A Requester that expects to write to a cache line can obtain an empty cache line with permission to store, instead of obtaining a valid copy of the cache line.
- A Requester can transition into an empty state if the Requester has a copy of the cache line when permission to store is requested. That copy of the cache line is invalidated before the Requester obtains permission to store. At the completion of the request, this results in the Requester having an empty cache line with permission to store.

B4.1.2 Ownership of cache line with partial Dirty data

Once ownership of a cache line without data is obtained, the Requester is permitted, but not required, to store to the cache line. If the Requester modifies part of the cache line, the cache line remains partially Unique Dirty. This cache line state is UDP.

B4.2 Request types

Protocol requests are categorized as follows:

- For Read request transactions, a data response is provided to the Requester.
- For Dataless request transactions, no data response is provided to the Requester.
- For Write request transactions, data is moved from the Requester.
- For Combined Write request transactions, data is moved from the Requester and a cache maintenance operation is performed.
- For Atomic request transactions, data is moved from the Requester and a data response is provided to the Requester in some request types.
- For Stash request transactions, data can be moved within a system to improve performance.
- Other request transactions:
  - Do not involve any data movement in the system.
  - Can be used to assist with DVM maintenance.
  - Can be used to warm the memory controller for a following read request.

The following subsections enumerate the resulting transactions and their characteristics. See Chapter B12 *Memory Tagging* for a description of the Memory Tagging mechanism. See Table B12.3 for information on the permitted MTE TagOp values for each request. See Chapter C2 *Communicating Nodes* for information on Request communicating nodes.

Note

It is legal for any transaction that is expected to target an HN-F, but not an HN-I, to target an HN-I. This can occur for an incorrect assignment of memory type for a transaction. It is required that the HN-I responds to such a transaction in a protocol-compliant manner.

B4.2.1 Read transactions

Read transactions have the following common characteristics:

- Data must be included in the completion response to the Requester, except for MakeReadUnique request. For MakeReadUnique, a data response is optional.
- When Data response is provided, the transferred data can be from the Home Node, another Request Node, or a Subordinate Node.
- In Allocating Read transactions, received data, if cached, must be cached in a system coherent manner.
- In Non-allocating Read transactions, that is, ReadNoSnp and ReadOnce*, received data is not expected to be cached. If cached, data is not cached in a system coherent manner.
- The request can result in a cache state change at the Requester. See Table B4.4 for expected and permitted initial cache and Table B4.5 for final cache state at the Requester for each of the Read transactions.
- The request can result in a cache state change at other Request Nodes in the system. See Table B4.6 for expected and permitted cache states at the peer Request Nodes for each of the Read transactions.

See Chapter B6 *Exclusive accesses* for details on Exclusive attribute in read transactions.

ReadNoSnp
Read request by a Request Node to a Non-snoopable address region. Alternatively, from a Home Node to any address region to obtain a copy of the addressed data. See Table B2.6 for use DMT, Order field, and ExpCompAck values.

ReadNoSnpSep
Read request from a Home Node to Subordinate Node, requesting the Completer to send only a data response. Used when separate completion and data responses are used to complete the read transaction.

ReadOnce
Read request to a Snoopable address region to obtain a snapshot of the coherent data.

ReadOnceCleanInvalid
Read request to a Snoopable address region to obtain a snapshot of the coherent data. It is recommended, but not required, that other cached copies of the cache line are cleaned and invalidated. If a Dirty copy is invalidated, it must be written back to memory.

> **Note**
>
> ReadOnceCleanInvalid is used instead of ReadOnce or ReadOnceMakeInvalid where the application determines that the data is still Valid, but is not used in the near future.
>
> Use of ReadOnceCleanInvalid by an application improves cache efficiency by reducing cache pollution.

The following should be considered when using ReadOnceCleanInvalid:

- The clean and invalidation in the ReadOnceCleanInvalid transaction is a hint. Completion of the transaction does not guarantee the cleaning or removal of all cached copies, therefore it cannot be used as a replacement for a CMO.
- Use of the transaction can cause the deallocation of a cache line and therefore caution is needed if the transaction could target the same cache line that other agents in the system are using for Exclusive accesses.

ReadOnceMakeInvalid
Read request to a Snoopable address region to obtain a snapshot of the coherent data. It is recommended, but not required, that other cached copies of the cache line are invalidated. If a Dirty copy is invalidated, the cache line does not need to be written back to memory. All cached copies must be invalidated if the invalidation hint is accepted and a Dirty copy is not written back to memory.

> **Note**
>
> ReadOnceMakeInvalid is used in preference to ReadOnce or ReadOnceCleanInvalid to obtain a snapshot of a data value when the application determines that the cached data is not going to be used again.
> The application can free up the caches and also, by discarding Dirty data, avoid an unnecessary WriteBack to memory.

The following should be considered when using ReadOnceMakeInvalid:

- The invalidation in the ReadOnceMakeInvalid transaction is a hint. Completion of the transaction does not guarantee removal of all cached copies. The invalidation hint cannot be used as a replacement for a CMO.
- Use of the transaction can cause the deallocation of a cache line and therefore caution is needed if the transactions could target the same cache line that other agents in the system are using for Exclusive accesses.
- The use of the ReadOnceMakeInvalid transaction can cause the loss of a Dirty cache line. Use of this transaction must be strictly limited to scenarios where the loss of a Dirty cache line is known and

harmless.

- For a ReadOnceMakeInvalid transaction, it is required that the invalidation of the cache line is committed before the read data response for the transaction. The invalidation of the cache line is not required to have completed at this point, but any later Write transaction from any agent, which starts after this point, is guaranteed not to be invalidated by this transaction.
- Data must be provided to the Requester in either a I, UC, or UD state.
- The Requester must ignore the cache state in the response.

ReadClean
Read request to a Snoopable address region to obtain a clean copy of a cache line. This can be used if the Requester is allocating the line to a cache that does not support dirty cache lines, such as an Instruction cache. Data must be provided to the Requester in either UC or SC states only.

ReadNotSharedDirty
Read request to a Snoopable address region to carry out a load from the cache line. Data must be provided to the Requester in UC, UD, or SC states only. SD state is not permitted.

> **Note**
>
> ReadNotSharedDirty is used instead of ReadShared when the Requester cannot accept data in SD state.

ReadShared
Read request to a Snoopable address region to carry out a load from the cache line. Data must be provided to the Requester in UC, UD, SC, or SD states.

> **Note**
>
> ReadShared is used instead of ReadNotSharedDirty when the Requester is able to accept data in the SD state.

ReadUnique
Read request to a Snoopable address region to carry out a store to the cache line. Data must be provided to the Requester in UC or UD state only.

ReadPreferUnique
Read request to a Snoopable address region requesting a unique copy of a cache line. ReadPreferUnique is used when the Requester prefers, but does not require, the data to be returned in Unique state:

- Data is provided in Unique state unless another Request Node is currently performing an exclusive sequence using the same address. In which case the data is provided in Shared state.
- It is permitted to always provide the data in Shared state to the Requester.

> **Note**
>
> This request is included in this specification to improve the execution efficiency of an exclusive sequence.

MakeReadUnique
Read request to a Snoopable address region requesting a unique copy of a cache line. Typical usage is when the Requester has a shared copy of the cache line and wants to obtain permission to store to the cache line.

> **Note**
>
> Because data return is guaranteed if the Requester receives an Invalidating snoop and retention of data is required otherwise, resending a request to obtain a Unique copy of the cache line is never required.

B4.2.1.1 Read request attribute values

This section discusses the attribute values for Read requests across node interfaces.

See Chapter C1 *Message Field Mappings* for complete list.

Table B4.1 lists permitted values for the key attributes in Read requests for Request Node to Home Node.

Table B4.1: Request Node to Home Node Read request permitted attribute values

Request	Size (bytes)	Excl	SnpAttr	MemAttr A C D E	Order	LikelyShared	ExpCompAck
ReadNoSnp	<=64	0,1	0	0010	11	0	0,1
				0011	00,10,11	0	0,1
				0000 0001 0101 1101	00,10	0	0,1
ReadOnce ReadOnceCleanInvalid	<=64	0	1	0101 1101	00,10	0	0,1
ReadOnceMakeInvalid	<=64	0	1	0101	00,10	0	0,1
ReadClean ReadNotSharedDirty ReadShared	64	0,1	1	0101 1101	00	0,1	1
ReadUnique	64	0	1	0101 1101	00	0	1
ReadPreferUnique MakeReadUnique	64	0,1	1	0101 1101	00	0	1

Table B4.2 lists the permitted values for the key attributes in Read requests for HN-F to SN-F.

Table B4.2: HN-F to SN-F Read request permitted attribute values

Request	Size (bytes)	Excl	SnpAttr	MemAttr A C D E	Order	LikelyShared	ExpCompAck
ReadNoSnp	<=64	0,1	0	0000 0001 0101 1101	00,01	0	0
ReadNoSnpSep	<=64	0	0	0000 0001 0101 1101	00,01	0	0

Table B4.3 lists the permitted values for the key attributes in Read requests for HN-I to SN-I.

Table B4.3: Request Node to Home Node Read request permitted attribute values

Request	Size (bytes)	Excl	SnpAttr	MemAttr A C D E	Order	LikelyShared	ExpCompAck
ReadNoSnp	<=64	0,1	0	0010	11	0	0
				0011	00,01,10,11	0	0
				0000 0001 0101 1101	00,01,10	0	0
ReadNoSnpSep	<=64	0	0	0010	11	0	0
				0011	00,01,10,11	0	0
				0000 0001 0101 1101	00,01,10	0	0

B4.2.1.2 Initial cache state at the Requester

Table B4.4 lists the permitted Requester cache states when a Request is sent. The following key is used for Table B4.4, Table B4.5 and Table B4.6:

Y Yes, permitted

- Not permitted

n/a Not applicable

Table B4.4: Permitted Requester cache state at the sending of Read request

Request	Initial state						
	UD	UC	SD	SC	I	UDP	UCE
ReadNoSnp	-	-	-	-	Y	-	-
ReadOnce	-	-	-	-	Y	-	-
ReadOnceCleanInvalid	-	-	-	-	Y	-	-
ReadOnceMakeInvalid	-	-	-	-	Y	-	-
ReadClean TagOp = *Transfer*	Y	Y	Y	Y	Y	Y	Y
ReadClean TagOp != *Transfer*	-	-	-	-	Y	-	Y
ReadNotSharedDirty	-	-	-	-	Y	-	Y
ReadShared	-	-	-	-	Y	-	Y
ReadUnique	Y	Y	Y	Y	Y	Y	Y
ReadPreferUnique	-	-	Y	Y	Y	-	Y
MakeReadUnique	-	-	Y	Y	-	-	-

B4.2.1.3 Final cache state at the Requester

Table B4.5 lists the permitted cache state at the Requester at the completion of the transaction.

Table B4.5: Permitted final Requester cache state

Request	Final state						
	UD	UC	SD	SC	I	UDP	UCE
ReadNoSnp	-	-	-	-	Y	-	-
ReadOnce	-	-	-	-	Y	-	-
ReadOnceCleanInvalid	-	-	-	-	Y	-	-
ReadOnceMakeInvalid	-	-	-	-	Y	-	-
ReadClean TagOp = *Transfer*	Y	Y	Y	Y	-	-	-

Continued on next page

Table B4.5 – *Continued from previous page*

Request	Final state						
	UD	**UC**	**SD**	**SC**	**I**	**UDP**	**UCE**
ReadClean TagOp != *Transfer*	-	Y	-	Y	-	-	-
ReadNotSharedDirty	Y	Y	-	Y	-	-	-
ReadShared	Y	Y	Y	Y	-	-	-
ReadUnique	Y	Y	-	-	-	-	-
ReadPreferUnique	Y	Y	Y	Y	-	-	-
MakeReadUnique(non-Excl)	Y	Y	-	-	-	-	-
MakeReadUnique(Excl)	Y	Y	Y	Y	-	-	-

B4.2.1.4 Peer cache state

Table B4.6 lists the permitted cache state at the peer Request Nodes at the completion of the transaction.

In response to a request, the Home must send the appropriate Snoops to ensure cached lines at the peer caches are either an expected or permitted final state.

Table B4.6: Permitted peer cache state at the completion of a Read request

Request	Peer final state							
	UD	**UC**	**SD**	**SC**	**I**	**UDP**	**UCE**	**No Change**
ReadNoSnp	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
ReadOnce	Y	Y	Y	Y	Y	Y	Y	Y
ReadOnceCleanInvalid	Y	Y	Y	Y	Y	Y	Y	Y
ReadOnceMakeInvalid	Y	Y	Y	Y	Y	Y	Y	Y
ReadClean	-	-	Y	Y	Y	-	-	-
ReadNotSharedDirty	-	-	Y	Y	Y	-	-	-
ReadShared	-	-	Y	Y	Y	-	-	-
ReadUnique	-	-	-	-	Y	-	-	-
ReadPreferUnique	-	-	Y	Y	Y	-	-	-
MakeReadUnique[a]	-	-	-	-	Y	-	-	-

[a] For MakeReadUnique(Excl), peer caches can not be required to change state.

B4.2.2 Dataless transactions

Request Nodes use Dataless transactions to perform coherent actions without the transfer of data from or to the Requester. The Dataless transactions, grouped for their common functionality, are:

- CleanUnique, MakeUnique
- Evict
- Independent Stash transactions
  - StashOnceUnique, StashOnceSepUnique, StashOnceShared, StashOnceSepShared
- Cache Maintenance transactions
  - CleanShared, CleanSharedPersist, CleanSharedPersistSep, CleanInvalid, CleanInvalidPoPA, MakeInvalid

Dataless transactions have the following common characteristics:

- Data must not be included in the completion response.
- The Request can result in data movement among other agents in the system.
- The Request can result in a cache state change at the Requester. See Table B4.10 for expected and permitted initial cache and Table B4.11 for final cache state at the Requester for each of the Dataless transactions.
- The Request can result in a cache state change at the other Request Nodes in the system. See Table B4.12 for expected and permitted cache states at the peer Request Nodes for each of the Dataless transactions.

See Chapter B6 *Exclusive accesses* for details on Exclusive attribute in Dataless transactions.

CleanUnique
Request to a Snoopable address region to change the cache state at the Requester to Unique to carry out a store to the cache line. Typical usage is when the Requester has a shared copy of the cache line and wants to obtain permission to store to the cache line. Any dirty copy of the cache line at a snooped cache must be written back to the memory.

MakeUnique
Request to a Snoopable address region to obtain ownership of the cache line without a data response. MakeUnique is only used when the Requester guarantees to carry out a store to all bytes of the cache line. Any dirty copy of the cache line at a snooped cache must be invalidated without carrying out a data transfer.

Evict
Used to indicate that a Clean cache line is no longer cached by the Request Node.

StashOnceUnique, StashOnceSepUnique
Request to a Snoopable address region to attempt to move the addressed cache line to a targeted cache to enable the target to store that line. The request includes:

- A valid node ID of another Request Node as the Stash target, along with an optional LPID within that node. When a valid target is not specified, the addressed cache line can be fetched to be cached at the request Completer.
- It is recommended, but not required, that the other agent is snooped to indicate obtaining the addressed cache line and ensures that being in a cache state suitable for writing to the cache line.
- The DataPull request from the target Request Node is treated as a ReadUnique Request.

See Chapter 7 *Cache Stashing*.

StashOnceShared, StashOnceSepShared
Request to a Snoopable address region to attempt to move the addressed cache line to a targeted cache. The request includes:

- The node ID of another Request Node. Optionally, the request can include the LPID within that node. When a valid target is not specified, the addressed cache line can be fetched to be cached at the request Completer.
- It is recommended, but not required, that the other agent is snooped to indicate that the other agent obtains the addressed cache line.

– The DataPull request from the target Request Node is treated as a ReadNotSharedDirty Request.

See Chapter 7 *Cache Stashing*.

B4.2.2.1 Cache Maintenance transactions

A *Cache Maintenance Operation* (CMO) assists with software cache management. The protocol includes the following five transactions to support a Cache Maintenance Operation:

CleanShared
The completion response to a CleanShared request ensures that all cached copies are changed to a Non-dirty state and any Dirty copy is written back to memory.

CleanSharedPersist
The completion response to a CleanSharedPersist request ensures that all cached copies are changed to a Non-dirty state and any Dirty cached copy is written back to the Point of Persistence (PoP).

CleanSharedPersistSep
The Persist or combined CompPersist completion response to a CleanSharedPersistSep request ensures that all cached copies are changed to a Non-dirty state and any Dirty cached copy is written back to the PoP. Functionality of CleanSharedPersistSep is similar to CleanSharedPersist but allows two separate responses to the Requester.
When sending a PCMO, it is expected, but not required, that a Requester uses a CleanSharedPersistSep transaction instead of CleanSharedPersist.
Such a Requester must support receiving both separate Comp and Persist responses and a combined CompPersist response.

CleanInvalid
The completion response to a CleanInvalid request ensures that all cached copies are invalidated. The request requires that any cached Dirty copies must be written to memory.

CleanInvalidPoPA
The completion response to a CleanInvalidPoPA request ensures that all cached copies prior to the PoPA are invalidated. The request requires that any cached Dirty copies must be written past the PoPA. This enables writes to a location in one PAS to be visible to other Physical Address Spaces. Additional Cache Maintenance Operations could be required in the other Physical Address Spaces to ensure that any update is visible.

MakeInvalid
The completion response to a MakeInvalid request ensures that all cached copies are invalidated. The request permits that any cached Dirty copies are discarded.

The following characteristics are common to all six CMO transactions:

- The Resp field value in the Comp, indicating cache state, must be ignored by both the Requester and the Home.
- Sending of a CMO transaction to the interconnect from a Request Node and from the interconnect to a Subordinate Node is controlled by the **BROADCASTPERSIST (BP)**, **BROADCASTCACHEMAINT (BCM)**, and **BROADCASTCMOPOPA** interface signals. See B16.2 *Optional interface broadcast signals*.

Note

Permitting Cache Maintenance Operations to be forwarded downstream of the Home Node incorporates system topologies where some observers could directly access locations downstream of the Home Node and software cache maintenance is required to make cached data visible to such observers.

- Cacheable and SnpAttr bit values must be observed.

- MemAttr values on a request received at Home must be preserved when the request is propagated to the Subordinate, except when the Subordinate is known to only have Normal memory. In this case, the MemAttr bit Device can be set to Normal.
- Order field must not be asserted:
  - A CMO intended for a particular address must not be sent to the interconnect before all previous transactions sent to the same address that can allocate the data in the Requester caches have completed.
  - A transaction intended for a particular address that can allocate data in Requester caches must not be sent to the interconnect before a previous CMO sent to the same address has completed.
  - A Completer is permitted, but not required, to wait for WriteData to send a Persist response. Examples of when a Completer can send an early Persist is when the Completer does not support Persistence on that memory location or the request encounters a *Non-data Error* (NDERR).
- A CMO is permitted, but not required, to be combined with a Write transaction to the same address when the write is ahead of the CMO. See B4.2.4 *Combined Write requests*.
- When Nonshareable_Cache_Maint property is True, a CMO must apply to locations marked in the page tables as Cacheable, irrespective of being marked Non-shareable or Shareable. The CMO must propagate to any cache that differentiates its entries based on Shareability.
- A CMO must:
  - Propagate to all caches that differentiate based on Cacheability.
  - A CMO must operate on any cache lines that have been cached with the same address and same PAS. A CMO is permitted, but not required, to operate on cache lines in a different PAS. The invalidation of a Dirty cache line in a different PAS must be ensured to be harmless.
- If a line is operated on by a:
  - CleanInvalid, CleanShared, or CleanSharedPersist: the line must propagate to an observable point by all agents that use the same PAS as the CMO.
  - CleanInvalidPoPA: the line must propagate to the observable PoPA by all agents in any PAS.
  - CleanSharedPersist: the line must propagate to the PoP.
  - CleanSharedPersist with Deep attribute set: the line must propagate to the Point of Deep Persistence (PoDP).

B4.2.2.2 Persistent CMO handling at Home

This section discusses the PoP at the Home and downstream of the Home. This section also introduces Deep attribute in Persistent CMO.

B4.2.2.2.1 Point of Persistence at Home

If the Home is the PoP, for CleanSharedPersistSep:

- The Home is permitted to not forward the CleanSharedPersistSep request to the Subordinate. When the Home decides not to forward the CleanSharedPersistSep request, a Persist response must be returned to the Requester.
- When the Home sends a Persist response, it is permitted, but not required, to combine the response with Comp and return a single CompPersist response to the Requester.

B4.2.2.2.2 Point of Persistence downstream of Home

If the PoP is downstream of the Home, for CleanSharedPersistSep:

- The Home must send the request downstream.
- The Subordinate Node must return a Comp response only after guaranteeing the request is accepted. A RetryAck response is not returned.
- When a Subordinate Node can guarantee all previous writes to the same address in a non-volatile memory are persistent, a Persist response must be returned to the Requester or the Home. The address location also contains the updated value even after the removal of power.
- The Home receives a Persist response from downstream. The Persist response is forwarded to the Requester:
  - The Subordinate is permitted, but not required, to return a combined CompPersist response to the Home.
  - The Home is permitted, but not required, to wait for the Persist response from the Subordinate. Permitting a Home to wait for a Persist response enables the Home to send a combined CompPersist response to the Requester instead of returning separate Comp and Persist responses.
- If the target is volatile memory, the Persist response can be given immediately. Such a situation must not return an error.

B4.2.2.2.3 Deep Persistent CMO

For high availability expectations, systems with non-volatile memory require guarantees that operation-critical data is preserved, even when the power and the back up battery fail simultaneously. The guarantee can be provided by a system by adding a mechanism to push previous writes to the PoDP. Deep Persistent CMO is supported by an attribute, called Deep, on PCMO transactions.

If the Completer of the request with Deep asserted does not support the Deep attribute, the Completer can ignore the attribute value and treat the request as having the Deep attribute deasserted. An error response must not be given to indicate that Deep persistence is not supported. See B13.10.19 *Deep persistence, Deep*.

B4.2.2.3 Dataless request attribute values

This section discusses the attribute values for Dataless requests across node interfaces.

See Chapter C1 *Message Field Mappings* for complete list.

Table B4.7 lists the values permitted for key attributes in Dataless requests from Request Node to Home Node.

Table B4.7: Request Node to Home Node Dataless request permitted attribute values

Request	Size (bytes)	Excl	SnpAttr	MemAttr A C D E	Order	LikelyShared	ExpCompAck
CleanUnique	64	0,1	1	0101 1101	00	0	1
MakeUnique	64	0	1	0101 1101	00	0	1

Continued on next page

Table B4.7 – *Continued from previous page*

Request	Size (bytes)	Excl	SnpAttr	MemAttr A C D E	Order	LikelyShared	ExpCompAck
Evict	64	0	1	0101	00	0	0
StashOnceUnique StashOnceSepUnique StashOnceShared StashOnceSepShared	64	0	1	0101 1101	00	0,1	1
CleanShared CleanSharedPersist CleanSharedPersistSep	64	0	1	0101 1101	00[a]	0	0
CleanInvalid CleanInvalidPoPA MakeInvalid		0	0	0010 0011 0000 0001 0101 1101	00[a]	0	0

[a] This field is inapplicable and must be set to 0.

Table B4.8 lists the values permitted for key attributes in Dataless requests from HN-F to SN-F.

Table B4.8: HN-F to SN-F Dataless request permitted attribute values

Request	Size (bytes)	Excl	SnpAttr	MemAttr A C D E	Order	LikelyShared	ExpCompAck
CleanShared CleanSharedPersist CleanSharedPersistSep CleanInvalid CleanInvalidPoPA MakeInvalid	64	0	0	0000 0001 0101 1101	00[a]	0	0

[a] This field is inapplicable and must be set to 0.

Table B4.9 lists the values permitted for key attributes in Dataless requests from HN-I to SN-I.

Table B4.9: HN-I to SN-I Dataless request permitted attribute values

Request	Size (bytes)	Excl	SnpAttr	MemAttr A C D E	Order	LikelyShared	ExpCompAck
CleanShared	64	0	0	0010	00[a]	0	0
CleanSharedPersist				0011			
CleanSharedPersistSep				0000			
CleanInvalid				0001			
CleanInvalidPoPA				0101			
MakeInvalid				1101			

[a] This field is inapplicable and must be set to 0.

B4.2.2.4 Initial cache state at the Requester

Table B4.10 lists the permitted Requester cache states when a request is sent. The following key is used for Table B4.10, Table B4.11 and Table B4.12:

Y Yes, permitted

- Not permitted

Table B4.10: Permitted Requester cache state at the sending of Dataless request

Request	Initial state						
	UD	UC	SD	SC	I	UDP	UCE
CleanUnique	Y	Y	Y	Y	Y	-	Y
MakeUnique	-	Y	Y	Y	Y	-	Y
Evict	-	-	-	-	Y	-	-
StashOnceUnique	-	-	-	-	Y	-	-
StashOnceSepUnique	-	-	-	-	Y	-	-
StashOnceShared	-	-	-	-	Y	-	-
StashOnceSepShared	-	-	-	-	Y	-	-
CleanShared	-	Y	-	Y	Y	-	-
CleanSharedPersist	-	Y	-	Y	Y	-	-
CleanSharedPersistSep	-	Y	-	Y	Y	-	-

Continued on next page

Table B4.10 – *Continued from previous page*

Request	Initial state						
	UD	UC	SD	SC	I	UDP	UCE
CleanInvalid	-	-	-	-	Y	-	-
CleanInvalidPoPA	-	-	-	-	Y	-	-
MakeInvalid	-	-	-	-	Y	-	-

B4.2.2.5 Cache states for Dataless transactions

This section describes B4.2.2.5.1 *Final cache state at the Requester* and B4.2.2.5.2 *Peer cache state* at the completion of a Dataless transaction.

B4.2.2.5.1 Final cache state at the Requester

Table B4.11 lists the permitted cache state at the Requester at the completion of the transaction.

Table B4.11: Permitted final Requester cache state

Request	Final state						
	UD	UC	SD	SC	I	UDP	UCE
CleanUnique	Y	Y	-	-	-	-	Y
MakeUnique	Y	-	-	-	-	-	-
Evict	-	-	-	-	Y	-	-
StashOnceUnique	-	-	-	-	Y	-	-
StashOnceSepUnique	-	-	-	-	Y	-	-
StashOnceShared	-	-	-	-	Y	-	-
StashOnceSepShared	-	-	-	-	Y	-	-
CleanShared	-	Y	-	Y	Y	-	-
CleanSharedPersist	-	Y	-	Y	Y	-	-
CleanSharedPersistSep	-	Y	-	Y	Y	-	-
CleanInvalid	-	-	-	-	Y	-	-
CleanInvalidPoPA	-	-	-	-	Y	-	-
MakeInvalid	-	-	-	-	Y	-	-

B4.2.2.5.2 Peer cache state

Table B4.12 lists the expected and permitted cache state at the peer Request Node at the completion of the Dataless transaction. In response to a request, the Home must send appropriate snoop to transition cached line at the peer caches to either expected or permitted final states.

Table B4.12: Permitted Requester cache state at the completion of a Dataless request

Request	Peer final state							
	UD	UC	SD	SC	I	UDP	UCE	No Change
CleanUnique	-	-	-	-	Y	-	-	-
MakeUnique	-	-	-	-	Y	-	-	-
CleanShared	-	Y	-	Y	Y	-	-	-
CleanSharedPersist	-	Y	-	Y	Y	-	-	-
CleanSharedPersistSep	-	Y	-	Y	Y	-	-	-
CleanInvalid	-	-	-	-	Y	-	-	-
CleanInvalidPoPA	-	-	-	-	Y	-	-	-
MakeInvalid	-	-	-	-	Y	-	-	-

The Peer Request Node cache state at the completion of Evict, StashOnceUnique, StashOnceSepUnique, StashOnceShared, StashOnceSepShared is not applicable.

B4.2.3 Write transactions

Write transactions move data from a Requester to a Completer, this could be the next level cache, memory, or a peripheral. The data being transferred, depending on the transaction type, can be coherent or non-coherent. Each write transaction must assert appropriate BE bits with the data.

B4.2.3.1 Immediate transactions

Immediate transactions, also known as Non-CopyBack Write transactions, are a subclass of Write transactions. Immediate Write transactions transfer data from Request Nodes to Home Nodes without initially obtaining coherent ownership of the data. Immediate Write transactions are also used to transfer data from Home Nodes to Subordinate Nodes.

Each Immediate Write transaction must assert the appropriate BE bits with the data. Immediate Write transactions can require snooping of other agents in the system.

In Immediate Write requests, except WriteNoSnpZero and WriteUniqueZero, DWT flow between the Request Node and the Subordinate Node is permitted only when the original request from the Request Node does not use OWO.

DWT flow between a Request Node and a Subordinate Node in WriteNoSnpZero and WriteUniqueZero is never permitted.

WriteNoSnpFull
Write a full cache line of data from a Request Node to a Non-snoopable address region, or write for a full cache line of data to any address region from the Home to Subordinate. All BE bits must be asserted.

WriteNoSnpPtl
Write up to a cache line of data from a Request Node to a Non-snoopable address region, or write up to a cache line of data to any address region from the Home to Subordinate. BE bits must be asserted for the appropriate byte lanes, including none or all, within the specified data size and deasserted in the rest of the data transfer.

WriteNoSnpDef
Deferrable write of a full cache line of data from a Request Node to a Non-snoopable address region, or write for a full cache line of data to any address region from the Home to Subordinate. All BE bits must be asserted.

> **Note**
>
> Multiple Deferrable write transactions are permitted to be outstanding from the same Requester.

WriteNoSnpZero
Write data value of 0 without transferring data bytes from a Request Node to a Non-snoopable address region, or write data value of 0 without transferring data bytes to any address region from Home to Subordinate.

WriteUniqueFull
Write to a Snoopable address region. Write a full cache line of data to the next-level cache or memory when the cache line is Invalid at the Requester. All BE bits must be asserted.

WriteUniquePtl
Write to a Snoopable address region. Write up to a cache line of data to the next-level cache or memory when the cache line is Invalid at the Requester. BE bits must be asserted for the appropriate byte lanes within the specified data size and deasserted in the rest of the data transfer.

WriteUniqueZero
Write to a Snoopable address region. Write without data bytes when the data value is 0.

WriteUniqueFullStash
Write to a Snoopable address region. Write a full cache line of data to the next-level cache or memory when the cache line is Invalid at the Requester. Also includes a request to the Stash target node to obtain the addressed cache line. All BE bits must be asserted.

WriteUniquePtlStash
Write to a Snoopable address region. Write up to a cache line of data to the next-level cache or memory when the cache line is Invalid at the Requester. Also includes a request to the Stash target node to obtain the addressed cache line. BE bits must be asserted for the appropriate byte lanes within the specified data size and deasserted in the remainder of the data transfer.

B4.2.3.2 CopyBack transactions

CopyBack transactions are a subclass of Write transactions. CopyBack transactions move coherent data from a cache to the next level cache or memory. Each CopyBack transaction must assert the appropriate BE bits with the data. CopyBack transactions do not require the snooping of other agents in the system.

WriteBackFull
WriteBack a full cache line of Dirty data to the next level cache or memory. All BE bits must be asserted except when the write data is CopyBackWriteData_I.

WriteBackPtl
WriteBack up to a cache line of Dirty data to the next level cache or memory. All appropriate BE bits, up to all 64, including none or all, must be asserted.

WriteCleanFull
WriteBack a full cache line of Dirty data to the next level cache or memory and retain a Clean copy in the cache. All BE bits must be asserted except when the write data is CopyBackWriteData_I.

WriteEvictFull
WriteBack of UniqueClean data to the next-level cache.

- All BE bits must be asserted except when the write data is CopyBackWriteData_I.
- The cache line must not propagate beyond its Snoop domain.

WriteEvictOrEvict

WriteBack of Clean data to the next-level cache. This request type is merging of WriteEvictFull and Evict into one request. The Home Node is allowed to determine if data is sent.

- Data might not be sent if the Completer does not accept data.
- If data is sent, then the Data size is a cache line length.
- Table B4.13 indicates the initial state of the cache line when the request is sent:

Table B4.13: LikelyShared value states

LikelyShared	Initial state
0	UC
1	SC

B4.2.3.3 Write request attribute values

This section discusses the attribute values for Write requests across node interfaces.

Table B4.14 lists the values permitted for key attributes in Write requests from Request Node to Home Node.

Table B4.14: Request Node to Home Node Write request attribute values

Request	Size (bytes)	Excl	SnpAttr	MemAttr A C D E	Order	LikelyShared	ExpCompAck
WriteNoSnpFull	64	0,1	0	0010	11	0	0
				0011	00,11	0	0
					10	0	0,1
				0000 0001 0001 1101	00	0	0
					10	0	0,1
WriteNoSnpPtl	<=64	0,1	0	0010	11	0	0
				0011	00,11	0	0
					10	0	0,1
				0000 0001 0101 1101	00	0	0
					10	0	0,1

Continued on next page

Table B4.14 – *Continued from previous page*

Request	Size (bytes)	Excl	SnpAttr	MemAttr A C D E	Order	LikelyShared	ExpCompAck
WriteNoSnpDef	64	0	0	0010	11	0	0
				0011	00,10,11	0	0
				0000 0001	00,10	0	0
WriteNoSnpZero	64	0	0	0010	11	0	0
				0011	00,10,11	0	0
				0000 0001 0101 1101	00,10	0	0
WriteUniqueFull WriteUniqueFullStash	64	0	1	0101 1101	00	0,1	0
					10	0,1	0,1
WriteUniquePtl WriteUniquePtlStash	<=64	0	1	0101 1101	00	0,1	0
					10	0,1	0,1
WriteUniqueZero	64	0	1	0101 1101	00,10	0,1	0
WriteBackFull WriteCleanFull	64	0	1	0101 1101	00	0,1	0
WriteBackPtl	64	0	1	0101 1101	00	0	0
WriteEvictFull	64	0	1	1101	00	0,1	0
WriteEvictOrEvict	64	0	1	1101	00	0,1	1

Table B4.15 lists the values permitted for key attributes in Write requests from HN-F to SN-F.

Table B4.15: HN-F to SN-F Write request permitted attribute values

Request	Size (bytes)	Excl	DoDWT	MemAttr A C D E	Order	LikelyShared	ExpCompAck
WriteNoSnpFull	64	0,1	0,1	0000 0001 0101 1101	00	0[a]	0[a]
WriteNoSnpPtl	<=64	0,1	0,1	0000 0001 0101 1101	00	0[a]	0[a]
WriteNoSnpZero	64	0	0[a]	0000 0001 0101 1101	00	0[a]	0[a]

[a] The field is inapplicable and must be set to 0

Table B4.16 lists the values permitted for key attributes in Write requests from HN-I to SN-I.

Table B4.16: HN-I to SN-I Write request permitted attribute values

Request	Size (bytes)	Excl	DoDWT	MemAttr A C D E	Order	LikelyShared	ExpCompAck
WriteNoSnpFull	64	0,1	0,1	0010	11	0[a]	0[a]
				0011	00, 10, 11	0[a]	0[a]
				0000 0001 0101 1101	00, 10	0[a]	0[a]

Continued on next page

Table B4.16 – *Continued from previous page*

Request	Size (bytes)	Excl	DoDWT	MemAttr A C D E	Order	LikelyShared	ExpCompAck
WriteNoSnpPtl	<=64	0,1	0,1	0010	11	0[a]	0[a]
				0011	00, 10, 11	0[a]	0[a]
				0000 0001 0101 1101	00, 10	0[a]	0[a]
WriteNoSnpDef	64	0	0,1	0010	11	0[a]	0[a]
				0011	00, 10, 11	0[a]	0[a]
				0000 0001	00, 10	0[a]	0[a]
WriteNoSnpZero	64	0	0[a]	0010	11	0[a]	0[a]
				0011	00, 10, 11	0[a]	0[a]
				0000 0001 0101 1101	00, 10	0[a]	0[a]

[a] The field is inapplicable and must be set to 0.

B4.2.3.4 Initial cache state at the Requester

Table B4.17 lists the permitted Requester cache states when a request is sent. The following key is used:

Y Yes, permitted

- Not permitted

Table B4.17: Permitted Requester cache state at the sending of a Write request

Request	Initial state						
	UD	UC	SD	SC	I	UDP	UCE
WriteNoSnpPtl	-	-	-	-	Y	-	-
WriteNoSnpFull	-	-	-	-	Y	-	-
WriteNoSnpDef	-	-	-	-	Y	-	-
WriteNoSnpZero	-	-	-	-	Y	-	-

Continued on next page

Table B4.17 – *Continued from previous page*

Request	Initial state						
	UD	**UC**	**SD**	**SC**	**I**	**UDP**	**UCE**
WriteUniquePtl	-	-	-	-	Y	-	-
WriteUniqueFull	-	-	-	-	Y	-	-
WriteUniqueZero	-	-	-	-	Y	-	-
WriteUniquePtlStash	-	-	-	-	Y	-	-
WriteUniqueFullStash	-	-	-	-	Y	-	-
WriteBackPtl	-	-	-	-	-	Y	-
WriteBackFull	Y	-	Y	-	-	-	-
WriteCleanFull	Y	-	Y	-	-	-	-
WriteEvictFull	-	Y	-	-	-	-	-
WriteEvictOrEvict	-	Y	-	Y	-	-	-

B4.2.3.5 Final cache state at the Requester

The permitted Requester cache state at the completion of a Write transaction, except for WriteCleanFull, is Invalid.

The permitted Requester cache state at the completion of a WriteCleanFull transaction is UC or SC.

B4.2.3.6 Peer cache state

The Peer Request Node cache state at the completion of WriteNoSnpPtl, WriteNoSnpFull, WriteNoSnpDef, and WriteNoSnpZero is not applicable.

The Peer Request Node cache state at the completion of WriteUniquePtl, WriteUniqueFull, WriteUniqueZero, WriteUniquePtlStash, and WriteUniqueFullStash must be Invalid.

The Peer Request Node cache state at the completion of CopyBack requests is not changed. The Home Node is not required to snoop the Peer Request Nodes to change their cache state.

B4.2.4 Combined Write requests

Write transactions can be combined with Cache Maintenance transactions when both are to the same address. The ability to combine the two requests to the same address is useful when a CMO or PCMO transaction reaches a point in the system where a Write operation must be completed before the CMO or PCMO transaction can be initiated. The use of a single Combined Write transaction avoids the need to serialize the Write and CMO or PCMO transactions. The point where the two requests are combined can be a Request Node or Home Node.

Table B4.18 and Table B4.19 show the Write, CMO, and PCMO combinations for which combining is permitted. Empty table cells indicate a combination that is not permitted or not applicable.

Note

Deep is an attribute on Persistent CMO requests. Deep has not been listed explicitly as a CMO type in Table B4.18 and Table B4.19.

When a PCMO is combined with a write, the PCMO is treated as a CleanSharedPersistSep, that is, in

addition to a completion response, a separate Persist response is required for the CMO part of the request.

Table B4.18 and Table B4.19 show the permitted combinations of Write and CMO requests. Table B4.18 and Table B4.19 use the following key:

Y Yes, permitted

- Not permitted

Table B4.18: Permitted combinations of Write and CMO for Request Node to Home Node requests

Write type	PCMO	Non-persistent CMO		
	With separate Persist response	CleanShared	CleanInvalid or CleanInvalidPoPA	MakeInvalid
WriteNoSnpFull WriteNoSnpPtl	Y	Y	Y	-
WriteNoSnpDef	-	-	-	-
WriteUniqueFull WriteUniquePtl	Y	Y	-	-
WriteUniqueStashFull WriteUniqueStashPtl	-	-	-	-
WriteBackFull	Y	Y	Y	-
WriteBackPtl	-	-	-	-
WriteCleanFull	Y	Y	-	-
WriteEvictFull	-	-	-	-

Table B4.19: Permitted combinations of Write and CMO for Home Node to Subordinate Node requests

Write type	PCMO	Non-persistent CMO		
	With separate Persist response	CleanShared	CleanInvalid or CleanInvalidPoPA	MakeInvalid
WriteNoSnpFull WriteNoSnpPtl	Y	Y	Y	-
WriteNoSnpDef	-	-	-	-

Examples of how combining of a Write request with Cache Maintenance is useful are:

- Combining WriteBack with CleanShared and CleanSharedPersist, supports Request Nodes that transition the cache line to Invalid when cleaned by a CMO irrespective of the type of CMO.
- Combining WriteUnique with CleanShared supports the case when the CleanSharedPersist target is known to not support Persistent transactions. Therefore, the CleanSharedPersist request needs to be converted to CleanShared by the Requester.

For each Write request the Combined Write requests are:

- WriteNoSnpFull:
  - WriteNoSnpFullCleanInv
  - WriteNoSnpFullCleanSh
  - WriteNoSnpFullCleanShPerSep
  - WriteNoSnpFullCleanInvPoPA
- WriteNoSnpPtl:
  - WriteNoSnpPtlCleanInv
  - WriteNoSnpPtlCleanSh
  - WriteNoSnpPtlCleanShPerSep
  - WriteNoSnpPtlCleanInvPoPA
- WriteUniqueFull:
  - WriteUniqueFullCleanSh
  - WriteUniqueFullCleanShPerSep
- WriteUniquePtl:
  - WriteUniquePtlCleanSh
  - WriteUniquePtlCleanShPerSep
- WriteBackFull:
  - WriteBackFullCleanInv
  - WriteBackFullCleanSh
  - WriteBackFullCleanShPerSep
  - WriteBackFullCleanInvPoPA
- WriteCleanFull:
  - WriteCleanFullCleanSh
  - WriteCleanFullCleanShPerSep

B4.2.4.1 Characteristics

A Combined Write request is permitted for both CopyBack and Immediate writes.

All permitted behaviors of the Combined Write request are the same as if the write and CMO are sent separately, except that:

- WriteNoSnp(Excl) is not permitted to be combined with a CMO.
- TagOp setting of Match is not permitted in Write*CMO.

A Home receiving a Combined Write request from a Request Node is permitted to forward the Combined Write to the Subordinate Node if both the write and the CMO or PCMO in the request need to be sent downstream. The Home is also permitted to use DWT for such a Write request if the write from the Request Node is an Immediate write and ExpCompAck is set to 0.

Where an uncombined CMO from a Request Node results in the cache line being evicted from a cache at the Home or a Dirty copy is passed in a Snoop response, the CMO propagated to the Subordinate can be combined with the write back to the Subordinate.

The receiver of a Combined Write request is permitted to separate the write and the CMO request and process them separately. In such a case, the CMO request must be ordered behind the write. Once the requests are separated, the Completer is permitted to interleave requests to the same address between the write and the CMO transaction.

The Size field value in a combined request corresponds to the size of Data in the Write request. The CMO is always on a 64-byte granularity.

The MemAttr and SnpAttr field values in a Combined Write request correspond to the memory attributes of the Write request. When the write and the CMO transactions are separated, the Write transaction inherits the MemAttr and SnpAttr values of the original combined request. The SnpAttr and Cacheable bit values of the separated CMO transaction must be set to the most pervasive.

B4.2.5 Atomic transactions

An Atomic transaction permits a Requester to issue a transaction with a memory address and an operation to be performed on the memory address to the interconnect. This transaction type moves the operation closer to where the data resides and is useful for atomically executing an operation and updating the memory location in a performance efficient manner.

Without an Atomic transaction, an atomic operation has to be executed using a sequence of memory accesses. These accesses could rely on Exclusive reads and writes.

By using an Atomic transaction:

- A more deterministic latency can be estimated for atomic operations.
- The blocking period of access to the memory location being modified is reduced, which subsequently reduces the impact on the forward progress of memory accesses by other agents.
- Providing fairness among different Requesters accessing a memory location becomes simpler, because accessing of that memory location by an atomic operation is arbitrated at the *Point of Serialization* (PoS) or *Point of Coherence* (PoC).

This specification defines the following terms relating to atomic operations and Atomic transactions:

Atomic operation The execution of a function involving multiple data values such that, the loading of the original value, the execution of the function, and the storing of the updated value, occurs in an atomic manner. This means that no other agent has access to the location during the entire operation.

Atomic transaction A transaction that is used to pass an atomic operation, along with the data values required for the execution of the atomic operation, from one agent in a system to another, so that the atomic operation can be carried out by a different component in the system than the component that requires the operation to be performed.

B4.2.5.1 Atomic transaction types

The CHI protocol defines four Atomic transaction types:

- AtomicStore
- AtomicLoad
- AtomicSwap
- AtomicCompare

See Chapter B12 *Memory Tagging* for a description of the Memory Tagging mechanism.

For information on the permitted MTE TagOp values for each Atomic request, see Table B12.3.

The following terminology is used to refer to the different data elements in the execution of an atomic operation:

TxnData The write data in the AtomicLoad, and AtomicStore transactions.

CompareData The compare value in the AtomicCompare transaction.

SwapData The swap value in the AtomicCompare, and AtomicSwap transactions.

InitialData The content of the addressed location before the atomic operation.

Enumeration of the four Atomic transaction types is as follows:

AtomicStore

- Sends a single data value with an address and the atomic operation to be performed.
- The target, a Home Node or a Subordinate Node, performs the required operation on the address location specified with the data supplied in the Atomic transaction.
- The target returns a completion response without data.
- Unlike in AtomicLoad transactions, the original value of AtomicStore transactions at the addressed location is not returned to the Requester.
- Number of operations supported is 8.

Table B4.20 shows the eight operations supported by the AtomicStore transaction.

Each of the AtomicStore operations apply to 1 byte, 2 byte, 4 byte, or 8 byte data sizes.

Table B4.20: AtomicStore operations

Operation	Action
STADD	Update location with: `(TxnData + InitialData)`
STCLR	Update location with bitwise: `(InitialData AND (NOT TxnData))`
STEOR	Update location with bitwise: `(InitialData XOR TxnData)`
STSET	Update location with bitwise: `(InitialData OR TxnData)`
STSMAX	Update location with TxnData if: `(((Signed INT) TxnData - (Signed INT) InitialData) > 0)`
STSMIN	Update location with TxnData if: `(((Signed INT) TxnData - (Signed INT) InitialData) < 0)`
STUMAX	Update location with TxnData if: `(((Unsigned INT) TxnData - (Unsigned INT) InitialData) > 0)`
STUMIN	Update location with TxnData if: `(((Unsigned INT) TxnData - (Unsigned INT) InitialData) < 0)`

AtomicLoad

- Sends a single data value with an address and the atomic operation to be performed.
- The target, a Home Node or a Subordinate Node, performs the required operation on the address location specified with the data value supplied in the Atomic transaction.
- The target returns the completion response with data. The data value is the original value at the addressed location.
- Number of operations supported is eight.

Table B4.21 shows the 8 operations supported by the AtomicLoad transaction.

Each of the AtomicLoad operations applies to 1 byte, 2 byte, 4 byte, or 8 byte data sizes.

Table B4.21: AtomicStore operations

Operation	Action
LDADD	Update location with: `(TxnData + InitialData)`
LDCLR	Update location with bitwise: `(InitialData AND (NOT TxnData))`
LDEOR	Update location with bitwise: `(InitialData XOR TxnData)`
LDSET	Update location with bitwise: `(InitialData OR TxnData)`
LDSMAX	Update location with TxnData if: `(((Signed INT) TxnData - (Signed INT) InitialData) > 0)`
LDSMIN	Update location with TxnData if: `(((Signed INT) TxnData - (Signed INT) InitialData) < 0)`
LDUMAX	Update location with TxnData if: `(((Unsigned INT) TxnData - (Unsigned INT) InitialData) > 0)`
LDUMIN	Update location with TxnData if: `(((Unsigned INT) TxnData - (Unsigned INT) InitialData) < 0)`

AtomicSwap

- Sends a single data value, the swap value, together with the address of the location to be operated on.
- The target, a Home Node or a Subordinate Node, swaps the value at the address location with the data value supplied in the transaction.
- The target returns the completion response with data. The data value is the original value at the addressed location.
- Number of operations supported is 1.

AtomicCompare

- Sends two data values, the compare value and the swap value, with the address of the location to be operated on.
- The target, a Home Node or a Subordinate Node, compares the value at the addressed location with the compare value:
  - If the values match, the target writes the swap value to the addressed location.
  - If the values do not match, the target does not write the swap value to the addressed location.

  - The target returns the completion response with data. The data value is the original value at the addressed location.
  - Number of operations supported is 1.

Other common characteristics of Atomic transactions are:

- Data must be included in the completion response to the Requester, except for AtomicStore transaction. For AtomicStore, the completion response does not include the data response.
  - When data is returned, inbound data size must be the same as the outbound data size, except for an AtomicCompare transaction. In AtomicCompare, inbound data size must be half of the outbound data size.
  - The data value in the response data must be the original value at the addressed location.
  - The received data must not be cached at the Requester.
- BE bits must be asserted for all valid data in the outbound data messages. BE bits for inbound data are inapplicable and can take any value.
- The request can result in a cache state change at other Request Nodes in the system. The peer Request Node cache state must be Invalid at the completion of the request.
- Atomic transactions cannot use DMT nor DWT flows.

A Requester that must perform an atomic operation on a cached copy of the line can do the following if the cache line is:

- Unique. The atomic operation can be peformed locally without generating an Atomic transaction.
- Shared but not Dirty. The Requester can either:
  - Generate a ReadUnique, CleanUnique, or MakeReadUnique to gain ownership of the cache line and perform the atomic operation locally.
  - Invalidate the local copy and send the Atomic transaction to the interconnect.
- Shared Dirty. The Requester can either:
  - Generate a ReadUnique, CleanUnique, or MakeReadUnique, gain ownership of the cache line, and perform the operation locally.
  - WriteBack and Invalidate the local copy and subsequently send the Atomic transaction to the interconnect.
- Optionally, in all the above cases, the Requester is permitted, but not required, to send the Atomic transaction with the SnoopMe bit set to direct the interconnect to send a Snoop request to the Requester to invalidate, and if necessary, extract the cached copy. See B13.10.35 *SnoopMe*.

B4.2.5.2 Atomic request attribute values

This section discusses the attribute values for Atomic requests across node interfaces.

Table B4.22 lists the permitted values for attributes for Atomic requests from Request Node to Home Node.

Table B4.22: Request Node to Home Node Atomic request permitted attribute values

Request	Size (bytes)	SnoopMe	SnpAttr	MemAttr A C D E	Order	LikelyShared	ExpCompAck
AtomicStore AtomicLoad AtomicSwap	1,2,4,8	0	0	0010	11	0	0
				0011	00,10,11	0	0
				0000 0001 0101 1101	00,10	0	0
		0,1	1	0101 1101	00,10	0	0
AtomicCompare	2,4,8,16,32	0	0	0010	11	0	0
				0011	00,10,11	0	0
				0000 0001 0101 1101	00,10	0	0
		0,1	1	0101 1101	00,10	0	0

Table B4.23 lists the permitted values for attributes for Atomic requests from HN-F to SN-F.

Table B4.23: HN-F to SN-F Atomic request permitted attribute values

Request	Size (bytes)	SnoopMe	DoDWT	MemAttr A C D E	Order	LikelyShared	ExpCompAck
AtomicStore AtomicLoad AtoicSwap	1,2,4,8	0[a]	0	0000 0001 0101 1101	00	0[a]	0[a]

Continued on next page

Table B4.23 – *Continued from previous page*

Request	Size (bytes)	SnoopMe	DoDWT	MemAttr A C D E	Order	LikelyShared	ExpCompAck
AtomicCompare	2,4,8,16,32	0[a]	0	0000 0001 0101 1101	00	0[a]	0[a]

[a] The field is inapplicable and must be set to 0.

Table B4.24 lists the permitted values for attributes for Atomic requests from HN-I to SN-I.

Table B4.24: HN-I to SN-I Atomic request permitted attribute values

Request	Size (bytes)	SnoopMe	DoDWT	MemAttr A C D E	Order	LikelyShared	ExpCompAck
AtomicStore AtomicLoad AtomicSwap	1,2,4,8	0[a]	0	0010	11	0[a]	0[a]
				0011	00,10,11	0[a]	0[a]
				0000 0001 0101 1101	00,10	0[a]	0[a]

Continued on next page

Table B4.24 – *Continued from previous page*

Request	Size (bytes)	SnoopMe	DoDWT	MemAttr A C D E	Order	LikelyShared	ExpCompAck
AtomicCompare	2,4,8,16,32	0[a]	0	0010	11	0[a]	0[a]
				0011	00,10,11	0[a]	0[a]
				0000	00,10	0[a]	0[a]
				0001			
				0101			
				1101			

[a] The field is inapplicable and must be set to 0.

B4.2.5.2.1 Communicating node pair

See Chapter C2 *Communicating Nodes* for expected and permitted node pairs for each Atomic transaction.

B4.2.5.3 Initial cache state at the Requester

The Requester cache state permitted at the issue of the Atomic transaction is any state.

At the time of issuing an Atomic transaction, if the Requester determines the cache state is not Invalid, or cannot determine that the cache state is Invalid, the value of SnoopMe in the Atomic request must be set to 1.

B4.2.5.4 Final cache state at the Requester

The Requester cache state permitted at the completion of an Atomic transaction is Invalid.

B4.2.5.5 Peer cache state

The peer Request Node cache state at the completion of Atomic transactions is Invalid.

B4.2.6 Other transactions

This section describes the protocol transactions that carry out miscellaneous actions.

See Chapter B12 *Memory Tagging* for a description of the Memory Tagging mechanism.

For information on the permitted MTE TagOp values for each miscellaneous request see Table B12.3.

See Chapter C2 *Communicating Nodes* for expected and permitted node pairs for other transactions.

B4.2.6.1 DVM transactions

DVM transactions are used for virtual memory system maintenance.

DVMOp DVM Operation. Actions include the passing of messages between components within a distributed virtual memory system. See Chapter 8 DVM Operations for details.

B4.2.6.2 Prefetch transaction

The Prefetch target transaction is used to speculatively fetch data from main memory.

PrefetchTgt Prefetch Target. A Request to a Snoopable memory address, sent from a Request Node directly to a Subordinate Node:

- The PrefetchTgt transaction does not include a response.
- The request can be used by the Subordinate Node to fetch the data from off-chip memory. The data can subsequently be buffered in anticipation of a subsequent Read request to the same location.
- The request does not include a response nor RetryAck. The Requester can deallocate the request as soon as the request is sent.
- The Receiver must accept the request without dependency on receiving of a subsequent Read request to the same address.
- The Receiver is permitted to initiate an internal action or discard the request without any further action.
- Data read from off-chip memory using PrefetchTgt must not hold Subordinate Node resources waiting indefinitely for a future Read request to the same address.
- The following fields are inapplicable and can take any value:
  - TxnID
  - Order
  - Endian
  - Size
  - MemAttr
  - SnpAttr
  - Excl
  - LikelyShared

B4.3 Snoop request types

The interconnect generates a Snoop request either in response to a request from a Request Node or due to an internal trigger such as a cache or snoop filter maintenance operation. A Snoop transaction, except for SnpDVMOp, operates on the cached data at the RN-F. A SnpDVMOp transaction carries out a DVM maintenance operation at the target node.

The Home selection of a Snoop to send is based on several criteria:

- Expected or permitted final cache states, required by the request causing the snoop, at the Requester and the snooped nodes.
- Avoid losing any Dirty tags present in the caches being snooped.
- Replacing a Non-forwarding with an equivalent Forwarding snoop, if one exists.
- A Forwarding snoop is permitted to be sent to one RN-F only.
- A stash snoop is permitted to be sent to one RN-F only.
- Snoops are permitted, but not required, to Non-snoopable address locations.

See B4.4 *Request transactions and corresponding Snoop requests*.

See B4.8 *Cache state transitions at a Snoopee* for the permitted responses for specific snoop types.

For details of Snoop transaction interaction with the *Memory Tagging Extension* (MTE) see Chapter B12 *Memory Tagging*.

SnpOnceFwd, SnpOnce
Snoop request to obtain the latest copy of the cache line, preferably without changing the cache line state at the Snoopee.

SnpStashUnique
Snoop request recommending that the Snoopee obtains a copy of the cache line in a Unique state:

- It is expected that the snoop for a StashOnceUnique request is not sent if the cache line is cached in Unique state at the Stash target.
- Permitted, but not required, to send the snoop to the Stash target for WriteUniqueFullStash and WriteUniquePtlStash only if the Snoopee does not have a cached copy of the cache line.
- The Snoopee must not return data with the Snoop response.
- Permits the Snoop response to include a Data Pull.
- Data Pull request in the Snoop response is treated as a ReadUnique.
- Must not change the cache line state at the Snoopee.

SnpStashShared
Snoop request recommending that the Snoopee obtains a copy of the cache line in a Shared state:

- It is expected to not send the snoop if the cache line is cached at the target.
- The Snoopee must not return data with the Snoop response.
- Permits the Snoop response to include a Data Pull.
- Data Pull request in the Snoop response is treated as a ReadNotSharedDirty.
- Must not change the cache line state at the Snoopee.

SnpCleanFwd, SnpClean
Snoop request to obtain a copy of the cache line in Clean state while leaving any cached copy in Shared state. Must not leave the cache line in Unique state.

SnpNotSharedDirtyFwd, SnpNotSharedDirty
Snoop request to obtain a copy of the cache line in SharedClean state while leaving any cached copy in a Shared state. Must not leave the cache line in Unique state.

SnpSharedFwd, SnpShared
Snoop request to obtain a copy of the cache line in Shared state while leaving any cached copy in Shared state. Must not leave the cache line in Unique state.

SnpUniqueFwd, SnpUnique
Snoop request to obtain a copy of the cache line in Unique state while invalidating any cached copies. Must change the cache line to Invalid state.

SnpPreferUniqueFwd, SnpPreferUnique
Snoop request to obtain a copy of the cache line in Unique state while invalidating any cached copies:

- Home is expected to use SnpPreferUniqueFwd or SnpPreferUnique in response to ReadPreferUnique.
- The behavior of the Snoopee is dependent on whether an exclusive sequence is being executed.

SnpUniqueStash
Snoop request to invalidate the cached copy at the Snoopee and recommends that the Snoopee obtains a copy of the cache line in Unique state:

- Permits the Snoop response to include a DataPull.
- Data Pull request in the Snoop response is treated as a ReadUnique.

SnpCleanShared
Snoop request to remove any Dirty copy of the cache line at the Snoopee. Must not leave the cache line in a Dirty state.

SnpCleanInvalid
Snoop request to Invalidate the cache line at the Snoopee and obtain any Dirty copy. Could also be generated by the interconnect without a corresponding request. Must change the cache line to Invalid state.

SnpMakeInvalid
Snoop request to Invalidate the cache line at the Snoopee and discard any Dirty copy:

- Does not return data with the Snoop response, Dirty data is discarded.
- Must change the cache line to Invalid state.

SnpMakeInvalidStash
Snoop request to invalidate the copy of the cache line and recommends that the Snoopee obtains a copy of the cache line in Unique state:

- Snoopee must not return data with the Snoop response, Dirty data must be discarded.
- Permits the Snoop response to include a DataPull.
- Data Pull request in the Snoop response is treated as a ReadUnique.

SnpQuery
SnpQuery probes the state of a cache line at a Request Node:

- Home can send a SnpQuery snoop without any corresponding request from a Requester.
- The Snoop response must include the precise state of the cache line at the targeted Snoopee.
- Snoopee must not return data with the Snoop response.
- The SnpQuery snoop must not change the state of the cache line at the Snoopee.

See B4.7.1.1 *MakeReadUnique transaction* and B6.3.1.1 *MakeReadUnique(Excl)* to see how SnpQuery can be used in the efficient handling of exclusive request flows.

SnpDVMOp

Generated at the interconnect, initiated by the DVMOp request:

- A single DVMOp request generates two snoop requests.
- Returns a single Snoop response for the two Snoop requests.

See B8.2.1 *Non-sync type DVM transaction flow*.

B4.4 Request transactions and corresponding Snoop requests

For a required coherency action, a Home can select from a choice of multiple permitted snoop types. This section describes examples of how a Home selects the snoop to send and the amount of snoops to send.

While responding to a Request, the selection of which snoop to use is determined from the intended outcome, that is the desired final state of the cache line at the Requester and the required or desired cache state at the Snoopee.

See B4.2 *Request types* for required cache state at peer Request Nodes. See B4.8 *Cache state transitions at a Snoopee* for details of Snoopee cache state transitions.

B4.4.1 Number of snoops to send

The Home Node can send Non-forwarding Snoops to more than one RN-F, including to all RN-Fs.

The Home Node is only permitted to send a Forwarding Snoop to one RN-F.

In the case of an invalidation, the invalidating snoop must be sent at least to all the other cached copies. Whereas for a Non-invalidating snoop, the Home Node is:

- Permitted, but not required, to send the snoop to all RN-Fs with the cached copies.
- Must be able to obtain a copy of the Dirty line. In the case of a Unique Dirty cache line state, the Home Node must send a snoop to the RN-F with the Unique Dirty cached copy.

For Write transactions with stash hint, the Home Node must also send invalidating snoops to all Non-stash target RN-Fs that have a copy of the cache line:

- For WriteUniqueFullStash, the expected snoop to Non-stash target nodes is SnpMakeInvalid.
- For WriteUniquePtlStash, the expected snoop to Non-stash target nodes is SnpCleanInvalid.

A snoop filter or directory can be included within the interconnect to support filtering of snoops.

B4.4.2 Selection of snoop to send

A Home Node, based on internal preferences and implementation constraints, might prefer one of the expected snoops or substitute an expected snoop with others. See Table B4.25 for the expected snoop per Request type.

Table B4.25 shows the snoops expected to be used by the Home Node for a given Request. A Home is permitted to substitute the expected snoop with another snoop that accomplishes the required Snoopee state transition.

Table B4.25: Expected Snoop requests per Request from a Request Node

Request category	Request	Expected Snoop
Read	ReadNoSnp	None or SnpOnceFwd[a]
	ReadNoSnpSep	n/a
	ReadOnce	SnpOnceFwd[a]
	ReadOnceCleanInvalid	SnpUnique or SnpOnceFwd[a]
	ReadOnceMakeInvalid	SnpUnique, SnpUniqueFwd[a], or SnpOnceFwd[a]
	ReadClean	SnpCleanFwd
	ReadNotSharedDirty	SnpNotSharedDirtyFwd

Continued on next page

Table B4.25 – *Continued from previous page*

Request category	Request	Expected Snoop
	ReadShared	SnpSharedFwd
	ReadUnique	SnpUniqueFwd
	ReadPreferUnique	SnpPreferUniqueFwd
	MakeReadUnique	SnpCleanInvalid or SnpUniqueFwd[b]
Dataless	CleanUnique	SnpCleanInvalid
	MakeUnique	SnpMakeInvalid
	Evict	None
	CleanShared	SnpCleanShared
	CleanSharedPersist CleanSharedPersistSep	SnpCleanShared
	CleanInvalid CleanInvalidPoPA	SnpCleanInvalid
	MakeInvalid	SnpMakeInvalid
Dataless-stash	StashOnceUnique StashOnceSepUnique	SnpStashUnique
	StashOnceShared StashOnceSepShared	SnpStashShared
Write	WriteNoSnp	None
	WriteNoSnpDef	None
	WriteUniqueFull	SnpMakeInvalid
	WriteUniquePtl	SnpCleanInvalid or SnpUnique
	WriteUniqueZero	SnpMakeInvalid
Write-stash	WriteUniqueFullStash	SnpMakeInvalidStash
	WriteUniquePtlStash	SnpUniqueStash or SnpMakeInvalidStash[c]
Write-CopyBack	WriteBack	None
	WriteClean	
	WriteEvictFull	
	WriteEvictOrEvict	
Atomic	AtomicStore	SnpUnique
	AtomicLoad	SnpUnique
	AtomicSwap	SnpUnique
	AtomicCompare	SnpUnique

Continued on next page

Table B4.25 – *Continued from previous page*

Request category	Request	Expected Snoop
Others	DVMOp	SnpDVMOp
	PCrdReturn	n/a
	PrefetchTgt	

[a] Forwarding snoops cannot be used for requests of less than 64B.

[b] Home is expected to use SnpUniqueFwd if the Requester is determined to have lost its copy of the cache line.

[c] Possible if Home already has the latest copy of the line.

The interconnect has the following behavior when generating a snoop request on receipt of a request from a Request Node:

- This specification supports a snoop filter or directory within the interconnect to track the state of cache lines present in RN-F caches. The tracking can be as detailed as knowing each RN-F that has a copy of the cache line, or as nonspecific as knowing that a cache line is present in one of the RN-F caches. Such tracking permits the interconnect to filter unnecessary snooping of an RN-F, for example:
  - If the snoop filter indicates that the cache line is not present in any of the RN-F caches, the interconnect does not send a snoop request.
  - If the cache line in the RN-F caches is already in the required state, for example the received request is ReadShared and all cached copies of the cache line are in SC state, the interconnect does not send a snoop request.
- The interconnect in response to a WriteUniqueFull, WriteUniqueFullStash, MakeUnique, and MakeInvalid must not use the SnpMakeInvalid snoop request unless either:
  - The transaction TagOp value is Update.
  - The interconnect can determine that the Snoopee does not hold Dirty tags.
- The interconnect in response to a MakeReadUnique must not use the SnpMakeInvalid Snoop request unless the interconnect can determine either that:
  - The Requester still has a cached copy of the cache line and the Snoopee does not have Dirty tags.
  - The Requester has lost its cached copy and the Snoopee does not have a Dirty copy of the cache line.
- It is permitted for the interconnect to generate a snoop request spontaneously without a corresponding request from a Request Node. For example, the interconnect can send a SnpUnique or SnpCleanInvalid request as a result of a backward invalidation from a snoop filter or interconnect cache.
- It is permitted for the interconnect to select which snoop request to send. For example:
  - For a WriteUniquePtl request, either a SnpCleanInvalid or SnpUnique snoop request can be sent. Both of these snoop transactions invalidate the cache line. If the cache line is dirty, data is returned with the response. The write data is written to memory once all Snoop responses are received and the partial data has been merged with any dirty data received with the Snoop response. The only difference in the behavior between the SnpCleanInvalid and SnpUnique snoop requests is that SnpUnique can return data from the UniqueClean (UC) state but SnpCleanInvalid does not. Using SnpUnique therefore could result in an unnecessary data transfer. This example shows the disadvantage of using SnpUnique instead of SnpCleanInvalid in certain circumstances.
- The interconnect is permitted to:

- For ReadNotSharedDirty, ReadShared, and ReadClean transactions, use SnpNotSharedDirty or SnpShared or SnpClean.
- For ReadShared transactions, use SnpNotSharedDirtyFwd or SnpSharedFwd or SnpCleanFwd.
- For ReadNotSharedDirty and ReadClean transactions, use SnpNotSharedDirtyFwd or SnpCleanFwd.
- For the ReadOnce transactions, use any Non-forwarding, Non-invalidating snoop types.
- For the ReadOnceCleanInvalid and ReadOnceMakeInvalid transactions, use any Non-forwarding snoop types except SnpMakeInvalid.
- For the ReadOnce and ReadOnceCleanInvalid transactions, use Forwarding snoop type SnpOnceFwd.
- For ReadOnceMakeInvalid transactions, use Forwarding snoop type SnpUniqueFwd or SnpOnceFwd.
- For WriteUniqueFullStash and WriteUniquePtlStash transactions, send SnpStashUnique or SnpMakeInvalidStash to the target Request Node if the target Request Node does not have the cache line.
- Replace any Invalidating snoop request by the SnpUnique or SnpCleanInvalid request.
- Replace any Forwarding snoop with a corresponding non-Forwarding type. The receiver is permitted to treat the forward indication as a hint and respond to the snoop with a corresponding non-Forwarding version in a protocol-compliant manner. This is permitted irrespective of the use of MTE. Forwarding snoops must not be used for requests of less than 64B.
- Use of SnpMakeInvalid for MakeInvalid and WriteUniqueZero is permitted only when the Home knows that the Snoopee does not have Dirty tags.

B4.5 Response types

Each request can generate one or more responses. Some responses can also include data. A Response is classified as follows:

- B4.5.1 *Completion response*
- B4.5.2 *WriteData response*
- B4.5.3 *Snoop response*
- B4.5.4 *Miscellaneous response*

B4.5.1 Completion response

A completion response is required for all transactions except PCrdReturn and PrefetchTgt. It is typically the last message sent from the Completer, to conclude a request transaction. However, the Requester could still send a CompAck response to conclude the transaction. A completion guarantees that the request has reached a PoS or a PoC, where it will be ordered with respect to requests to the same address from any Requester in the system. See B2.6 *Ordering* for details on the Ordering guarantees.

B4.5.1.1 Read and Atomic transaction completion

A Read completion is either in the form of a single response on the RDAT channel using the CompData opcode, or two separate responses, one on the RSP channel using the RespSepData opcode and a second one on the RDAT channel using the DataSepResp opcode. See B4.5.1.2 *Dataless transaction completion* for completion without data for the MakeReadUnique transaction.

AtomicLoad, AtomicSwap, and AtomicCompare completion is sent on the RDAT channel and uses the CompData opcode.

The CompData and DataSepResp completion responses include the Resp field that indicates the following:

Cache state The final permitted state of the cache line at the Requester for all reads except ReadNoSnp and ReadOnce.

Pass Dirty Indicates if the responsibility for updating memory is passed to the Requester. The assertion of the Pass Dirty bit is shown by _PD in the response name.

When using separate Comp and Data responses, RespSepData also includes the Resp field with Cache state and Pass Dirty indications. The Resp field value in RespSepData must be either inapplicable and set to 0 or the same as in the corresponding DataSepResp.

Table B4.26 shows the permitted Read transaction completion, the encoding of the Resp field, and the meaning of the response. A Subordinate Node can send DataSepResp only in response to ReadNoSnpSep, and only CompData in response to ReadNoSnp.

Table B4.26: Permitted Read transaction completion and Resp field encodings

Response	Resp[2:0]	Final cache line state	Notes
CompData_I	0b000	I	Indicates that a coherent copy of the cache line cannot be kept.
RespSepData_I	0b000	Not applicable	Cache state must be determined from DataSepResp response.

Continued on next page

Table B4.26 – *Continued from previous page*

Response	Resp[2:0]	Final cache line state	Notes
CompData_UC DataSepResp_UC RespSepData_UC	0b010	UC, UCE, SC or I, when the cache state in the response is applicable.	This response is also permitted for ReadNoSnp and ReadOnce transactions but the cache line is not coherent. Responsibility for a Dirty cache line is not being passed.
CompData_SC DataSepResp_SC RespSepData_SC	0b001	SC or I	Responsibility for a Dirty cache line is not being passed.
CompData_UD_PD DataSepResp_UD_PD RespSepData_UD_PD	0b110	UD or SD	Responsibility for a Dirty cache line is being passed.
CompData_SD_PD	0b111	SD	Responsibility for a Dirty cache line is being passed.

In a response with a NDERR indication, the cache state encoded in Resp is permitted to be any value, including reserved values. See B9.1.2 *Error response fields*.

B4.5.1.2 Dataless transaction completion

A completion for Dataless transactions and the MakeReadUnique transaction without data is sent on the CRSP channel and uses the Comp, CompPersist, CompCMO, or CompStashDone opcode.

CompCMO is the completion message for the CMO and PCMO in a Combined Write transaction. CompCMO must only be used in Combined Write transactions. CompCMO can be combined with Persist response as CompPersist opcode.

The Comp response includes the Resp field that indicates the following:

Cache state The final state the cache line is permitted to be in at the Requester, except for CMO transactions. For CMO transactions, the cache state field value in the completion, specifically in Comp, CompCMO and CompPersist transactions, is ignored and the cache state remains unchanged.

Table B4.27 shows the permitted Dataless transaction completion, the encoding of the Resp field, and the meaning of the response.

Table B4.27: Permitted Dataless transaction completion and Resp field encodings

Response	Resp[2:0]	Final cache line state	Notes
Comp_I	0b000	I	
Comp_UC	0b010	UD, UC, UCE, SC, or I	
Comp_SC	0b001	SC or I	
Comp_UD_PD	0b110	UD or SD	Responsibility for a Dirty cache line is being passed.

In a response with a NDERR indication, the cache state encoded in Resp is permitted to be any value, including reserved values. See B9.1.3 *Errors and transaction structure*. See B2.3.2.4 *Combined Immediate Write and CMO*

for CompCMO and CompPersist. See B7.3 *Independent Stash request* for CompStashDone.

B4.5.1.3 Write and Atomic transaction completion

A Write and AtomicStore completion is sent on the CRSP channel and uses the Comp or CompDBIDResp opcode.

No cache state information, or responsibility for a Dirty cache line, is communicated using the Write transaction completion. The Resp field of a Comp or CompDBIDResp response must be set to zero for any Write transaction completion, except for a WriteNoSnpDef transaction. See B4.5.1.3.1 *WriteNoSnpDef* for more information. All cache state information and responsibility for a Dirty cache line are communicated with the WriteData. See B4.5.2 *WriteData response*.

The permitted Write transaction completion responses are:

Comp Used when the completion response is separate from the DBIDResp or DBIDRespOrd response.

CompDBIDResp Used when the completion response is combined with the DBIDResp or DBIDRespOrd response.
All CopyBack requests must use the CompDBIDResp completion response.
Immediate writes and AtomicStore, can either send Comp and DBIDResp or DBIDRespOrd responses separately or can opportunistically combine the two responses and send CompDBIDResp if both are ready to be sent to the Requester.

B4.5.1.3.1 WriteNoSnpDef

The permitted Resp field and RespErr field value combinations in Comp or CompDBIDResp responses for WriteNoSnpDef transactions are shown in Table B4.28.

Table B4.28: WriteNoSnpDef permitted Resp and RespErr field value combinations in Comp and CompDBIDResp

RespErr[1:0]	Resp[2:0]	Response	Description
00	000	OK/Successful	The Completer supports the WriteNoSnpDef transaction and the Deferrable write is successful.
	001	OK/Unsupported	The Completer does not support WriteNoSnpDef transaction. The Requester must complete the transaction flow. The Completer must not update memory.
	010	OK/Defer	The Completer supports WriteNoSnpDef transaction. The transaction cannot be serviced at this time and is not successful. The Memory location is not updated. **Note** The processing of the Defer response and resending of the request is expected to be done by the software running on the Request Node. Hardware at the Request Node is not expected to process the Defer response and resend the request. An OK/Defer decision for one request does not influence the decision on the next ordered request.
10	000	DERR	Data error
11	000	NDERR	Non-data error
Others		Reserved	Reserved

An HN-F is permitted, but not expected, to be the target Home Node for the WriteNoSnpDef request. An HN-F, that receives an unexpected WriteNoSnpDef request, must not forward the WriteNoSnpDef request to an SN-F. The HN-F must return a response of OK/Unsupported.

If a Home Node detects that a Deferrable write is targeting a Subordinate that does not support Deferrable writes, the transaction must not be propagated. The Home is expected to send an OK/Unsupported response. The Home is permitted to send NDERR response instead of the OK/Unsupported response.

B4.5.1.4 Miscellaneous transaction completion

A Comp response, with the Resp field set to 0, is always used for DVM transaction completion.

B4.5.2 WriteData response

The WriteData response is part of Write request and DVMOp transactions. The Requester sends WriteData to the Completer after receiving a guarantee that a buffer is available to accept the data. Buffer availability is signaled through a DBIDResp or DBIDRespOrd response sent from the Completer.

The WriteData response is sent on the WDAT channel and uses the following opcodes.

CopyBackWriteData, CBWrData
Used for WriteBack, WriteClean, WriteEvictFull, and WriteEvictOrEvict, and CopyBack Combined Write transactions. Transfers coherent data from the cache at the Requester to the interconnect. Includes an indication of the cache line state prior to sending the WriteData response.

NonCopyBackWriteData, NCBWrData
Used for WriteUnique and WriteNoSnp, WriteNoSnpDef, and Combined Immediate Write transactions. Also used for a DVMOp transaction. The cache state in the response must be I.

NonCopyBackWriteDataCompAck, NCBWrDataCompAck
Used for Immediate Write and Combined Write transactions. Combined NonCopyBackWriteData and CompAck. The cache state in the response must be I.

WriteDataCancel
Used to inform the Completer that a Write request is canceled before write data is sent.

- A Request Node can send WriteDataCancel instead of NonCopyBackWriteData in WriteNoSnpPtl, WriteUniquePtl, WriteUniquePtlStash, and corresponding Combined Write transactions.
- A Home Node can send WriteDataCancel instead of NonCopyBackWriteData in WriteNoSnpFull, WriteNoSnpPtl, and corresponding Combined Write transactions to the Subordinate Node.
- A Request Node or Home Node can send WriteDataCancel instead of NonCopyBackWriteData in WriteNoSnpDef transactions, if NDERR, OK/Defer, or OK/Unsupported are seen in a:
  - Comp response that arrives before NonCopyBackWriteData is sent
  - CompDBIDResp response
- Must not be used in Write requests to Device memory, except during WriteNoSnpDef transactions.
- All data packets originally intended to be transferred must be sent. BE field value in the WriteDataCancel message must be set to all zeroes.
- Cache state in the response must be I.

The response includes the Resp field, which indicates the following:

Cache state Indicates the state of the cache line before sending the WriteData response. This state can differ from the state of the cache line when the original transaction request was sent if a snoop request, to the same address, is received by the Requester after sending the original transaction request, but before sending the corresponding WriteData response.

Pass Dirty Indicates if the responsibility for updating memory is passed by the Requester. The assertion of the Pass Dirty bit is shown by _PD in the response name.

Table B4.29 shows the permitted WriteData responses, the Opcode and Resp field encodings, and the meaning of the response.

Table B4.29: Permitted WriteData responses, and Opcode and Resp field encodings

Response	DAT Opcode	Resp[2:0]	Cache line state when data was sent	Notes
CopyBackWriteData_I	0x2	0b000	Imprecise and must be ignored.	Indicates a CopyBack request has been canceled. The data in the response must be 0 and all BE must be deasserted.

Continued on next page

Table B4.29 – *Continued from previous page*

Response	DAT Opcode	Resp[2:0]	Cache line state when data was sent	Notes
CopyBackWriteData_UC	0x2	0b010	UC	Data corresponding to a CopyBack request.
CopyBackWriteData_SC	0x2	0b001	SC	Data corresponding to a CopyBack request.
CopyBackWriteData_UD_PD	0x2	0b110	UD or UDP	Data corresponding to a CopyBack request. Responsibility for updating the memory is passed.
CopyBackWriteData_SD_PD	0x2	0b111	SD	Data corresponding to a CopyBack request. Responsibility for updating the memory is passed.
NonCopyBackWriteData	0x3	0b000	I	Data corresponding to an Immediate Write request.
NonCopyBackWriteDataCompAck	0xC	0b000	I	Data corresponding to an Immediate Write request and combined CompAck to indicate that the transaction has completed.
WriteDataCancel	0x7	0b000	I	Indicates an Immediate Write request has been canceled. The data in the response must be 0 and all BE must be deasserted.

Note

The cache line state at the Requester after the Write transaction has completed is not determined from the cache state information in the WriteData response. The cache line can be determined to remain Valid or not after the transaction by the opcode of the transaction:

- A WriteBack or WriteEvictFull transaction must be in I state.
- A WriteClean transaction can remain allocated and be in a Clean state.

B4.5.3 Snoop response

A Snoop transaction includes a Snoop response. A Snoop response can be with or without data. The forms of Snoop response are:

Snoop response without data

- This Snoop response is used when no data transfer is required.
- It is sent on the SRSP channel and uses the SnpResp opcode.
- For Stash snoops, a DataPull request can be included.
- Snoop response without data is always used for the response to a SnpDVMOp transaction.

Snoop response without data to Home and DCT

- This Snoop response is used when the Snoopee sends Data to the Requester and a data transfer to the Home is not required.
- It is sent on the SRSP channel and uses the SnpRespFwded opcode.

Snoop response with data

- This Snoop response is used when a full cache line of data is transferred to Home.
- It is sent on the WDAT channel and uses the SnpRespData opcode.
- For Stash snoops, a DataPull request can be included.

Snoop response with partial data

- This Snoop response is used when a partial cache line of data is transferred to the Home.
- It is sent on the WDAT channel and uses the SnpRespDataPtl opcode.
- For Stash snoops, a DataPull request can be included.
- It is sent when the combination of the Snoop request and cache line state is:
  - Any Snoop request except SnpMakeInvalid, and the cache line state is UDP.

Snoop response with data to Home and DCT

- This Snoop response is used when the Snoopee sends Data to the Requester and a data transfer to the Home is also required.
- It is sent on the DAT channel and uses the SnpRespDataFwded opcode.

The Snoop response includes the Resp field, which indicates the following:

Cache state The final state of the cache line at the snooped node after sending the Snoop response.

Pass Dirty Indicates that the responsibility for updating memory is passed to the Requester or Interconnect. Pass Dirty must only be asserted for a Snoop response with data. The assertion of the Pass Dirty bit is shown by _PD in the response name.

The Snoop response also includes the FwdState field that is applicable in Snoop responses with DCT and indicates the cache state and pass dirty value in the CompData response sent to the Requester.

The Snoop response attributes convey sufficient information for the interconnect to determine both:

- The appropriate response to the initial Requester
- If data must be written back to memory.

The Snoop response attributes are also sufficient to support snoop filter or directory maintenance in the interconnect.

See B12.9 *Snoop requests* for details on Snoop responses and MTE interaction.

Note

The Snoop response cache state information provides the state of the cache line after the Snoop response is sent. This is different from:

- A WriteData response, where the cache state information provides the state of the cache line at the point the write data is sent.
- A read data response, where the cache state information indicates the permitted state of the cache line after the transaction completes.

Table B4.30 shows the permitted Non-forward type snoop responses without data, the RSP Opcode and Resp field encodings, and the meaning of the response.

Table B4.30: Permitted Non-forward type snoop responses without data

Response	RSP Opcode	Resp[2:0]	Cache line state
SnpResp_I	0x1	0b000	I
SnpResp_SC	0x1	0b001	SC or I
SnpResp_UC	0x1	0b010	UC, UCE, SC, or I
SnpResp_UD	0x1	0b010	UD
SnpResp_SD	0x1	0b011	SD

Table B4.31 shows the permitted Forward type snoop responses without data, the RSP Opcode, Resp, and FwdState field encodings, and the meaning of the response. The permitted Forward type snoop responses listed in Table B4.31 forward a copy of the data to the Requester.

Table B4.31: Permitted Forward type snoop responses without data

Response	RSP Opcode	Resp[2:0]	FwdState[2:0]	Cache line state at Snoopee	Forwarded state	Notes
SnpResp_I_Fwded_I	0x9	0b000	0b000	I	I	
SnpResp_I_Fwded_SC	0x9	0b000	0b001	I	SC	
SnpResp_I_Fwded_UC	0x9	0b000	0b010	I	UC	
SnpResp_I_Fwded_UD_PD	0x9	0b000	0b110	I	UD	Responsibility for updating the memory is passed
SnpResp_I_Fwded_SD_PD	0x9	0b000	0b111	I	SD	Responsibility for updating the memory is passed
SnpResp_SC_Fwded_I	0x9	0b001	0b000	SC	I	
SnpResp_SC_Fwded_SC	0x9	0b001	0b001	SC	SC	

Continued on next page

Table B4.31 – *Continued from previous page*

Response	RSP Opcode	Resp[2:0]	FwdState[2:0]	Cache line state at Snoopee	Forwarded state	Notes
SnpResp_SC_Fwded_SD_PD	0x9	0b001	0b111	SC	SD	Responsibility for updating the memory is passed
SnpResp_UC_Fwded_I SnpResp_UD_Fwded_I	0x9	0b010	0b000	UC or UD	I	**Note** A single encoding is used to indicate that the cache line is Unique. This encoding is used for UC and UD.
SnpResp_SD_Fwded_I	0x9	0b011	0b000	SD	I	
SnpResp_SD_Fwded_SC	0x9	0b011	0b001	SD	SC	

Table B4.32 shows the permitted Non-forward type snoop responses with data, the DAT Opcode and Resp field encodings, and the meanings of the response.

Table B4.32: Permitted Non-forward type snoop responses with data

Response	DAT Opcode	Resp[2:0]	Cache line state	Notes
SnpRespData_I	0x1	0b000	I	
SnpRespData_UC SnpRespData_UD	0x1	0b010	UC or UD	**Note** A single encoding is used to indicate that the cache line is Unique. This encoding is used for UC and UD.
SnpRespData_SC	0x1	0b001	SC	
SnpRespData_SD	0x1	0b011	SD	
SnpRespData_I_PD	0x1	0b100	I	Responsibility for updating the memory is passed to the Home
SnpRespData_UC_PD	0x1	0b110	UC	Responsibility for updating the memory is passed to the Home

Continued on next page

Table B4.32 – *Continued from previous page*

Response	DAT Opcode	Resp[2:0]	Cache line state	Notes
SnpRespData_SC_PD	0x1	0b101	SC	Responsibility for updating the memory is passed to the Home
SnpRespDataPtl_I_PD	0x5	0b100	I	Partial data. Responsibility for updating the memory is passed to the Home.
SnpRespDataPtl_UD	0x5	0b010	UDP	Partial data

Table B4.33 shows the permitted Forward type snoop responses with data, the DAT Opcode, Resp, and FwdState field encodings, and the meaning of the response.

Table B4.33: Permitted Forward type snoop responses with data

Response	RSP Opcode	Resp[2:0]	FwdState[2:0]	Cache line state	Forwarded state	Notes
SnpRespData_I_Fwded_SC	0x6	0b000	0b001	I	SC	
SnpRespData_I_Fwded_SD_PD	0x6	0b000	0b111	I	SD	Responsibility for updating the memory is passed to the Requester
SnpRespData_SC_Fwded_SC	0x6	0b001	0b001	SC	SC	
SnpRespData_SC_Fwded_SD_PD	0x6	0b001	0b111	SC	SD	Responsibility for updating the memory is passed to the Requester
SnpRespData_SD_Fwded_SC	0x6	0b011	0b001	SD	SC	
SnpRespData_I_PD_Fwded_I	0x6	0b100	0b000	I	I	Responsibility for updating the memory is passed to the Home
SnpRespData_I_PD_Fwded_SC	0x6	0b100	0b001	I	SC	Responsibility for updating the memory is passed to the Home
SnpRespData_SC_PD_Fwded_I	0x6	0b101	0b000	SC	I	Responsibility for updating the memory is passed to the Home
SnpRespData_SC_PD_Fwded_SC	0x6	0b101	0b001	SC	SC	Responsibility for updating the memory is passed to the Home

The cache line state associated with a Snoop response with data must be a legal value, even if the RespErr field indicates there is a *Data Error* (DERR). A Snoop response with data is not permitted to have a NDERR. See B9.1.4.7 *Snoop transactions*.

In responses to stash snoops, the Snoopee can send a Read request combined with the Snoop response (SnpResp_X_Read), by setting the DataPull bit. The permitted Snoop responses with Data Pull are:

- For SnpUniqueStash:
  - SnpResp_I_Read
  - SnpRespData_I_Read
  - SnpRespData_I_PD_Read
  - SnpRespDataPtl_I_PD_Read
- For SnpMakeInvalidStash:
  - SnpResp_I_Read
- For SnpStashUnique:
  - SnpResp_I_Read
  - SnpResp_UC_Read
  - SnpResp_SC_Read
  - SnpResp_SD_Read
- For SnpStashShared:
  - SnpResp_I_Read
  - SnpResp_UC_Read

B4.5.4 Miscellaneous response

This section describes responses that cannot be classified as a completion, WriteData, or Snoop response.

The miscellaneous responses are:

CompAck

- Sent by the Requester on receipt of the completion response.
- Used by Read, Dataless, WriteNoSnp, WriteUnique, and CopyBack Write transactions.
- The RespErr field is applicable and must be set to 0.
- For Immediate Write transactions, the Resp value in a CompAck response is inapplicable and must be set to 0.
- For CopyBack Write transactions, the Resp value in a CompAck response is applicable. The permitted values for Resp in CompAck are shown in Table B4.34.

See B2.3 *Transaction structure*.

Table B4.34: Permitted CompAck responses for CopyBack Write transactions

Response	RSP Opcode	Resp[2:0]	Cache line state when response was sent	Action when a hidden copy of the cache line is at Home	Notes
CompAck_I	0x2	0b000	Imprecise and must be ignored	Must not yet be exposed	Indicates a CopyBack request has been canceled

Continued on next page

Table B4.34 – *Continued from previous page*

Response	RSP Opcode	Resp[2:0]	Cache line state when response was sent	Action when a hidden copy of the cache line is at Home	Notes
CompAck_UC	0x2	0b010	UC	Expected, but not required, to be exposed, unless a Unique copy still exists elsewhere[a]	
CompAck_SC	0x2	0b001	SC	Expected, but not required, to be exposed	
CompAck_UD_PD	0x2	0b110	UD	Expected, but not required, to be exposed, unless a Unique copy still exists elsewhere[a]	Responsibility for updating the memory is passed
CompAck_SD_PD	0x2	0b111	SD	Expected, but not required, to be exposed	Responsibility for updating the memory is passed

[a] A Unique copy can still exist at the Requester when the CopyBack Write transaction was a WriteClean.

RetryAck

- Sent by a Completer to a Requester if the request is not accepted at the Completer due to lack of appropriate resources.
- Response is permitted for any request transaction except PCrdReturn or PrefetchTgt.
- The RespErr field is applicable and must be set to 0.
- The Resp field is inapplicable and must be set to 0.

See B2.3.8 *Retry*.

PCrdGrant

- Grants a Protocol Credit. A subsequent request, sent using the Protocol Credit, is guaranteed to be accepted by the target.
- The RespErr field is applicable and must be set to 0.
- The Resp field is inapplicable and must be set to 0. See B2.3.8 *Retry*.

ReadReceipt

- Sent for a request that has an ordering requirement with respect to other ordered requests from the same Requester.
- Sent by a Subordinate Node to indicate it has accepted a Read request and will not send a RetryAck response.

- See B2.6.5.1 *Ordering requirements* for how ReadReceipt is used in an ordered request.
- Applies to ReadNoSnp, ReadNoSnpSep, and ReadOnce* request transactions.
- The RespErr field is applicable and must be set to 0.
- The Resp field is inapplicable and must be set to 0.

See B2.3.1 *Read transactions*.

DBIDResp

- Response sent to signal to the Requester that resources are available to accept the WriteData response.
- DBIDResp response also indicates that the Completer provides certain transaction ordering guarantees. See B2.6.5 *Transaction ordering*.
- Applies to Write, Combined Write, DVMOp, and Atomic request transactions.
- The response is permitted from Home Node to Request Node and Subordinate Node to both a Home Node and Request Node.
- The RespErr field is applicable and must be set to 0.
- The Resp field is inapplicable and must be set to 0.

See B2.3 *Transaction structure*.

DBIDRespOrd

- Response sent to signal to the Requester that resources are available to accept the WriteData response.
- DBIDRespOrd response also indicates that the Completer provides certain transaction ordering guarantees. See B2.6.5 *Transaction ordering*.
- Applies to Write, Combined Write, and Atomic request transactions.
- DBIDRespOrd is not permitted in DVM transactions.
- The response is permitted from Home Node to Request Node only.
- The RespErr field is applicable and must be set to 0.
- The Resp field is inapplicable and must be set to 0.

See B2.6.5 *Transaction ordering*.

Persist

- Sent by a Completer for CleanSharedPersistSep transaction to indicate that any data written earlier to the same memory location is made persistent.
- The RespErr field is applicable. See Table B9.5.
- The Resp field is inapplicable and must be set to 0.

See B2.3.2.5 *Combined Immediate Write and Persist CMO* and B2.3.2.6 *Combined CopyBack Write and CMO*.

StashDone

- Sent by a Completer for StashOnceSep to signal the ordering of the request at the Completer.
- The RespErr field is applicable. See Table B9.6.
- The Resp field is inapplicable and must be set to 0.

See B7.3 *Independent Stash request*.

TagMatch

– Sent by the Completer for a Write transaction with TagOp of Match to signal the completion of the Tag Match operation.

– The RespErr field is applicable. See Table B9.9.

– The Resp field is applicable. See Table B13.35.

See B12.11.1 *Tag Match*.

B4.6 Silent cache state transitions

A cache can change state due to an internal event without notifying the rest of the system.

The legal silent cache state transitions are shown in Table B4.35 and Table B4.36. In some cases it is possible, but not required, to issue a transaction to indicate that the transition has occurred. If such a transaction is issued, then the cache state transition is visible to the interconnect and is not classified as a silent transition.

The RN-F action described in Table B4.35 as Local sharing, describes the case where an RN-F specifies a Unique cache line as Shared, effectively disregarding the fact that the cache line remains Unique to the RN-F. For example, this can happen when the RN-F contains multiple internal agents and the cache line becomes shared between them.

For silent cache state transitions:

- Cache eviction and Local sharing transitions can occur at any point and are IMPLEMENTATION DEFINED.
- Store and Cache Invalidate transitions can only occur as the result a deliberate action, which in the case of a core is caused by the execution of a particular program instruction.
- A cache state change from UC to UCE is not permitted.

Table B4.35 indicates how a silent cache state transition can be made non-silent at the interface.

Table B4.35: Legal silent cache state transitions and how they can be made non-silent

RN-F action	Present RN-F state	Next RN-F state	Transaction that can be used to make the action non-silent
Cache eviction	UC	I	Evict, WriteEvictFull, or WriteEvictOrEvict
	UCE	I	Evict
	SC	I	Evict or WriteEvictOrEvict
Local sharing	UC	SC	-
	UD	SD	-
Cache Invalidate	UD	I	Evict
	UDP	I	Evict

Table B4.36 indicates how a silent cache state transition can occur within the RN-F, depending on different types of stores.

Table B4.36: Legal silent cache state transitions resulting from internal RN-F stores

RN-F action	Present RN-F state	Next RN-F state	A result of
Store	UC	UD	Full or partial cache line store
	UCE	UDP	Partial cache line store
		UD	Full cache line store
	UDP	UD	Store that fills the cache line

Note

Sequences of silent transitions can also occur. Any silent transition that results in the cache line being in UD, UDP, or SC state can undergo a further silent transition.

B4.7 Cache state transitions at a Requester

This section specifies the cache state transitions and completion responses for the following request transactions:

- B4.7.1 *Read request transactions*
- B4.7.2 *Dataless request transactions*
- B4.7.3 *Write request transactions*
- B4.7.4 *Atomic transactions*
- B4.7.5 *Other request transactions*

B4.7.1 Read request transactions

Table B4.37 shows the cache state transitions at the Requester, and the completion responses, for Read request transactions except for the MakeReadUnique transaction.

For details of the permitted completion responses and cache state transitions at the Requester for the MakeReadUnique transaction, both Non-exclusive and Exclusive, see B4.7.1.1 *MakeReadUnique transaction*.

The cache state in the Data response to the Requester from the Subordinate Node is UC, that is, CompData_UC or DataSepResp_UC irrespective of the original request type.

If Home sends DataSepResp in response to ReadNoSnp, ReadOnce, ReadOnceCleanInvalid, or ReadOnceMakeInvalid, it is expected to use DataSepResp_UC.

The Requester must ignore the cache state in the CompData or DataSepResp response to ReadNoSnp, ReadOnce, ReadOnceCleanInvalid, and ReadOnceMakeInvalid and implicitly assume the cache state value to be I.

Note

In a non-DMT data transfer, where the CompData response is sent from the Subordinate to Home, the cache state in the response can be either I or UC. It is expected that typically the design of a Subordinate can be simplified by always using UC. Home subsequently sends CompData to the Requester with the appropriate cache state value.

Table B4.37: Cache state transitions at the Requester for Read request transactions

Request type	Initial expected state	Initial permitted state	Final state	Comp responses	Separate responses
ReadNoSnp	I	-	I	CompData_UC, CompData_I	RespSepData + DataSepResp_UC
ReadOnce	I	-	I	CompData_UC, CompData_I	RespSepData + DataSepResp_UC
ReadOnceCleanInvalid	I	-	I	CompData_UC, CompData_I	RespSepData + DataSepResp_UC
ReadOnceMakeInvalid	I	-	I	CompData_UD_PD, CompData_UC, CompData_I	RespSepData + DataSepResp_UC

Continued on next page

Table B4.37 – *Continued from previous page*

Request type	Initial expected state	Initial permitted state	Final state	Comp responses	Separate responses
ReadClean TagOp = *Transfer*	I	-	SC	CompData_SC	RespSepData + DataSepResp_SC
			UC	CompData_UC	RespSepData + DataSepResp_UC
	UC, UCE[be]	-	UC	CompData_SC	RespSepData + DataSepResp_SC
			UC	CompData_UC	RespSepData + DataSepResp_UC
	UD, UDP[ce]	-	UD	CompData_SC	RespSepData + DataSepResp_SC
			UD	CompData_UC	RespSepData + DataSepResp_UC
	SC	-	SC	CompData_SC	RespSepData + DataSepResp_SC
			UC	CompData_UC	RespSepData + DataSepResp_UC
	SD[de]	-	SD	CompData_SC	RespSepData + DataSepResp_SC
			UD	CompData_UC	RespSepData + DataSepResp_UC
ReadClean TagOp != *Transfer*	I	-	SC	CompData_SC	RespSepData + DataSepResp_SC
			UC	CompData_UC	RespSepData + DataSepResp_UC
	UCE	-	UC	CompData_SC	RespSepData + DataSepResp_SC
			UC	CompData_UC	RespSepData + DataSepResp_UC
ReadNotSharedDirty	I, UCE[a]	-	SC	CompData_SC	RespSepData + DataSepResp_SC
			UC	CompData_UC	RespSepData + DataSepResp_UC
			UD	CompData_UD_PD	RespSepData + DataSepResp_UD_PD

Continued on next page

Table B4.37 – *Continued from previous page*

Request type	Initial expected state	Initial permitted state	Final state	Comp responses	Separate responses
ReadShared	I, UCE[a]	-	SC	CompData_SC	RespSepData + DataSepResp_SC
			UC	CompData_UC	RespSepData + DataSepResp_UC
			SD	CompData_SD_PD	-
			UD	CompData_UD_PD	RespSepData + DataSepResp_UD_PD
ReadUnique	I, SC	UC, UCE	UC	CompData_UC	RespSepData + DataSepResp_UC
			UD	CompData_UD_PD	RespSepData + DataSepResp_UD_PD
	SD[e]	UD, UDP[e]	UD	CompData_UC	RespSepData + DataSepResp_UC
				CompData_UD_PD	RespSepData + DataSepResp_UD_PD
ReadPreferUnique	I, SC	UCE	SC	CompData_SC	RespSepData + DataSepResp_SC
			UC	CompData_UC	RespSepData + DataSepResp_UC
			UD	CompData_UD_PD	RespSepData + DataSepResp_UD_PD
	SD[de]	-	SD	CompData_SC	RespSepData + DataSepResp_SC
			UD	CompData_UC	RespSepData + DataSepResp_UC
				CompData_UD_PD	RespSepData + DataSepResp_UD_PD
MakeReadUnique	See Table B4.40 and Table B4.41				

[a] For ReadNotSharedDirty and ReadShared transactions, the Requester with an initial state of UCE must not upgrade the cache line to UDP or UD while the request is outstanding.

[b] A Requester in initial state of UC that receives a CompData_SC or DataSepResp_SC response must stay in UC state. Similarly, a Home that uses a Snoop filter to track the cached state at the Requester, must not downgrade the state of the cache line in the snoop filter based on the state in the response to the Requester.

[c] A Requester in initial state of UD that receives a CompData_SC or DataSepResp_SC response must stay in UD state. Similarly, a Home that uses a Snoop filter to track the cached state at the Requester, must not downgrade the state of the cache line in the snoop filter based on the state in the response to the Requester.

[d] A Requester in initial state of SD that receives a CompData_SC or DataSepResp_SC response must stay in SD state. Similarly, a Home that uses a Snoop filter to track the cached state at the Requester, must not downgrade the state of the cache line in the snoop filter based on the state in the response to the Requester.

[e] Data received from memory must be dropped if the cache state is UD or SD, or merged if the cache state is UDP. Data received from memory must be the same as the cached data when the cache state is SC or UC.

B4.7.1.1 MakeReadUnique transaction

This section describes the permitted responses and the cache state transitions at the Requester, for the MakeReadUnique and MakeReadUnique(Excl) transactions. The additional MakeReadUnique(Excl) behavioral requirements are described in B6.3.1.1 *MakeReadUnique(Excl)*.

B4.7.1.1.1 Permitted responses

Table B4.38 shows the permitted responses in a MakeReadUnique transaction. Table B4.39 shows the additional responses that are permitted only when the Exclusive bit in the request is set to one. This attribute is indicated by adding the suffix (Excl) to the request.

Some key features of the permitted responses are:

- A response without data, Comp_UD_PD, indicates that the Requester is being passed responsibility for a Dirty cache line. This can occur when the Requester holds a SharedClean copy of the cache line and another agent holds a SharedDirty copy. The Home invalidates the SharedDirty copy, for example using a SnpMakeInvalid transaction, and subsequently passes the responsibility for the Dirty cache line to the Requester that issued the MakeReadUnique transaction.
- A response with a cache state of SC is only permitted in response to an Exclusive version of MakeReadUnique. Comp_SC, an example of a response with a cache state of SC, is sent when the Home determines that the Exclusive Store has failed but the snoop filter, or response to a SnpQuery snoop to the Requester, indicates that the Requester still holds a copy of the cache line while there is another shared copy in the system.

> **Note**
>
> The above situation can occur with a non-full address PoC exclusive monitor. The monitor bit for an LP can be reset by another LP performing an Exclusive store to a different address location. This store to a different address does not invalidate the cached copy of the address location being used for Exclusive access by the first Requester.

Table B4.38 shows the permitted responses in both the non-Exclusive and Exclusive MakeReadUnique transaction.

Table B4.38: Permitted responses in the MakeReadUnique transaction

Response received	Cache line state	Action
Comp_UC	Can be Clean or Dirty at the Requester	Requester retains its copy of the cache line All other cached copies have been invalidated.
Comp_UD_PD	Must become Dirty at the Requester	Requester retains its copy of the cache line. Dirty copy of the cache line held elsewhere has been invalidated.
CompData_UC	Clean copy is given to the Requester	Requester lost the cache line while the transaction was in progress. Combined data and completion responses are given.
CompData_UD_PD	Dirty copy is given to the Requester	Requester lost the cache line while the transaction was in progress. Combined data and completion responses are given.

Continued on next page

Table B4.38 – *Continued from previous page*

Response received	Cache line state	Action
RespSepData, DataSepResp_UC	Clean copy is given to the Requester	Requester lost the cache line while the transaction was in progress. Separate data and completion responses are given.
RespSepData, DataSepResp_UD_PD	Dirty copy is given to the Requester	Requester lost the cache line while the transaction was in progress. Separate data and completion responses are given by the Home.

Table B4.39 shows the additional permitted responses in the Exclusive MakeReadUnique transaction.

Table B4.39: Additional permitted responses in the MakeReadUnique(Excl) transaction

Response received	Global Exclusive check	Shared cached copy	Copy of the cache line
Comp_SC	Fail	Exists in another cache	Retained by Requester
CompData_SC	Fail	Exists in another cache	Could be lost by Requester
RespSepData, DataSepResp_SC	Fail	Exists in another cache	Could be lost by Requester

B4.7.1.1.2 Expected snoops

The use of snoops by the Home to invalidate a cache line at the Snoopee in response to a non-Exclusive MakeReadUnique, or an Exclusive MakeReadUnique that passes an Exclusive check, are as follows:

- The Home is expected to use a SnpCleanInvalid snoop:
  - The Home is permitted to use SnpUnique instead of SnpCleanInvalid.
  - The Home is permitted to use SnpUniqueFwd if the Requester is determined to have lost the cached copy of the cache line.
  - If the Home has not invalidated the Requester that sent the MakeReadUnique, it is also permitted to use SnpMakeInalid. The SnpResp_I response from a SD copy can be used to implicitly transfer the Dirty responsibility.

See B6.3.1.1.2 *Home behavior* for the types of snoops the Home is expected and permitted to use when an Exclusive MakeReadUnique request fails the Exclusive check.

B4.7.1.1.3 Cache line state transitions

The final state of the cache line after a MakeReadUnique transaction depends on the state of the cache line immediately before the transaction response is received. This can be different from the state of the cache line at the point the transaction is issued.

For MakeReadUnique, the Requester must keep a copy of the cache line unless an invalidating snoop is received.

- Invalidating snoops are SnpUnique, SnpUniqueFwd, SnpCleanInvalid, SnpMakeInvalid, SnpUniqueStash, and SnpMakeInvalidStash.

- The Requester is permitted to treat SnpPreferUnique and SnpPreferUniqueFwd as either invalidating or non-invalidating. The Home can determine how these snoops are treated by the Snoopee by inspecting the Snoop response.
- All other snoops are non-invalidating and the Requester is required to keep a copy of the cache line.

If the Home does not have an SF, or the SF is imprecise, and the Home cannot determine if the Requester still has a copy of the cache line at the point that the transaction completes, the Home must assume the cache line is lost at the Requester and provide data with the response.

A Requester that receives a response with data, while still holding the line in SD state, must use its own copy of the cache line, rather than the copy returned with the response.

Note

A Requester that receives a response with data, while still holding the line in SD state implies there is no snoop filter present or that the snoop filter is imprecise. In this case, the data could be returned with the response is Stale.

If the Requester is aware that there is no snoop filter, a CleanUnique transaction can be used instead of MakeReadUnique to avoid an unnecessary memory read when the cache line is not cached at any other agent.

Note

Use of the CleanUnique transaction can avoid an unnecessary memory read in the absence of a snoop filter. The unnecessary memory read occurs when the cache line remains cached at the Requester, but a snoop filter is not present in the system or a SnpQuery snoop is not used and a cached copy is not available from another agent in the system. However, using a CleanUnique transaction results in the Requester needing to issue another transaction in the case where the cache line is lost due to a snoop while the transaction is in progress.

Table B4.38 shows the state transitions and responses that are applicable to both non-Exclusive and Exclusive versions of MakeReadUnique. Table B4.41 shows the additional state transitions and responses that apply only to the Exclusive version of the request.

The response rules are:

- The cache state in the response to a non-Exclusive MakeReadUnique must be Unique.
- The cache state in the response to an Exclusive MakeReadUnique is permitted to be Unique or Shared.
- The cache state in the response to an Exclusive and to a Non-exclusive MakeReadUnique must not include Shared Dirty.
- For each permitted combined completion and data response, a corresponding response with separate completion and data is permitted.

Table B4.40 and Table B4.41 include columns for:

- The initial cache state.
- The cache state immediately before the transaction response is received.
- An indication of Home sending the original Requester an invalidating snoop after the Requester has issued the MakeReadUnique request.
- The final state for each of the possible response combinations.

The handling of a MakeReadUnique transaction at the Home depends on the availability or not of a precise Snoop Filter. Table B4.40 and Table B4.41 also include the responses that are permitted with and without a precise Snoop Filter:

- The table column labeled: Snoop Filter - Precise, covers the cases when the precise state of the cache line at the Requester, at the response time, is known. The Home obtains the knowledge of precise state from a snoop filter, or by sending a SnpQuery snoop, or in some other IMPLEMENTATION DEFINED manner.

- The column labeled: Snoop Filter - Imprecise or Absent, covers the cases when the precise state of the cache line at the Requester, at the response time, is not known. This also includes the case when there is no snoop filter, or the Home could decide to ignore the available precise information, or not attempt to obtain the information.

Table B4.40 and Table B4.41 use the following key:

Y Yes, permitted

- Not permitted

Table B4.40: Cache state transitions at the Requester for the MakeReadUnique request (non-Excl and Excl)

Initial state	State at time of response	Home sent invalidating snoop	Final state	Comp response	Precise Snoop Filter	Imprecise or Absent Snoop Filter	Notes
SD	SD	No	UD	Comp_UC	Y	-	
				CompData_UC	-	Y	Returned data could be stale
				RespSepData, DataSepResp_UC	-	Y	
SC, SD	SC	No	UC	Comp_UC	Y	-	
				CompData_UC	-	Y	
				RespSepData, DataSepResp_UC	-	Y	
			UD	Comp_UD_PD	Y	-	
				CompData_UD_PD	-	Y	Data in response is identical to the Requester copy
				RespSepData, DataSepResp_UD_PD	-	Y	
	I	Yes	UC	CompData_UC	Y	Y	Line lost to an invalidating snoop
				RespSepData, DataSepResp_UC	Y	Y	
			UD	CompData_UD_PD	Y	Y	
				RespSepData, DataSepResp_UD_PD	Y	Y	
I, UC, UD	Not permitted						

Table B4.41: Additional cache state transitions at the Requester for the MakeReadUnique request (Excl)

Initial state	State at time of response	State when Home sent invalidating snoop	Final state	Comp response	Precise Snoop Filter	Imprecise or Absent Snoop Filter
SC, SD	SC	No	SC	Comp_SC	Y	-
				CompData_SC	-	Y
				RespSepData DataSepResp_SC	-	Y
	I	Yes	SC	CompData_SC	Y	Y
				RespSepData DataSepResp_SC	Y	Y
SD	SD	No	SD	Comp_SC	Y	-
				CompData_SC	-	Y
				RespSepData DataSepResp_SC	-	Y

See B6.3.1.1 *MakeReadUnique(Excl)* for additional MakeReadUnique(Excl) behavioral requirements.

B4.7.2 Dataless request transactions

Table B4.42 shows the cache state transitions at the Requester, and the completion responses, for Dataless request transactions.

Table B4.42: Cache state transitions at the Requester for Dataless request transactions

Request type	Initial expected state	Initial permitted state	Final state	Comp response
CleanUnique	I	UC, UCE	UCE	Comp_UC
	SC	UC	UC	Comp_UC
	SD	UD	UD	Comp_UC
MakeUnique	I, SC, SD	UC, UCE	UD	Comp_UC
Evict	I	-	I	Comp_I
StashOnceUnique	I	-	I	Comp
StashOnceSepUnique	I	-	I	Comp + StashDone or CompStashDone
StashOnceShared	I	-	I	Comp
StashOnceSepShared	I	-	I	Comp + StashDone or CompStashDone

Continued on next page

Table B4.42 – *Continued from previous page*

Request type	Initial expected state	Initial permitted state	Final state	Comp response
CleanShared, CleanSharedPersist	I, SC, UC	-	No Change	Comp_UC
				Comp_SC
				Comp_I
CleanSharedPersistSep	I, SC, UC	-	No Change	Comp_UC + Persist or CompPersist_UC
				Comp_SC + Persist or CompPersist_SC
				Comp_I + Persist or CompPersist_I
CleanInvalid CleanInvalidPoPA	I	-	I	Comp_I
MakeInvalid	I	-	I	Comp_I

Before a CleanInvalid, CleanInvalidPoPA, MakeInvalid, or Evict transaction, it is permitted for the cache state to be UC, UCE or SC. However, it is required that the cache state transitions to the I state before the transaction is issued. Therefore, Table B4.42 shows I state as the only initial state.

B4.7.3 Write request transactions

Table B4.43 shows the cache state transitions at the Requester, the WriteData response, and the combined or separate completion and DBIDResp response for Write and corresponding Combined Write request transactions. Combined Write request transactions are not listed in Table B4.43. See B4.2.4 *Combined Write requests*.

Table B4.43: Requester cache state transitions for Write request transactions

Request type	Initial state at Requester	State before WriteData or CompAck responses[a]	Final state	Comp response	WriteData or CompAck response
WriteNoSnpPtl	I	-	I	DBIDResp* + Comp or CompDBIDResp	NonCopyBackWriteData or NonCopyBackWriteDataCompAck or WriteDataCancel
WriteNoSnpFull	I	-	I	DBIDResp* + Comp or CompDBIDResp	NonCopyBackWriteData or NonCopyBackWriteDataCompAck
WriteNoSnpDef	I	-	I	DBIDResp* + Comp or CompDBIDResp	NonCopyBackWriteData or WriteDataCancel
WriteNoSnpZero	I	-	I	DBIDResp* + Comp or CompDBIDResp	None

Continued on next page

Table B4.43 – *Continued from previous page*

Request type	Initial state at Requester	State before WriteData or CompAck responses[a]	Final state	Comp response	WriteData or CompAck response
WriteUniquePtl WriteUniquePtlStash	I	I	I	DBIDResp* + Comp or CompDBIDResp	NonCopyBackWriteData or NonCopyBackWriteDataCompAck or WriteDataCancel
WriteUniqueZero	I	I	I	DBIDResp* + Comp or CompDBIDResp	None
WriteUniqueFull WriteUniqueFullStash	I	-	I	DBIDResp* + Comp or CompDBIDResp	NonCopyBackWriteData or NonCopyBackWriteDataCompAck
WriteBackFull	UD	UD	I	CompDBIDResp	CBWrData_UD_PD
				Comp[b]	CompAck_UD_PD
		UC	I	CompDBIDResp	CBWrData_UC
				Comp[b]	CompAck_UC
	UD, SD	SD	I	CompDBIDResp	CBWrData_SD_PD
				Comp[b]	CompAck_SD_PD
		SC	I	CompDBIDResp	CBWrData_SC
				Comp[b]	CompAck_SC
		I	I	CompDBIDResp	CBWrData_I
				Comp[b]	CompAck_I
WriteBackPtl	UDP	UDP	I	CompDBIDResp	CBWrData_UD_PD
		I	I	CompDBIDResp	CBWrData_I
WriteCleanFull	UD	UD	UC	CompDBIDResp	CBWrData_UD_PD
				Comp[b]	CompAck_UD_PD
		UC	UC	CompDBIDResp	CBWrData_UC
				CompDBIDResp	CBWrData_I
				Comp[b]	CompAck_UC
				Comp[b]	CompAck_I

Continued on next page

Table B4.43 – *Continued from previous page*

Request type	Initial state at Requester	State before WriteData or CompAck responses[a]	Final state	Comp response	WriteData or CompAck response
	UD, SD	SD	SC	CompDBIDResp	CBWrData_SD_PD
				Comp[b]	CompAck_SD_PD
		SC	SC	CompDBIDResp	CBWrData_SC
				CompDBIDResp	CBWrData_I
				Comp[b]	CompAck_SC
				Comp[b]	CompAck_I
		I	I	CompDBIDResp	CBWrData_I
				Comp[b]	CompAck_I
WriteEvictFull	UC	UC	I	CompDBIDResp	CBWrData_UC
				Comp[b]	CompAck_UC
		SC	I	CompDBIDResp	CBWrData_SC
				Comp[b]	CompAck_SC
		I	I	CompDBIDResp	CBWrData_I
				Comp[b]	CompAck_I
WriteEvictOrEvict	UC	UC[c]	I	CompDBIDResp	CBWrData_UC
				Comp[b]	CompAck_UC
	UC, SC	SC	I	CompDBIDResp	CBWrData_SC
				Comp[b]	CompAck_SC
		I	I	CompDBIDResp	CBWrData_I
				Comp[b]	CompAck_I

[a] A snoop could be received while a write is pending and results in a cache line state change before the WriteData or CompAck response.

[b] Comp is sent if the Home decides not to request data.

[c] Once the request is sent the Requester is permitted to keep the cache state in UC but must not modify the cache line.

Note

After completion of a WriteClean transaction, the cache line could be in a Unique state to immediately transition to a Dirty state.

B4.7.4 Atomic transactions

Table B4.44 shows the cache state transitions at the Requester, and the completion and response for Atomic transactions.

Table B4.44: Cache state transitions at the Requester for Atomic request transactions

Atomic request	Initial expected state	Initial permitted state	Final state	Comp response	WriteData response
AtomicStore	I, SC, UCE, SD	UC, UD, UDP	I	DBIDResp* + Comp_I or CompDBIDResp	NonCopyBackWriteData
AtomicLoad	I, SC, UCE, SD	UC, UD, UDP	I	DBIDResp* + CompData_I	NonCopyBackWriteData
AtomicSwap	I, SC, UCE, SD	UC, UD, UDP	I	DBIDResp* + CompData_I	NonCopyBackWriteData
AtomicCompare	I, SC, UCE, SD	UC, UD, UDP	I	DBIDResp* + CompData_I	NonCopyBackWriteData

B4.7.5 Other request transactions

DVMOp and PrefetchTgt requests do not have any cache state transitions associated with them.

B4.8 Cache state transitions at a Snoopee

This section specifies the cache state transitions and completion responses for the following Snoop transactions:

- B4.8.1 *Non-Forwarding and Non-stash Snoop transactions*
- B4.8.2 *Stash Snoop transactions*
- B4.8.3 *Forwarding Snoop transactions*

A Snoopee, an RN-F that receives a snoop, performs two actions. One action is a state change of the cached line and the second action is sending a response message either to the Home, or to both the Home and the Requester.

The cache state change depends on the snoop type, the initial state of the cache line and the value of DoNotGoToSD in the snoop. See B4.10 *Do not transition to SD*.

The Snoopee must send a response to Home either with Data or without Data. In addition, for Forwarding snoops the Snoopee can also forward a Data response to the Requester.

The type of response sent is determined by the snoop type, initial cache state, cache state change, and the value of RetToSrc. See B4.9 *Returning Data with Snoop response*.

B4.8.1 Non-Forwarding and Non-stash Snoop transactions

The Non-forwarding and Non-stash Snoop transactions are:

- B4.8.1.1 *SnpOnce*
- B4.8.1.2 *SnpClean, SnpShared, SnpNotSharedDirty, and SnpPreferUnique*
- B4.8.1.3 *SnpUnique and SnpPreferUnique*
- B4.8.1.4 *SnpCleanShared, SnpCleanInvalid, and SnpMakeInvalid*
- B4.8.1.5 *SnpQuery*

B4.8.1.1 SnpOnce

Table B4.45 shows for SnpOnce, the initial, expected final, and other permitted final cache states at the snooped Requester, the RetToSrc field value, and the valid completion response from a snooped RN-F for SnpOnce.

Table B4.45: SnpOnce Cache state transitions, RetToSrc value, and valid completion responses

Initial state	Final expected state	Final permitted state	RetToSrc [a]	Snoop response
I	I	-	X	SnpResp_I
UC	UC	I, SC	X	SnpResp_UC
				SnpRespData_UC
	SC	I	X	SnpResp_SC
				SnpRespData_SC
	I	-	X	SnpResp_I
				SnpRespData_I
UCE	UCE	I	X	SnpResp_UC
	I	-	X	SnpResp_I

Continued on next page

Table B4.45 – *Continued from previous page*

Initial state	Final expected state	Final permitted state	RetToSrc [a]	Snoop response
UD	UD	SD	X	SnpRespData_UD
	SD	-	X	SnpRespData_SD
	SC	I	X	SnpRespData_SC_PD
	I	-	X	SnpRespData_I_PD
UDP	I	-	X	SnpRespDataPtl_I_PD
	UDP	-	X	SnpRespDataPtl_UD
SC	SC	I	0	SnpResp_SC
			1	SnpRespData_SC
	I	-	0	SnpResp_I
			1	SnpRespData_I
SD	SD	-	X	SnpRespData_SD
	SC	I	X	SnpRespData_SC_PD
	I	-	X	SnpRespData_I_PD

[a] X indicates that the protocol requirements apply for both states of RetToSrc.

B4.8.1.2 SnpClean, SnpShared, SnpNotSharedDirty, and SnpPreferUnique

Table B4.46 shows the initial, expected final, and other permitted final cache states at the snooped Requester, the RetToSrc field value, and the valid completion response from a snooped RN-F for SnpClean, SnpShared, SnpNotSharedDirty and SnpPreferUnique. Table B4.46 must be used to determine SnpPreferUnique cache state transitions when the Snoopee is in the middle of executing an Exclusive access sequence. A Snoopee that is not in the middle of executing an Exclusive access is permitted, but not expected, to use Table B4.46 to determine SnpPreferUnique cache state transitions.

SnpPreferUnique is applicable when the Snoopee is in the middle of executing an Exclusive sequence. SnpPreferUnique is protocol compliant at any time. See also Table B4.47.

See B4.8.3.5 *SnpPreferUnique and SnpPreferUniqueFwd* for constraints on determining when a Requester is in the middle of executing an exclusive sequence.

Table B4.46: SnpClean, SnpShared, SnpNotSharedDirty and SnpPreferUnique cache state transitions, RetToSrc value, and valid completion responses

Initial state	Final expected state	Final permitted state	RetToSrc [a]	Snoop response
I	I	-	X	SnpResp_I
UC	SC	I	X	SnpResp_SC
				SnpRespData_SC
	I	-	X	SnpResp_I
				SnpRespData_I
UCE	I	-	X	SnpResp_I
UD	SD[b]	-	X	SnpRespData_SD
	SC	I	X	SnpRespData_SC_PD
	I	-	X	SnpRespData_I_PD
UDP	I	-	X	SnpRespDataPtl_I_PD
SC	SC	I	0	SnpResp_SC
			1	SnpRespData_SC
	I	-	0	SnpResp_I
			1	SnpRespData_I
SD	SD[b]	-	X	SnpRespData_SD
	SC	I	X	SnpRespData_SC_PD
	I	-	X	SnpRespData_I_PD

[a] X indicates that the protocol requirements apply for both states of RetToSrc.

[b] This state transition is not permitted if DoNotGoToSD is set.

B4.8.1.3 SnpUnique and SnpPreferUnique

Table B4.47 shows the initial, expected final, and other permitted final cache states at the snooped Requester, the RetToSrc field value, and the valid completion response from a snooped RN-F for SnpUnique, for SnpUnique and SnpUniquePrefer. Table B4.47 is expected to be used to determine SnpPreferUnique cache state transitions when the Snoopee is not in the middle of executing an Exclusive access sequence.

SnpPreferUnique is only applicable when the Snoopee is not executing an Exclusive sequence.

Table B4.47: SnpUnique and SnpUniquePrefer cache state transitions, RetToSrc value, and valid completion responses

Initial state	Final expected state	Final permitted state	RetToSrc [a]	Snoop response
I	I	-	X	SnpResp_I
UC	I	-	X	SnpResp_I
				SnpRespData_I
UCE	I	-	X	SnpResp_I
UD	I	-	X	SnpRespData_I_PD
UDP	I	-	X	SnpRespDataPtl_I_PD
SC	I	-	0	SnpResp_I
			1	SnpRespData_I
SD	I	-	X	SnpRespData_I_PD

[a] X indicates that the protocol requirements apply for both states of RetToSrc.

B4.8.1.4 SnpCleanShared, SnpCleanInvalid, and SnpMakeInvalid

Table B4.48 shows the initial, expected final, and other permitted final cache states at the snooped Requester, the RetToSrc field value, and the valid completion response from a snooped RN-F for SnpCleanShared, SnpCleanInvalid, and SnpMakeInvalid.

Table B4.48: Cache state transitions, RetToSrc value, and valid completion responses

Snoop request type	Initial state	Final expected state	Final permitted state	RetToSrc	Snoop response
SnpCleanShared	I	I	-	0	SnpResp_I
	UC	UC	I, SC	0	SnpResp_UC
		SC	I	0	SnpResp_SC
		I	-	0	SnpResp_I
	UCE	I	-	0	SnpResp_I
	UD	UC	I, SC	0	SnpRespData_UC_PD
		SC	I	0	SnpRespData_SC_PD
		I	-	0	SnpRespData_I_PD
	UDP	I	-	0	SnpRespDataPtl_I_PD
	SC	SC	I	0	SnpResp_SC
		I	-	0	SnpResp_I

Continued on next page

Table B4.48 – *Continued from previous page*

Snoop request type	Initial state	Final expected state	Final permitted state	RetToSrc	Snoop response
	SD	SC	I	0	SnpRespData_SC_PD
		I	-	0	SnpRespData_I_PD
SnpCleanInvalid	I	I	-	0	SnpResp_I
	UC	I	-	0	SnpResp_I
	UCE	I	-	0	SnpResp_I
	UD	I	-	0	SnpRespData_I_PD
	UDP	I	-	0	SnpRespDataPtl_I_PD
	SC	I	-	0	SnpResp_I
	SD	I	-	0	SnpRespData_I_PD
SnpMakeInvalid	I	I	-	0	SnpResp_I
	UC	I	-	0	SnpResp_I
	UCE	I	-	0	SnpResp_I
	UD	I	-	0	SnpResp_I
	UDP	I	-	0	SnpResp_I
	SC	I	-	0	SnpResp_I
	SD	I	-	0	SnpResp_I

B4.8.1.5 SnpQuery

Table B4.49 shows the initial, expected final, and other permitted final cache states at the snooped Requester, the RetToSrc field value, and the valid completion response from a snooped RN-F for SnpQuery.

Table B4.49: SnpQuery cache state transitions, RetToSrc value, and valid completion responses

Initial state	Final expected state	Final permitted state	RetToSrc	Snoop response
I	I	-	0	SnpResp_I
UC	UC	-	0	SnpResp_UC
UCE	UCE	-	0	SnpResp_UC
UD	UD	-	0	SnpResp_UD
UDP	UDP	-	0	SnpResp_UD
SC	SC	-	0	SnpResp_SC
SD	SD	-	0	SnpResp_SD

B4.8.2 Stash Snoop transactions

The following sub-sections show the permitted responses for the Stash type snoops:

- B4.8.2.1 *SnpUniqueStash and SnpMakeInvalidStash*
- B4.8.2.2 *SnpStashUnique and SnpStashShared*

B4.8.2.1 SnpUniqueStash and SnpMakeInvalidStash

The permitted responses to SnpUniqueStash and SnpMakeInvalidStash are the same as the responses to SnpUnique and SnpMakeInvalid respectively.

The RetToSrc bit value must not be set to 1 in SnpUniqueStash and SnpMakeInvalidStash.

Any Snoop response to SnpUniqueStash and SnpMakeInvalidStash can include a DataPull request. A DataPull request in a Snoop response to SnpUniqueStash and SnpMakeInvalidStash must be treated as a ReadUnique request.

Table B4.50 shows the Snoopee cache state transitions and required Snoop responses for SnpUniqueStash and SnpMakeInvalidStash. The Snoop responses do not include the DataPull options. DataPull is permitted with any Snoop response.

Table B4.50: Snoop response to SnpUniqueStash and SnpMakeInvalidStash

Snoop request type	Initial state	Final expected state	Final permitted state	RetToSrc	Snoop response
SnpUniqueStash	I	I	-	0	SnpResp_I
	UC	I	-	0	SnpRespData_I
					SnpResp_I
	UCE	I	-	0	SnpResp_I
	UD	I	-	0	SnpRespData_I_PD
	UDP	I	-	0	SnpRespDataPtl_I_PD
	SC	I	-	0	SnpResp_I
	SD	I	-	0	SnpResp_I_PD
SnpMakeInvalidStash	Any	I	-	0	SnpResp_I

B4.8.2.2 SnpStashUnique and SnpStashShared

For SnpStashUnique and SnpStashShared, the Snoopee must not change cache state.

The Snoopee is permitted to not perform a cache lookup before responding, in which case the Snoop response must be SnpResp_I.

The Snoopee is permitted to include the precise cache state in the response.

A Snoop response can include DataPull only if the cache state in the response is precise.

The Snoopee can include a DataPull in response to a SnpStashUnique only if the cache-data is not present, or present in a Shared state. The Snoopee can include Data Pull in response to SnpStashShared only if the cache cache-data is not present.

A DataPull request in a Snoop response to SnpStashUnique must be treated as a ReadUnique request. A DataPull request in a Snoop response to SnpStashShared must be treated as a ReadNotSharedDirty request.

The inclusion of DataPull in the Snoop response must ensure that the initial state must not violate the initial state conditions permitted for the corresponding independent Read requests. See B4.2.1 *Read transactions*.

Table B4.51 shows the Snoopee cache state transitions, the required Snoop responses, and DataPull options for SnpStashUnique.

Table B4.51: Snoop response to SnpStashUnique

Initial state	Final expected state	Final permitted state	RetToSrc	Snoop response
I	I	-	0	SnpResp_I
				SnpResp_I_Read
UC	UC	-	0	SnpResp_UC
				SnpResp_I
UCE	UCE	-	0	SnpResp_UC
				SnpResp_UC_Read
				SnpResp_I
UD	UD	-	0	SnpResp_UD
				SnpResp_I
UDP	UDP	-	0	SnpResp_UD
				SnpResp_I
SC	SC	-	0	SnpResp_SC
				SnpResp_SC_Read
				SnpResp_I
SD	SD	-	0	SnpResp_SD
				SnpResp_SD_Read
				SnpResp_I

Table B4.52 shows the Snoopee cache state transitions, the required Snoop responses, and Data Pull options for SnpStashShared.

Table B4.52: Snoop response to SnpStashShared

Initial state	Final expected state	Final permitted state	RetToSrc	Snoop response
I	I	-	0	SnpResp_I_Read
				SnpResp_I
UC	UC	-	0	SnpResp_UC
				SnpResp_I

Continued on next page

Table B4.52 – *Continued from previous page*

Initial state	Final expected state	Final permitted state	RetToSrc	Snoop response
UCE	UCE	-	0	SnpResp_UC
				SnpResp_UC_Read
				SnpResp_I
UD	UD	-	0	SnpResp_UD
				SnpResp_I
UDP	UDP	-	0	SnpResp_UD
				SnpResp_I
SC	SC	-	0	SnpResp_SC
				SnpResp_I
SD	SD	-	0	SnpResp_SD
				SnpResp_I

B4.8.3 Forwarding Snoop transactions

Forwarding (Fwd) type snoops are used by Home to support DCT. The Forwarding Snoop transactions are:

- B4.8.3.1 *SnpOnceFwd*
- B4.8.3.2 *SnpCleanFwd, SnpNotSharedDirtyFwd*
- B4.8.3.3 *SnpSharedFwd*
- B4.8.3.4 *SnpUniqueFwd*
- B4.8.3.5 *SnpPreferUnique and SnpPreferUniqueFwd*

The rules, common to all Forwarding snoops at the Snoopee are:

- Expected, but not required, to forward a copy of the cache line to the Requester if the cache line is in one of the following states:
  - UD
  - UC
  - SD
  - SC
- Not permitted to set the FwdNID to the same node ID as the Snoopee. That is, forwarded data cannot be requested from a Snoopee to itself.
- The Snoopee is permitted, but not expected, to convert the Snoop to its corresponding Non-forwarding type.
- Must not forward data in Unique state in response to a Non-invalidating type snoop.
- Snoopee receiving a Snoop request with the DoNotGoToSD bit set, except when the Snoop is SnpOnceFwd, must not transition to SD, even if the coherency conditions permit it.
- In certain cases, based on the Snoop type, the state of the cache line at the Snoopee, and the RetToSrc value in the Snoop request, the Snoopee forwards a copy to Home along with a copy to the Requester. Forwarding snoops must not be used if the original request requests tags. If tags are available, Clean tags are permitted to be forwarded to the Requester along with the data.
- Home is not permitted to send a Forwarding type snoop for:
  - Atomic transactions

– Passing Exclusive read transactions

> **Note**
>
> Exclusive read transactions that fail due to Non-exclusive support for the address range being accessed are treated as corresponding Non-exclusive reads, Home can therefore use Forwarding type snoops in these cases.

For the rules that are specific to a particular Forwarding snoop see the individual sub-sections in B4.8.3 *Forwarding Snoop transactions*.

The tables in the following individual sub-sections show the Snoopee state transitions and the corresponding responses to the Requester and the Home.

The first column in the tables shows the initial cache state, and is the combined data and tag state. Table B4.53 shows the combined state used and the corresponding data and possible tag state combinations.

Table B4.53: Combined state and the corresponding data and tag states

Combined state	Data state	Tag state
I	I	I
UC	UC	I
		Clean
UCE	UCE	I
UD	UD	I
		Clean
		Dirty
UDP	UDP	I
SC	SC	I
		Clean
SD	SD	I
		Clean
		Dirty

The last three columns in the tables correspond to the TagOp value in the Data responses to Home. They are organized, based on the initial state of the tags:

Dirty Two columns:

– First column: Indicates if the transition itself is permitted or not. This column is only relevant when the tag initial state is Dirty. Table B4.54 shows the conventions used, P and NP are only relevant when the data initial state is UD or SD and the tag initial state is Dirty.

Table B4.54: Key to table conventions

Symbol	Description
P	The transition is permitted
NP	The transition is not permitted
-	TagOp is not applicable

  - Second column: The response TagOp value when the transition is permitted. The TagOp value in the response:
    - Must be Update if the response state includes Pass Dirty.
    - Must be Transfer if the response does not include Pass Dirty.

Invalid, Clean One column:

  - TagOp in the response can be:
    - *Invalid* if the initial tag state is Invalid or Clean.
    - *Transfer* when the initial tag state is Clean.

In Snoop responses to Home without data, TagOp is inapplicable.

B4.8.3.1 SnpOnceFwd

The rules, in addition to the common rules, to be followed by a Snoopee that receives a SnpOnceFwd are:

- Snoopee must forward the cache line in I state.
  - As a consequence, the Snoopee must not forward Pass Dirty to the Requester.
- Snoopee must return data to Home only when Dirty state is changed to Clean or Invalid.
- RetToSrc bit in the snoop must be set to 0.
- Snoopee can ignore the DoNotGoToSD value in the snoop.

Table B4.55 shows the Snoopee cache state transitions and the required Snoop responses.

See Table B4.54 for the symbol key.

Table B4.55: SnpOnceFwd Snoopee state transitions with Clean and Dirty tags

Snoopee cache state				Response to Requester	Response to Home	TagOp value in response to Home		
Initial	Final expected	Final permitted	RetToSrc			Initial start state of Dirty[a]	Initial start state of Dirty	Initial start state of Invalid or Clean
I	I	-	0	No Fwd	SnpResp_I	-	-	-
UC	UC	-	0	CompData_I	SnpResp_UC_Fwded_I	-	-	-
	SC	I	0	CompData_I	SnpResp_SC_Fwded_I	-	-	-
	I	-	0	CompData_I	SnpResp_I_Fwded_I	-	-	-
UCE	UCE	-	0	No Fwd	SnpResp_UC	-	-	-
	I	-	0	No Fwd	SnpResp_I	-	-	-

Continued on next page

Table B4.55 – *Continued from previous page*

Snoopee cache state				Response to Requester	Response to Home	TagOp value in response to Home		
Initial	**Final expected**	**Final permitted**	**RetToSrc**			**Initial start state of Dirty[a]**	**Initial start state of Dirty**	**Initial start state of Invalid or Clean**
UD	UD	-	0	CompData_I	SnpResp_UD_Fwded_I	P[b]	-	-
	SD	-	0	CompData_I	SnpResp_SD_Fwded_I	P[b]	-	-
	SC	I	0	CompData_I	SnpRespData_SC_PD_Fwded_I	P	Update[c]	I,Transfer[d]
	I	-	0	CompData_I	SnpRespData_I_PD_Fwded_I	P	Update	I,Transfer
UDP	UDP	-	0	No Fwd	SnpRespDataPtl_UD	-	-	I
	I	-	0	No Fwd	SnpRespDataPtl_I_PD	-	-	I
SC	SC	I	0	No Fwd	SnpResp_SC	-	-	-
				CompData_I	SnpResp_SC_Fwded_I	-	-	-
	I	-	0	No Fwd	SnpResp_I	-	-	-
				CompData_I	SnpResp_I_Fwded_I	-	-	-
SD	SD	-	0	CompData_I	SnpResp_SD_Fwded_I	P[b]	-	-
	SC	I	0	CompData_I	SnpRespData_SC_PD_Fwded_I	P	Update	I,Transfer
	I	-	0	CompData_I	SnpRespData_I_PD_Fwded_I	P	Update	I,Transfer

[a] Indicates if the transition itself is permitted or not. This column is only relevant when the tag initial state is Dirty.

[b] This transition is permitted, even though the Snoop response does not include data and the tag state is Dirty, because Dirty tags are retained by the Snoopee.

[c] This transition from tag state of Dirty is permitted because a Dirty copy of tags and data are passed to the Home.

[d] This transaction from an initial state of Dirty data, and Invalid or Clean tags, includes the Snoop response of data with Clean tags when available.

B4.8.3.2 SnpCleanFwd, SnpNotSharedDirtyFwd

The rules, in addition to the common rules, to be followed by a Snoopee that receives a SnpCleanFwd or a SnpNotSharedDirtyFwd are:

- Snoopee must forward the cache line in SC state.
- Snoopee must transition to either SD, SC, or I state.
- For behavior related to the RetToSrc bit see B4.9 *Returning Data with Snoop response*.

Table B4.56 shows the Snoopee cache state transitions and the required Snoop responses.

See Table B4.54 for the symbol key.

Table B4.56: SnpCleanFwd and SnpNotSharedDirtyFwd Snoopee state transitions with Clean and Dirty tags

Snoopee cache state				Response to Requester	Response to Home	TagOp value in response to Home		
Initial	**Final expected**	**Final permitted**	**RetToSrc**			**Initial start state of Dirty[a]**	**Initial start state of Dirty**	**Initial start state of Invalid or Clean**
I	I	-	X[b]	No Fwd	SnpResp_I	-	-	-
UC	SC	I	0	CompData_SC	SnpResp_SC_Fwded_SC	-	-	-
			1	CompData_SC	SnpRespData_SC_Fwded_SC	-	-	I, Transfer
	I	-	0	CompData_SC	SnpResp_I_Fwded_SC	-	-	-
			1	CompData_SC	SnpRespData_I_Fwded_SC	-	-	I, Transfer
UCE	I	-	X[b]	No Fwd	SnpResp_I	-	-	-
UD	SD[c]	-	0	CompData_SC	SnpResp_SD_Fwded_SC	P	-	-
			1	CompData_SC	SnpRespData_SD_Fwded_SC	P	Transfer	I, Transfer
	SC	I	X[b]	CompData_SC	SnpRespData_SC_PD_Fwded_SC	P	Update	I, Transfer
	I	-	X[b]	CompData_SC	SnpRespData_I_PD_Fwded_SC	P	Update	I, Transfer
UDP	I	-	X[b]	No Fwd	SnpRespDataPtl_I_PD	-	-	I
SC	SC	I	0	CompData_SC	SnpResp_SC_Fwded_SC	-	-	-
			1	CompData_SC	SnpRespData_SC_Fwded_SC	-	-	I, Transfer
	I	-	0	CompData_SC	SnpResp_I_Fwded_SC	-	-	-
			1	CompData_SC	SnpRespData_I_Fwded_SC	-	-	I, Transfer
SD	SD[c]	-	0	CompData_SC	SnpResp_SD_Fwded_SC	P	-	-
			1	CompData_SC	SnpRespData_SD_Fwded_SC	P	Transfer	I, Transfer
	SC	I	X[b]	CompData_SC	SnpRespData_SC_PD_Fwded_SC	P	Update	I, Transfer
	I	-	X[b]	CompData_SC	SnpRespData_I_PD_Fwded_SC	P	Update	I, Transfer

[a] Indicates if the transition itself is permitted or not. This column is only relevant when the tag initial state is Dirty.

[b] The protocol requirements apply for both states of RetToSrc.

[c] This state transition is not permitted if DoNotGoToSD is asserted.

B4.8.3.3 SnpSharedFwd

The rules, in addition to the common rules, to be followed by a Snoopee that receives a SnpSharedFwd are:

- Snoopee is permitted to forward the cache line in either SD or SC state.
- Snoopee must transition to either SD, SC, or I state.
- For behavior related to the RetToSrc bit see B4.9 *Returning Data with Snoop response*.

Table B4.57 shows the Snoopee cache state transition and the required Snoop responses.

See Table B4.54 for the symbol key.

Table B4.57: SnpCleanFwd and SnpNotSharedDirtyFwd Snoopee state transitions with Clean and Dirty tags

Snoopee cache state				Response to Requester	Response to Home	TagOp value in response to Home		
Initial	Final expected	Final permitted	RetToSrc			Initial start state of Dirty[a]	Initial start state of Dirty	Initial start state of Invalid or Clean
I	I	-	X[b]	No Fwd	SnpResp_I	-	-	-
UC	SC	I	0	CompData_SC	SnpResp_SC_Fwded_SC	-	-	-
			1	CompData_SC	SnpRespData_SC_Fwded_SC	-	-	I,Transfer
	I	-	0	CompData_SC	SnpResp_I_Fwded_SC	-	-	-
			1	CompData_SC	SnpRespData_I_Fwded_SC	-	-	I,Transfer
UCE	I	-	X[b]	No Fwd	SnpResp_I	-	-	-
UD	SD[c]	-	0	CompData_SC	SnpResp_SD_Fwded_SC	P	-	-
			1	CompData_SC	SnpRespData_SD_Fwded_SC	P	Transfer	I,Transfer
	SC	I	0	CompData_SD_PD	SnpResp_SC_Fwded_SD_PD	NP	-	-
			1	CompData_SD_PD	SnpRespData_SC_Fwded_SD_PD	NP	-	I,Transfer
			X[b]	CompData_SC	SnpRespData_SC_PD_Fwded_SC	P	Update	I,Transfer
	I	-	0	CompData_SD_PD	SnpResp_I_Fwded_SD_PD	NP	-	-
			1	CompData_SD_PD	SnpRespData_I_Fwded_SD_PD	NP	-	I,Transfer
			X[b]	CompData_SC	SnpRespData_I_PD_Fwded_SC	P	Update	I,Transfer
UDP	I	-	X[b]	No Fwd	SnpRespDataPtl_I_PD	-	-	I
SC	SC	I	0	CompData_SC	SnpResp_SC_Fwded_SC	-	-	-
			1	CompData_SC	SnpRespData_SC_Fwded_SC	-	-	I,Transfer
	I	-	0	CompData_SC	SnpResp_I_Fwded_SC	-	-	-
			1	CompData_SC	SnpRespData_I_Fwded_SC	-	-	I,Transfer
SD	SD[b]	-	0	CompData_SC	SnpResp_SD_Fwded_SC	P	-	-
			1	CompData_SC	SnpRespData_SD_Fwded_SC	P	Transfer	I,Transfer
	SC	I	0	CompData_SD_PD	SnpResp_SC_Fwded_SD_PD	NP	-	-
			1	CompData_SD_PD	SnpRespData_SC_Fwded_SD_PD	NP	-	I,Transfer
			X[b]	CompData_SC	SnpRespData_SC_PD_Fwded_SC	P	Update	I,Transfer
	I	-	0	CompData_SD_PD	SnpResp_I_Fwded_SD_PD	NP	-	-
			1	CompData_SD_PD	SnpRespData_I_Fwded_SD_PD	NP	-	I,Transfer
			X[b]	CompData_SC	SnpRespData_I_PD_Fwded_SC	P	Update	I,Transfer

[a] Indicates if the transition itself is permitted or not. This column is only relevant when the tag initial state is Dirty.

[b] The protocol requirements apply for both states of RetToSrc.

[c] This state transition is not permitted if DoNotGoToSD is asserted.

B4.8.3.4 SnpUniqueFwd

Use of the SnpUniqueFwd snoop is only permitted if the cache line is cached at a single RN-F:

- Home is permitted to send the SnpUniqueFwd snoop to an RN-F in Shared state if Home determines that the Invalidating snoop needs to be sent to only one cache.

The rules, in addition to the common rules, to be followed by a Snoopee that receives a SnpUniqueFwd are:

- Snoopee must forward the cache line in Unique state.
- Snoopee that has the cache line in Dirty state must Pass Dirty to the Requester not to Home.
- Snoopee must transition to I state.
- Snoopee must not return data to Home.

RetToSrc bit in the snoop must be set to 0.

Table B4.58 shows the Snoopee cache state transitions and the required Snoop responses.

See Table B4.54 for the symbol key.

Table B4.58: SnpUniqueFwd Snoopee state transitions with Clean and Dirty tags

Snoopee cache state				Response to Requester	Response to Home	TagOp value in response to Home		
Initial	Final expected	Final permitted	RetToSrc			Initial start state of Dirty[a]	Initial start state of Dirty	Initial start state of Invalid or Clean
I	I	-	0	No Fwd	SnpResp_I	-	-	-
UC	I	-	0	CompData_UC	SnpResp_I_Fwded_UC	-	-	-
UCE	I	-	0	No Fwd	SnpResp_I	-	-	-
UD	I	-	0	CompData_UD_PD	SnpResp_I_Fwded_UD_PD	NP[b]	-	-
				No Fwd	SnpRespData_I_PD	P	Update	I, Transfer
UDP	I	-	0	No Fwd	SnpRespDataPtl_I_PD	-	-	I
SC	I	-	0	CompData_UC	SnpResp_I_Fwded_UC	-	-	-
SD	I	-	0	CompData_UD_PD	SnpResp_I_Fwded_UD_PD	NP[b]	-	-
				No Fwd	SnpRespData_I_PD	P	Update	I, Transfer

[a] Indicates if the transition itself is permitted or not. This column is only relevant when the tag initial state is Dirty.

[b] This transition is not permitted because Dirty tags are lost. The Dirty tags are lost because the Snoop response to the Home does not include data with which to pass the Dirty tags to the Home.

B4.8.3.5 SnpPreferUnique and SnpPreferUniqueFwd

Use of the SnpPreferUniqueFwd snoop is only permitted if the cache line is cached at a single RN-F.

The rules, in addition to the common rules, to be followed by a Snoopee that receives a SnpPreferUniqueFwd and does not treat SnpPreferUniqueFwd as the Non-forwarding snoop are:

- When the Snoopee is in the process of executing an Exclusive access sequence, using the same address:
  - Snoopee must forward the cache line in SC state.
  - Snoopee must transition to either SD or SC state.
  - Snoopee must not transition to I state, except when in UCE or UDP state.
  - For behavior related to the RetToSrc bit see B4.9 *Returning Data with Snoop response* related to SnpNotSharedDirtyFwd.
  - A Snoopee that is not in the middle of executing an Exclusive access sequence is permitted, but not expected, to treat the snoop as a Non-invalidating snoop.
- When the Snoopee is not in the process of executing an Exclusive access sequence, using the same address, and treats the snoop as an Invalidating snoop:
  - Snoopee must forward the cache line in Unique state.

- Snoopee that has the cache line in Dirty state must Pass Dirty to the Requester not to Home.
- Snoopee must transition to I state.
- Snoopee must not return data to Home.
- RetToSrc bit value in the snoop is ignored and treated as 0.

For SnpPreferUnique and SnpPreferUniqueFwd, it is IMPLEMENTATION DEFINED when an agent is in the process of executing an exclusive sequence. To determine if a Snoopee treats SnpPreferUnique as an invalidating snoop or a non-invalidating snoop, the Snoop response must be inspected. It is protocol compliant for a Snoopee to always treat the snoop as non-invalidating.

Table B4.59 and Table B4.60 show the Snoopee cache state transitions and the required Snoop responses.

See Table B4.54 for the symbol key.

Table B4.59: State transitions for SnpPreferUniqueFwd when Snoopee is executing an exclusive sequence

Snoopee cache state				Response to Requester	Response to Home	TagOp value in response to Home		
Initial	Final expected	Final permitted	RetToSrc			Initial start state of Dirty[a]	Initial start state of Dirty	Initial start state of Invalid or Clean
I	I	-	X	No Fwd	SnpResp_I	-	-	-
UC	SC	-	0	CompData_SC	SnpResp_SC_Fwded_SC	-	-	-
			1	CompData_SC	SnpRespData_SC_Fwded_SC	-	-	I,Transfer
UCE	I	-	X	No Fwd	SnpResp_I	-	-	-
UD	SD*	-	0	CompData_SC	SnpResp_SD_Fwded_SC	P	-	-
			1	CompData_SC	SnpRespData_SD_Fwded_SC	P	Transfer	I,Transfer
	SC	-	X	CompData_SC	SnpRespData_SC_PD_Fwded_SC	P	Update	I,Transfer
UDP	I	-	X	No Fwd	SnpRespDataPtl_I_PD	-	-	I
SC	SC	-	0	CompData_SC	SnpResp_SC_Fwded_SC	-	-	-
			1	CompData_SC	SnpRespData_SC_Fwded_SC	-	-	I,Transfer
SD	SD*	-	0	CompData_SC	SnpResp_SD_Fwded_SC	P	-	-
			1	CompData_SC	SnpRespData_SD_Fwded_SC	P	Transfer	I,Transfer
	SC	-	X	CompData_SC	SnpRespData_SC_PD_Fwded_SC	P	Update	I,Transfer

[a] Indicates if the transition itself is permitted or not. This column is only relevant when the tag initial state is Dirty.

Table B4.60 shows the expected and permitted Snoopee cache state transitions and responses to the SnpPreferUniqueFwd Snoop request when the Snoopee is not executing an Exclusive sequence.

See Table B4.54 for the symbol key.

Table B4.60: State transitions for SnpPreferUniqueFwd when Snoopee is not executing an exclusive sequence

Initial Snoopee state	Final expected Snoopee state	Final permitted Snoopee state	RetToSrc	Response to Requester	Response to Home	TagOp value in response to Home		
						Initial start state of Dirty[a]	Initial start state of Dirty	Initial start state of Invalid or Clean
I	I	-	X	No Fwd	SnpResp_I	-	-	-
UC	I	-	X	CompData_UC	SnpResp_I_Fwded_UC	-	-	-
UCE	I	-	X	No Fwd	SnpResp_I	-	-	-
UD	I	-	X	CompData_UD_PD	SnpResp_I_Fwded_UD_PD	NP	-	-
			X	No Fwd	SnpRespData_I_PD	P	Update	I,Transfer
UDP	I	-	X	No Fwd	SnpRespDataPtl_I_PD	-	-	I
SC	I	-	X	CompData_UC	SnpResp_I_Fwded_UC	-	-	-
SD	I	-	X	CompData_UD_PD	SnpResp_I_Fwded_UD_PD	NP	-	-
			X	No Fwd	SnpRespData_I_PD	P	Update	I,Transfer

[a] Indicates if the transition itself is permitted or not. This column is only relevant when the tag initial state is Dirty.

B4.9 Returning Data with Snoop response

The rules for returning a copy of the cache line with the Snoop response are detailed below.

For Non-forwarding snoops, except SnpMakeInvalid, the rules for returning a copy of the cache line to the Home are:

- Irrespective of the value of RetToSrc, must return a copy if the cache line is Dirty.
- Irrespective of the value of RetToSrc, optionally can return a copy if the cache line is Unique Clean.
- If the RetToSrc value is 1, must return a copy if the cache line is Shared Clean.
- If the RetToSrc value is 0, must not return a copy if the cache line is Shared Clean.

For Forwarding snoops where data is being forwarded, the rules for returning a copy of the cache line to the Home are:

- Irrespective of the value of RetToSrc, must return a copy if a Dirty cache line cannot be forwarded or kept.
- If the RetToSrc value is 1, must return a copy if the cache line is Dirty or Clean.
- If the RetToSrc value is 0, must not return a copy if the cache line is Clean.

RetToSrc is inapplicable and must be set to 0 in:

- SnpCleanShared, SnpCleanInvalid, and SnpMakeInvalid
- SnpOnceFwd and SnpUniqueFwd
- SnpMakeInvalidStash, SnpUniqueStash, SnpStashUnique, and SnpStashShared
- SnpQuery

RetToSrc is applicable and can take any value in all other snoops except SnpDVMOp.

RetToSrc is inapplicable and must be set to 0 in SnpDVMOp.

Home must only set RetToSrc on the Snoop request to a single Request Node.

B4.10 Do not transition to SD

Do not transition to SD (DoNotGoToSD) is a field that modifies Non-invalidating snoops.

DoNotGoToSD specifies when a Snoopee must not transition to SD state as a result of the Snoop request.

> **Note**
>
> A non-forced change or silent change from UD to SD is permitted irrespective of the value of DoNotGoToSD.
>
> Any forced change from Unique to Shared must obey DoNotGoToSD.

See B13.10.38 *Do not transition to SD state, DoNotGoToSD* for the field applicability and field value encoding.

B4.11 Hazard conditions

This section lists the responsibilities of the RN-F and HN-F to handle address hazards and race conditions among Snoopable transactions. Ordering among Non-snoopable transactions and among Snoopable transactions is described in B2.6 *Ordering*.

In addition to many Requesters issuing transactions at the same time, the protocol also permits each Requester to make multiple outstanding requests, and to receive multiple outstanding snoop requests. The interconnect, that is, ICN(HN-F, HN-I and MN), is responsible to ensure that there is a defined order in which transactions to the same cache line can occur, and that the defined order is the same for all components.

B4.11.1 At the RN-F node

An RN-F node must respond to received Snoop requests, except for SnpDVMOp(Sync), in a timely manner without creating any Protocol layer dependency on completion of outstanding requests.

If a pending request to the same cache line is present at the RN-F and the pending request has not received any response packets:

- The Snoop request must be processed normally.
- The cache state must transition as applicable for each Snoop request type.
- The cached data or CopyBack request data must be returned with the Snoop response, or forwarded to the Requester, if required by the Snoop request type, Snoop request attributes, and cache state.

If a pending request to the same cache line is present at the RN-F and the pending request has received at least one Data response packet or a RespSepData response:

- The RN-F must wait to receive all Data response packets before responding to the Snoop request.
- Once all the Data response packets are received by RN-F:
  - The Snoop request must be processed normally.
  - The cache state must transition as applicable for each Snoop request type.
  - The cached data must be returned with the Snoop response, or forwarded to the Requester, if required by the Snoop request type, Snoop request attributes, and cache state.

If the pending request is a CopyBack request, the following additional requirements apply:

- Request transaction flow must be completed after receiving the CompDBIDResp or Comp response.
- The cache state in the CopyBackWriteData or CompAck response must be the state of the cache line after the snoop request is processed, not the state at the time of sending the CopyBack request.
- If the cache line state is I after the Snoop response is sent, the cache state in the CopyBackWriteData or CompAck response must be I. Additionally, for a CopyBackWriteData response, all BE bits must be deasserted, and the corresponding data must be set to 0.
- If the cache line state is UC or SC after the Snoop response is sent, a Request Node is permitted to not send valid CopyBack Data. If the Request Node decides not to send valid CopyBack Data, the cache state in the CopyBackWriteData or CompAck response must be I. Additionally, for a CopyBackWriteData response, all BE bits must be deasserted, and the corresponding data must be set to 0.
- If data is included with CopyBackWriteData, it can be:
  - The same data that has been sent with the Snoop response.
  - More up to date data than sent with the snoop response. This is only possible when the snoop is SnpOnce, SnpOnceFwd, or SnpCleanShared, and the snoop response indicates the cache line can be further modified.

– Older data than was sent with the snoop response. This is only possible when the CopyBack write is a WriteClean, the snoop request is SnpOnce*, and the response indicates that the cache line can be further modified.

For non-ordered transactions, the Request Node can send CompAck without waiting for DataSepResp. For ordered transactions, the Request Node can send CompAck as soon as the first packet of DataSepResp is received. In both cases, a Request Node must not respond to a Snoop request before receiving all data packets.

The RN-F could receive multiple snoop requests before a response is received for a pending CopyBack request for the same cache line. In this instance, the data response carries the cache line state after the completion of the response to the last snoop request. Such a scenario is possible because the CopyBack request can be queued behind multiple Read and Dataless requests at the HN-F.

B4.11.2 At the ICN(HN-F) node

An HN-F orders transactions to the same cache line by sequencing transaction responses and Snoop transactions to the Requesters. As the interconnect is not required to be ordered, the arrival order of these messages, in certain cases, could not be the same as the order in which they were issued at the HN-F.

If a Response message that includes data requires multiple packets or beats of transfers over the interconnect, receiving or sending the message by Home implies sending or receiving all the packets corresponding to that message. That is, when a Home starts sending the message, all packets of the message must be sent without dependence on completion of any other request or response message.

Similarly, when a Home accepts part of the Data message, the remaining packets of that message must be accepted without any dependence on forward progress of any other request or response message.

When a subsequent forwarding of data depends upon receiving a Data message, the forwarding of data action can occur after receiving the first Data packet. A subsequent Non-data forwarding action that is processing of a subsequent request at Home as a consequence of sending or receiving of data by Home, must wait until all data is sent or received.

While a Snoop transaction response is pending, the only transaction responses that are permitted to be sent to the same address are:

- RetryAck for a CopyBack
- RetryAck and DBIDResp for a WriteUnique and Atomics
- RetryAck and, if applicable, a ReadReceipt for a Read request type
- RetryAck for a Dataless request type

Once a completion is sent for a transaction, the HN-F must not send a Snoop request to the same cache line until receiving:

- A CompAck for any Read and Dataless requests except for ReadOnce* and ReadNoSnp.
- A CompAck response for a CopyBack request that the HN-F has requested completes without a data transfer.
- A CopyBackWriteData response for a CopyBack request that the HN-F has requested completes with a data transfer.
- A NonCopyBackWriteData response for an Atomic request.
- A WriteData response, and if applicable, CompAck, for a WriteUnique request where the HN-F has not used a DWT flow.
- A Comp response from the Subordinate Node for a WriteUnique request where the HN-F has used a DWT flow.

Chapter B5
Interconnect Protocol Flows

This chapter shows interconnect protocol flows for different transaction types, and interconnect hazard conditions. The protocol flows are illustrated using Time-Space diagrams. It contains the following sections:

See Figure 2 for details of the conventions used to illustrate protocol flow.

In the transaction flow diagrams that follow:

- There are multiple coherent Request Nodes, an HN-F, and an SN-F.
- If the HN-F receives multiple data responses, that is, one response from a snooped RN-F and another from an SN-F, the data being forwarded to the Requester is highlighted in bold.
- There is no interconnect cache at the HN-F. This results in all requests to the HN-F initiating a request to the SN-F.

B5.1 Read transaction flows

This section gives examples of the interconnect protocol flow for Read transactions.

B5.1.1 Read transactions with DMT and without snoops

For Read transactions without snoops, it is recommended to use DMT.

Figure B5.1 shows an example DMT transaction flow using the ReadShared transaction.

In Figure B5.1, a response from SN-F to HN-F is not required because CompAck from the Requester is used to deallocate the request at Home.

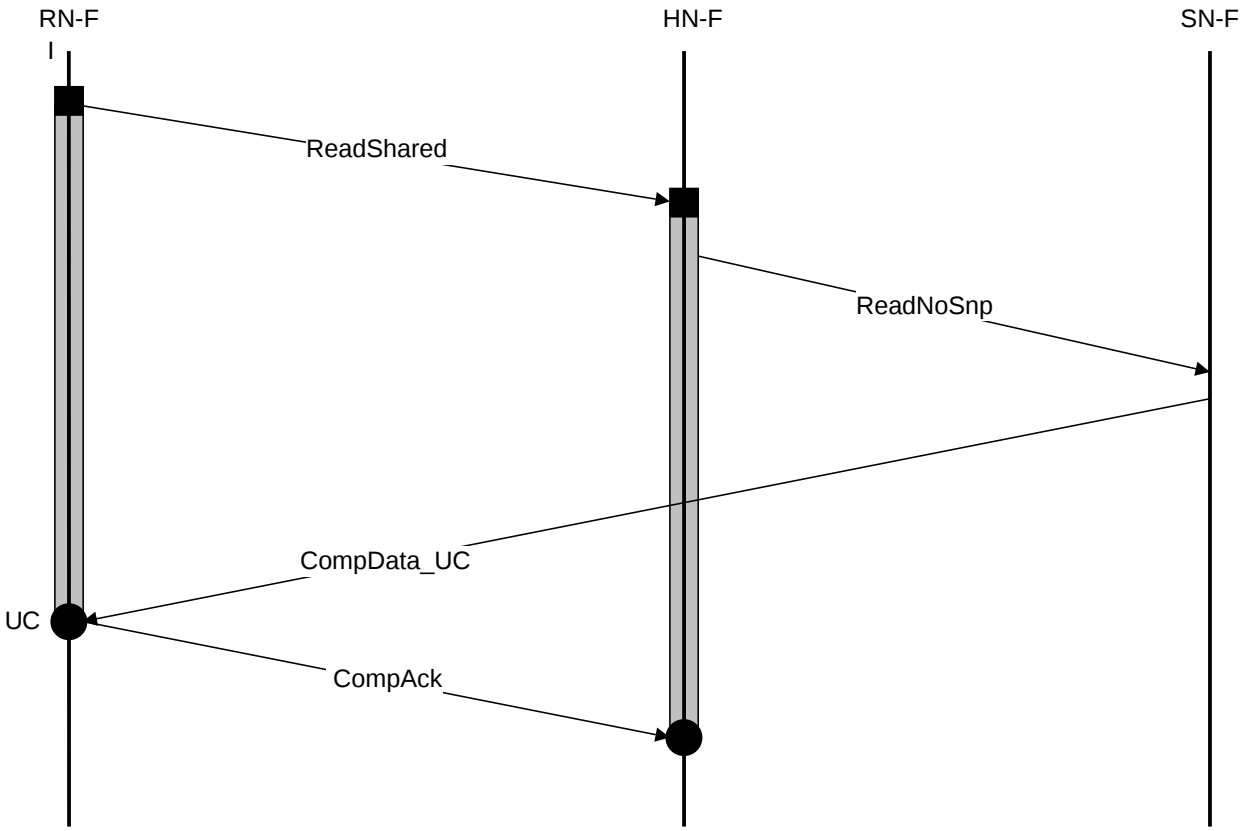

Figure B5.1: DMT Read transaction example without snoops

The steps in the ReadShared transaction flow in Figure B5.1 are:

1. RN-F sends a ReadShared request to HN-F.
2. HN-F sends a ReadNoSnp request to SN-F.
3. SN-F sends a data response directly to RN-F, using CompData_UC.
4. RN-F sends CompAck to HN-F as the request is ReadShared and requires CompAck to complete the transaction.

B5.1.2 Read transaction with DMT and with snoops

For Read transactions with snoops and data from memory, DMT is recommended.

Figure B5.2 shows an example DMT transaction flow using the ReadShared transaction.

From SN-F to HN-F, a response is not required because CompAck from the Requester is used to deallocate the request at Home.

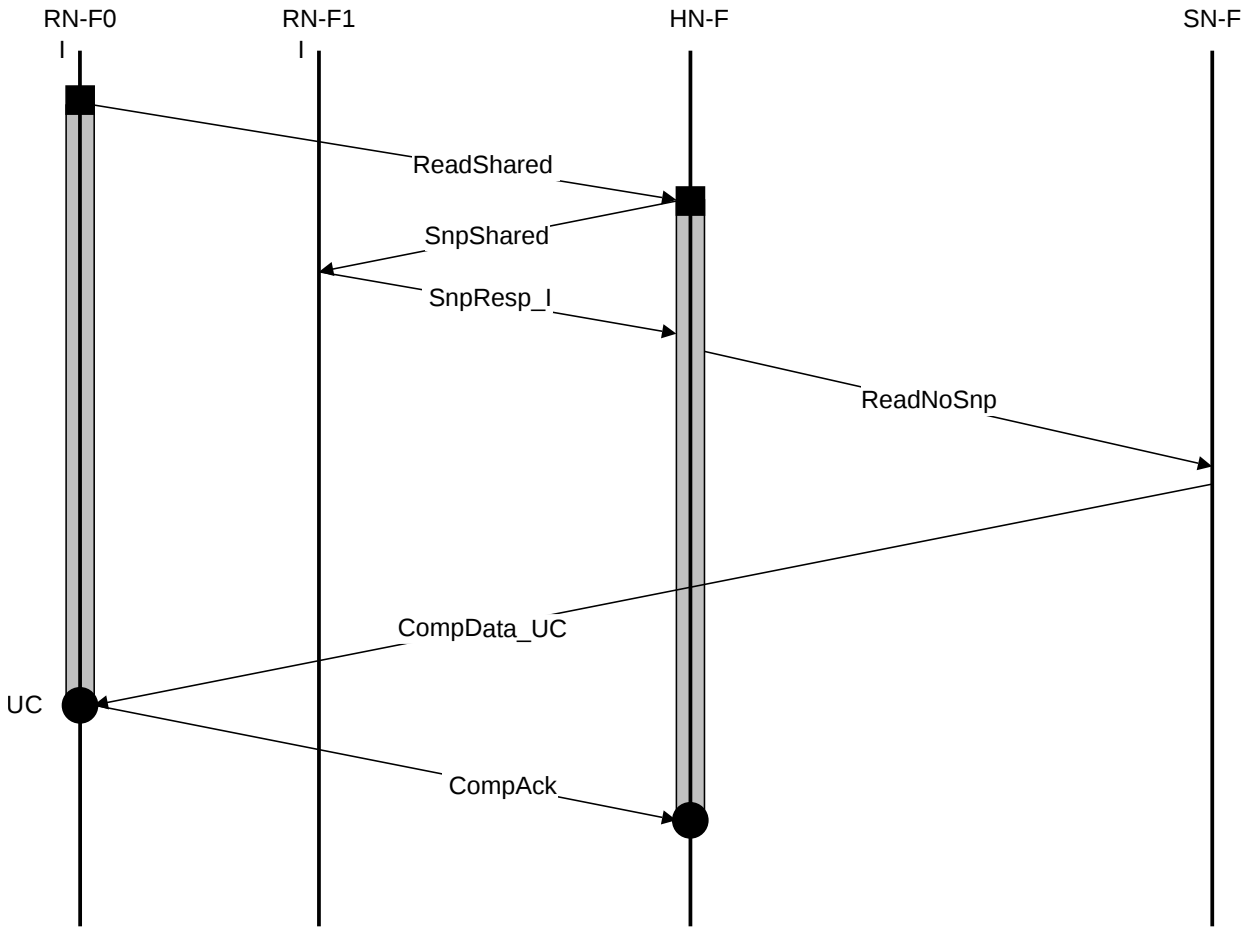

Figure B5.2: DMT Read transaction example with snoops and data from memory

The steps in the ReadShared transaction flow in Figure B5.2 are:

1. RN-F0 sends a ReadShared request to HN-F.
2. HN-F sends a SnpShared request to RN-F1. RN-F1 returns SnpResp_I response to HN-F.
3. HN-F sends a ReadNoSnp request to SN-F after receiving the Snoop response from RN-F1. This guarantees that RN-F1 has not responded with data.
4. SN-F sends a data response directly to RN-F0, using CompData_UC.
5. RN-F0 sends CompAck to HN-F as the request is ReadShared and requires CompAck to complete the transaction.

B5.1.3 Read transaction with DCT

For Read transactions with snoops and data from other Request Node caches, Direct Cache Transfer (DCT) is recommended.

B5.1.3.1 DCT from cache line in UC state

Figure B5.3 shows an example flow for a DCT transaction. The Requester is RN-F0 and the forwarding cache is at RN-F1.

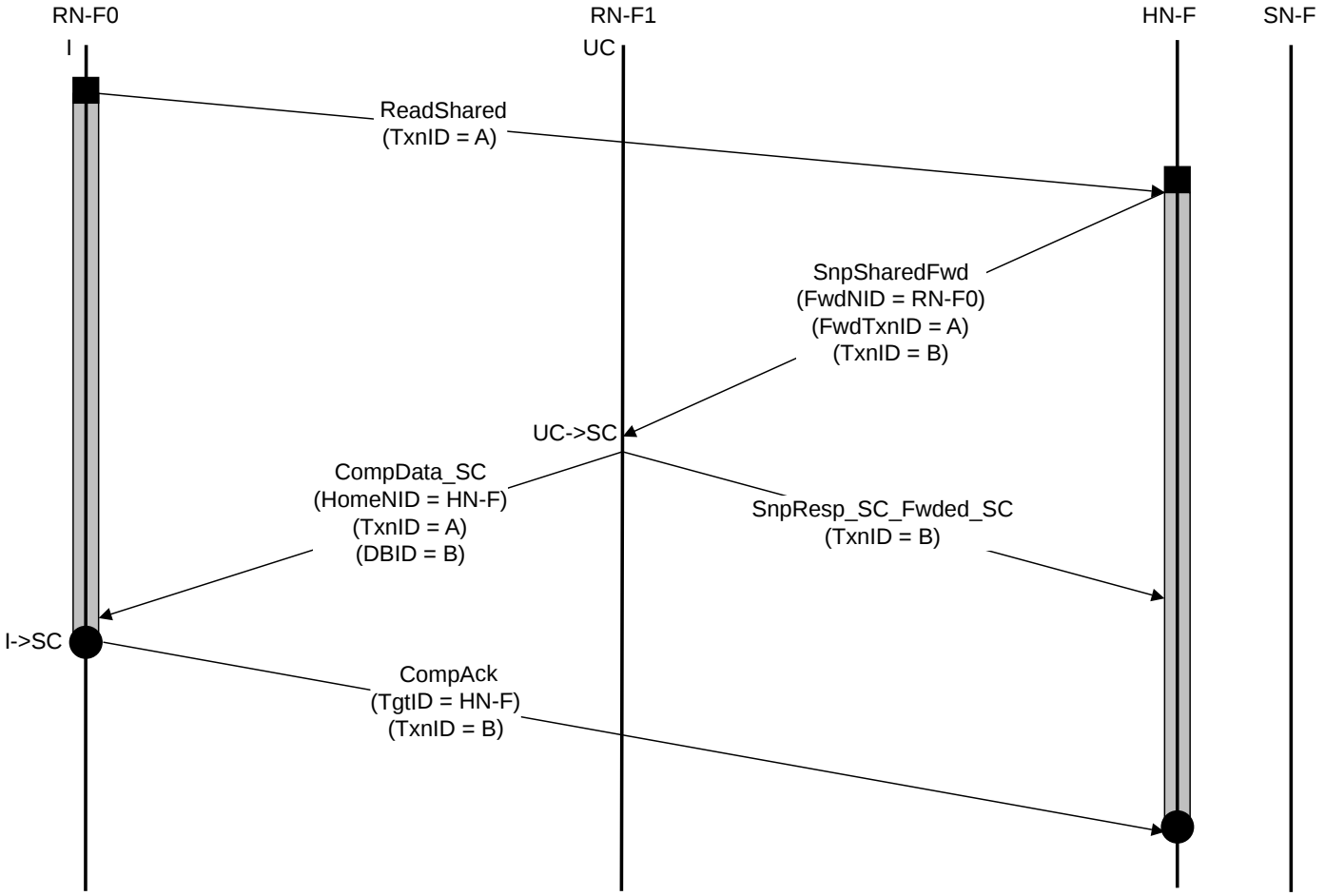

Figure B5.3: DCT from cache line in UC state

The steps in the DCT transaction flow in Figure B5.3 are:

1. RN-F0 sends a ReadShared request to HN-F.
2. HN-F sends a SnpSharedFwd, a Forwarding snoop request to RN-F1.
3. RN-F1 forwards CompData_SC response to RN-F0. The TxnID is the same as the original ReadShared request.
4. RN-F1 also sends a SnpResp_SC_Fwded_SC snoop response to HN-F. RN-F1 cache line state transitions from UC to SC.
5. After receiving the CompData_SC response, RN-F0 cache line state transitions from I to SC and RN-F0 sends a CompAck response to HN-F.

Note

Steps 3 and 4 in the DCT transaction flow can occur in any order as CompData and SnpResp are sent on different channels.

B5.1.3.2 Double data return in a DCT transaction

Figure B5.4 shows an example DCT transaction flow that sends the data to HN-F and forwards the data to the RN-F0.

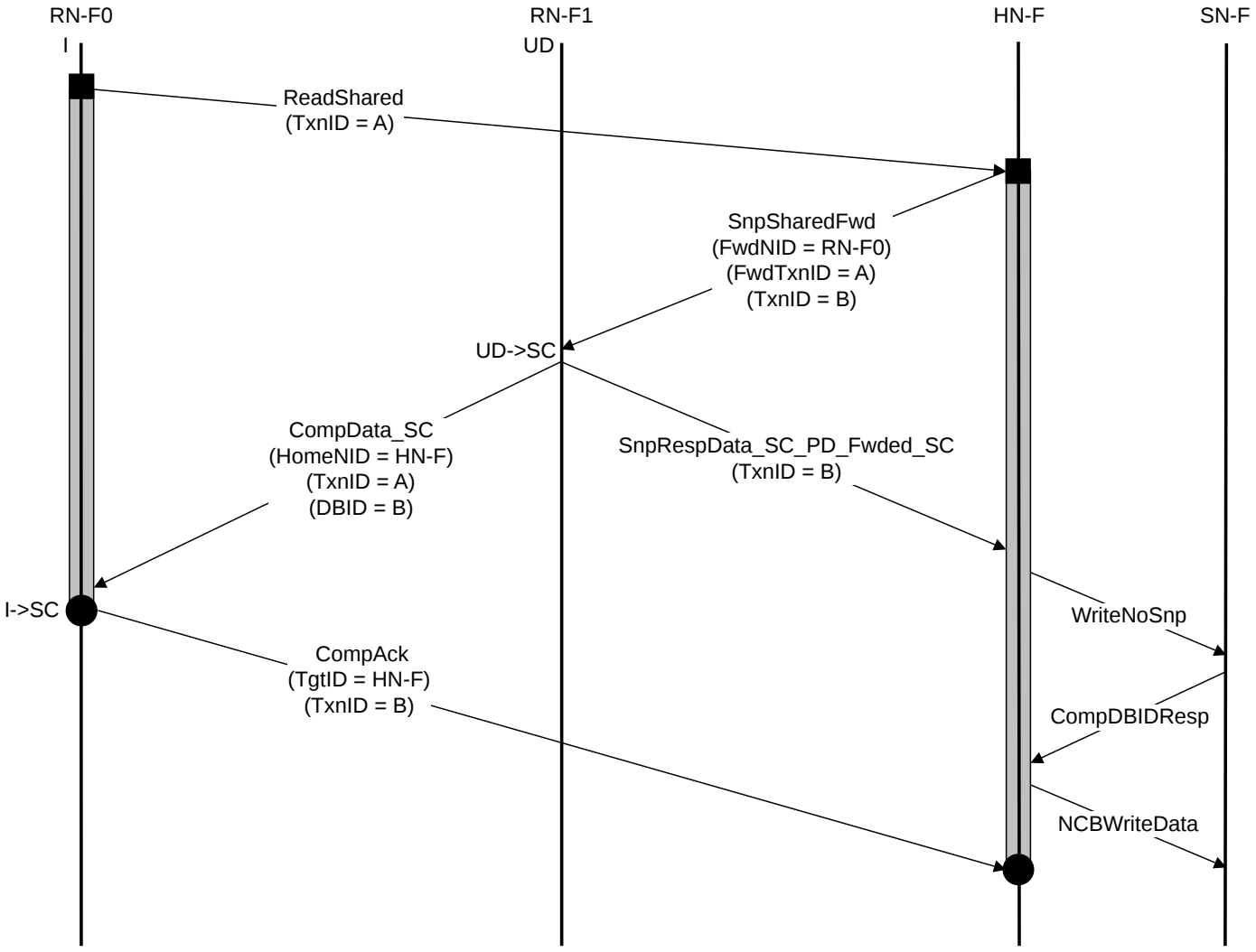

Figure B5.4: Double data return in DCT transaction

The steps in the DCT transaction flow in Figure B5.4 are:

1. RN-F0 sends a ReadShared request to HN-F.
2. HN-F sends a SnpSharedFwd Snoop request to RN-F1.
3. RN-F1 sends CompData_SC response to RN-F0. The TxnID is the same as the original ReadShared request. RN-F0 cache line transitions from I to SC.

4. RN-F1 also sends a SnpRespData_SC_PD_Fwded_SC snoop response to HN-F that includes a copy of the cache line and passes responsibility for the Dirty cache line to HN-F. RN-F1 cache line transitions from UD to SC.
5. HN-F sends a WriteNoSnp request to the SN-F.
6. SN-F responds to the HN-F with CompDBIDResp to request the data.
7. HN-F sends NCBWrData to the SN-F.
8. RN-F0 sends CompAck after receiving the Data response.

B5.1.4 Read transaction without DMT or DCT

Figure B5.5 shows an example of the flow without DMT using the ReadNoSnp transaction. In Figure B5.5, the ReadNoSnp has the ExpCompAck set in the original request.

The request does not generate any snoops and receives the data from a response to a memory read by the HN-F.

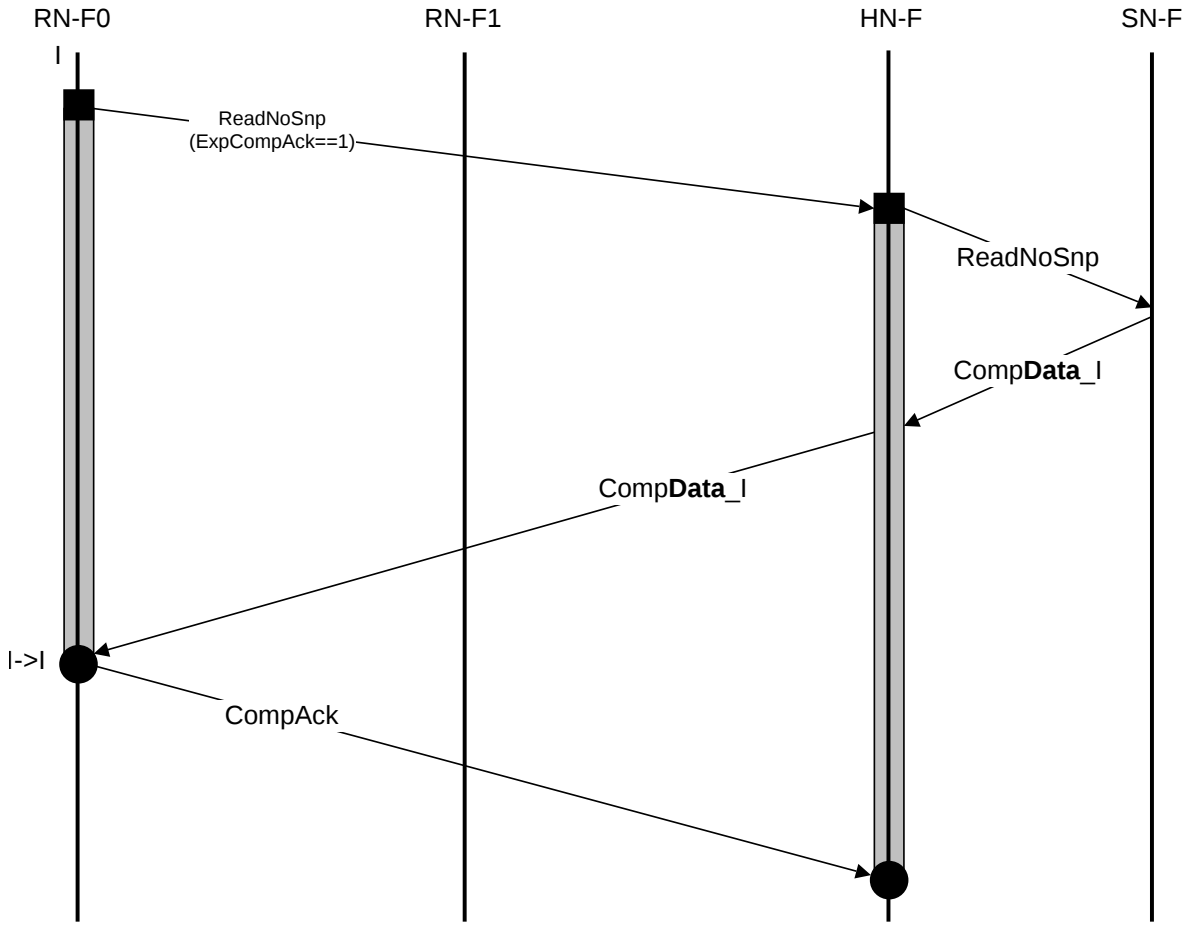

Figure B5.5: ReadNoSnp transaction flow

The steps in the ReadNoSnp transaction flow in Figure B5.5 are:

1. RN-F0 issues a ReadNoSnp transaction, with ExpCompAck set to 1.
2. HN-F receives and allocates the request.

> **Note**
>
> HN-F does not send snoops as the request is recognized as a Non-snoopable request type.

3. HN-F sends a ReadNoSnp to SN-F.
4. SN-F returns CompData_I to HN-F.
5. HN-F in turn returns the data to RN-F0.

> **Note**
>
> If ExpCompAck had not been asserted in the original ReadNoSnp request, the HN-F could have deallocated the request at this point.

6. RN-F0 deallocates the request and sends CompAck towards the HN-F.
7. HN-F receives the CompAck response and deallocates the request.

The copy of data being transferred is marked in bold in Figure B5.5.

B5.1.5 Read transaction with snoop response with partial data and no memory update

An example of this type of flow is a ReadUnique transaction.

Figure B5.6 shows the transaction flow, the copy of data being transferred is marked in bold.

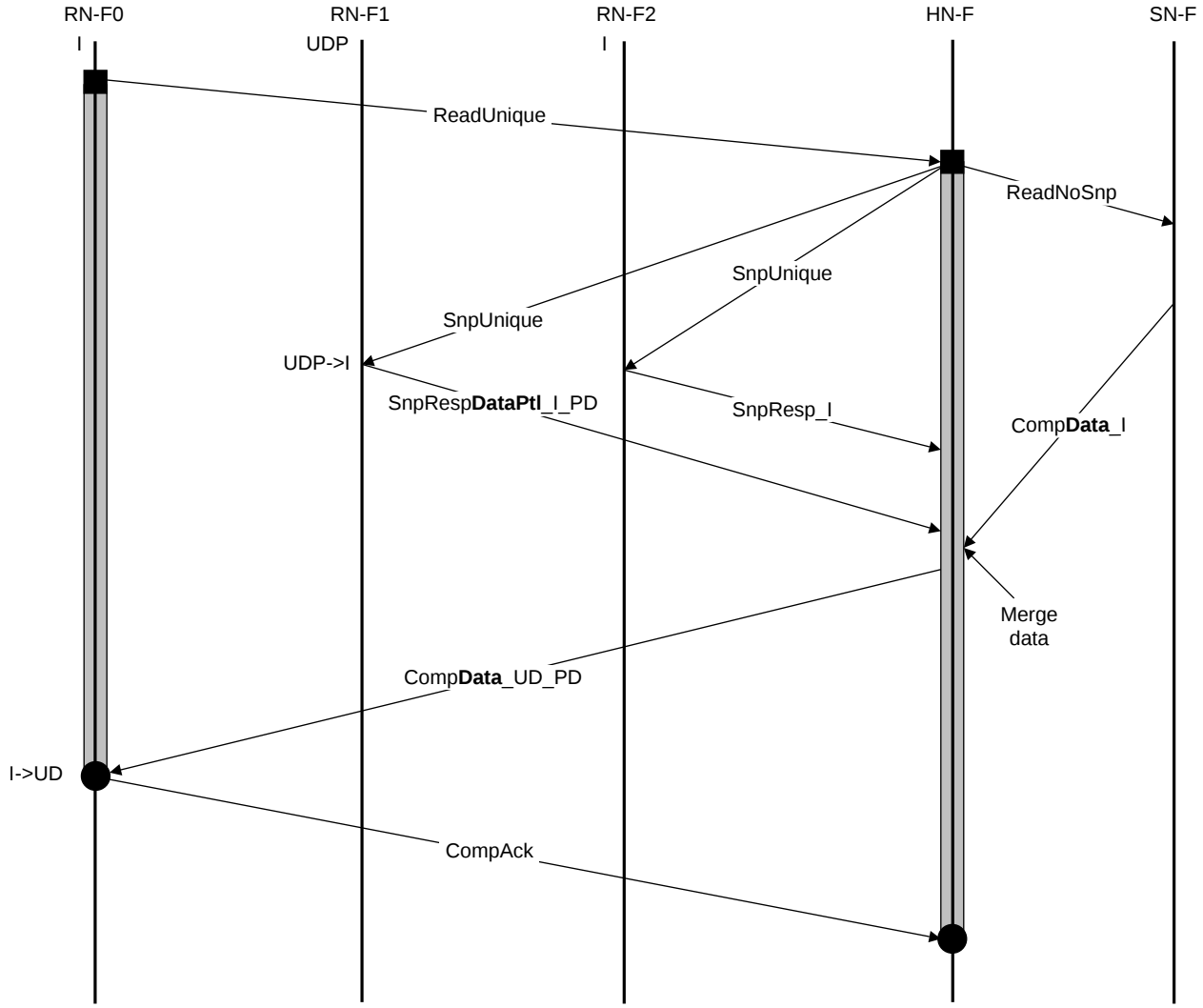

Figure B5.6: ReadUnique with partial data snoop response

The steps in the ReadUnique with partial data snoop response transaction flow in Figure B5.6 are:

1. RN-F0 sends ReadUnique request to HN-F.
2. HN-F sends ReadNoSnp request to SN-F and SnpUnique requests to RN-F1 and RN-F2.
3. RN-F1 transitions the cache line from UDP to I and returns SnpRespDataPtl_I_PD to HN-F. RN-F2 returns SnpResp_I to the HN-F. Meanwhile, the SN-F returns CompData response to HN-F. HN-F merges the data returned from the SN-F with the partial data received from the RN-F1.
4. HN-F sends CompData_UD_PD to RN-F0. RN-F0 cache line state transitions from I to UD.
5. RN-F0 issues CompAck response to the HN-F to indicate transaction completion.

B5.1.6 Read transaction with snoop response with partial data and memory update

An example of this type of flow is a ReadClean transaction.

Figure B5.7 shows the transaction flow, the copy of data being transferred is marked in bold.

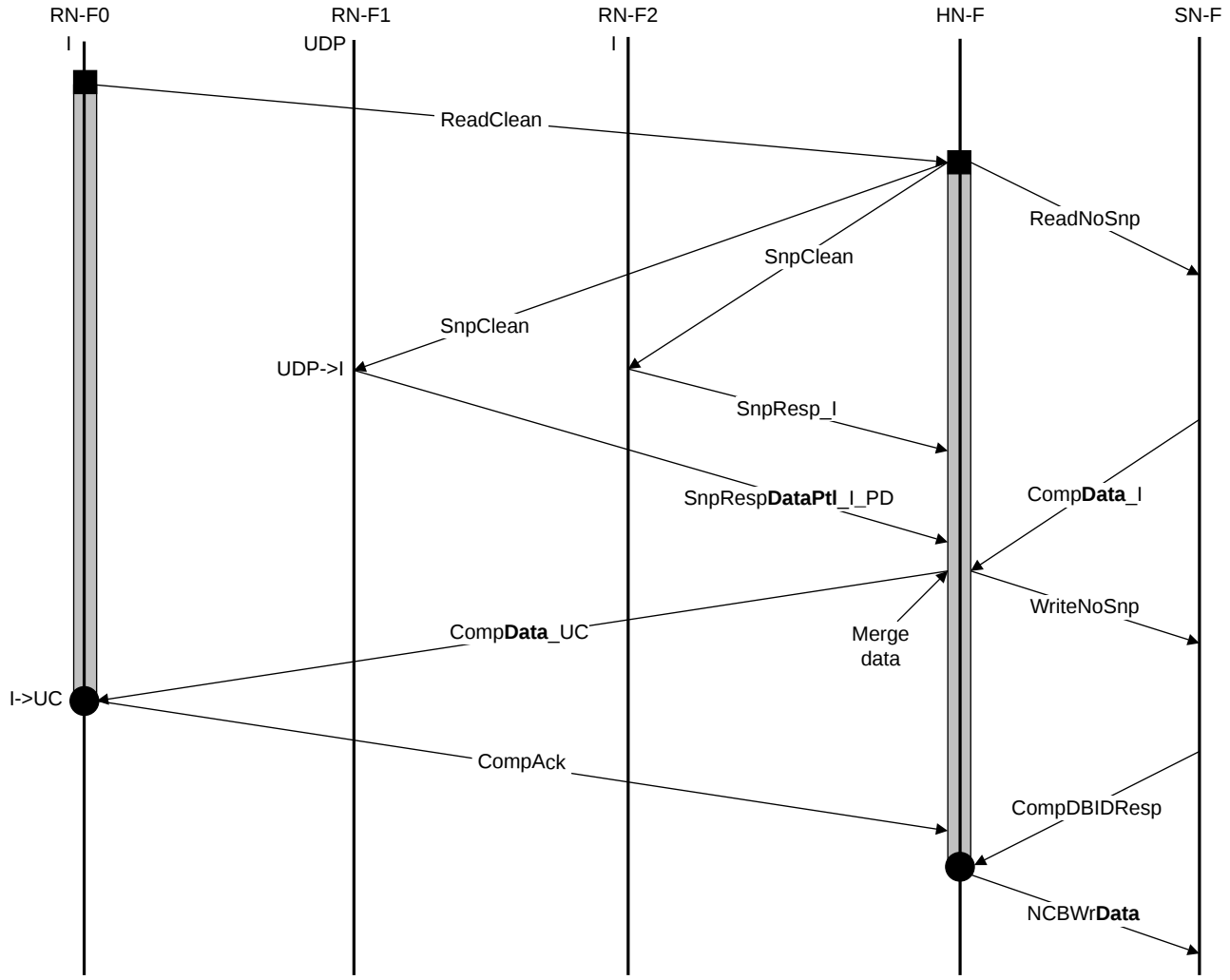

Figure B5.7: ReadClean with partial data snoop response

The steps in the ReadClean with partial data snoop response transaction flow in Figure B5.7 are:

1. RN-F0 sends ReadClean request to HN-F.
2. HN-F sends ReadNoSnp request to SN-F and SnpClean requests to RN-F1 and RN-F2. RN-F1 cache line state transitions from UDP to I.
3. RN-F1 returns SnpRespDataPtl_I_PD and RN-F2 returns SnpResp_I to the HN-F. Meanwhile, the SN-F returns CompData_I response to HN-F. This merges the data.
4. HN-F sends CompData_UC to RN-F0 and WriteNoSnp to SN-F. RN-F0 cache line state transitions from I to UC.
5. RN-F0 issues CompAck response to the HN-F to indicate transaction completion.
6. SN-F issues CompDBIDResp to HN-F.
7. HN-F sends NCBWrData to SN-F.

B5.1.7 ReadOnce* and ReadNoSnp with early Home deallocation

Figure B5.8 shows the optimized flow for an unordered ReadOnce request.

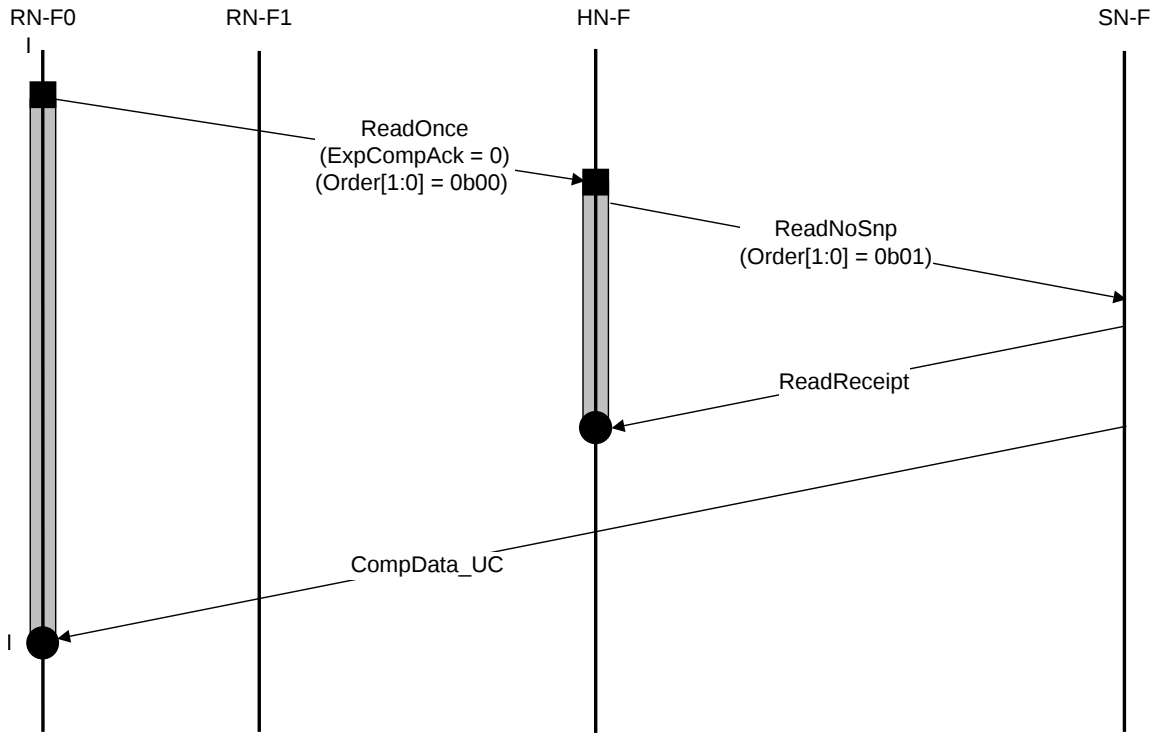

Figure B5.8: DMT optimization for unordered ReadOnce

The steps in the optimized ReadOnce transaction flow in Figure B5.8 are:

1. RN-F0 sends an unordered ReadOnce request to HN-F with Order[1:0] set to `0b00`.
2. HN-F sends a DMT ReadNoSnp request to SN-F with the Order[1:0] set to `0b01`.
3. SN-F sends ReadReceipt to Home.
4. HN-F deallocates the request after receiving the ReadReceipt response.
5. SN-F sends CompData_UC directly to RN-F0.

Note

Use of a ReadNoSnp transaction from Home to Subordinate, in the case where CompAck is not required, avoids the need to send a RespSepData response from Home to Requester.

B5.1.8 ReadNoSnp transaction with DMT and separate Non-data and Data-only response

Figure B5.9 shows an example DMT transaction flow with separate Non-data and Data-only response.

In this example, there is no ordering requirement and HN-F can deallocate the request once it receives ReadReceipt, without waiting for CompAck from the RN-F.

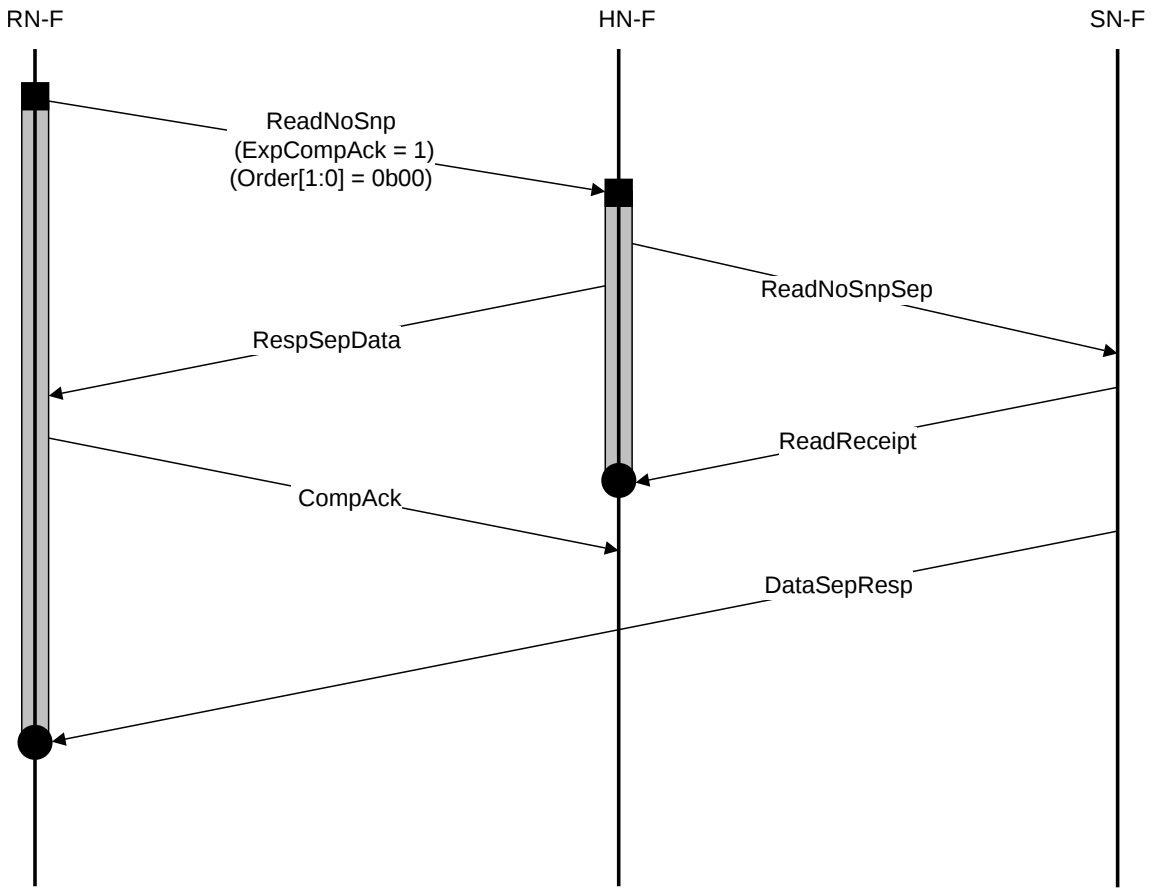

Figure B5.9: DMT Read transaction example with separate Non-data and Data-only

The steps in the ReadNoSnp transaction in Figure B5.9 are:

1. RN-F sends an unordered ReadNoSnp request to HN-F.
2. HN-F sends a ReadNoSnpSep request to SN-F.
3. HN-F sends a RespSepData response to RN-F.
4. SN-F sends a ReadReceipt to HN-F.
5. RN-F sends CompAck after receiving RespSepData.
6. SN-F sends DataSepResp to RN-F returning the read data.

B5.1.9 ReadNoSnp transaction with DMT with ordering and separate Non-data and Data-only

Figure B5.10 shows an example DMT transaction flow with ordering and separate Non-data and Data-only.

ReadNoSnp with non-zero Order field in Figure B5.10 requires that:

- Next ordered request can be sent only after receiving of RespSepData.
- RN-F must wait for RespSepData and at least one packet of DataSepResp before sending CompAck.
- HN-F must not send next ordered request to SN-F until receiving CompAck.

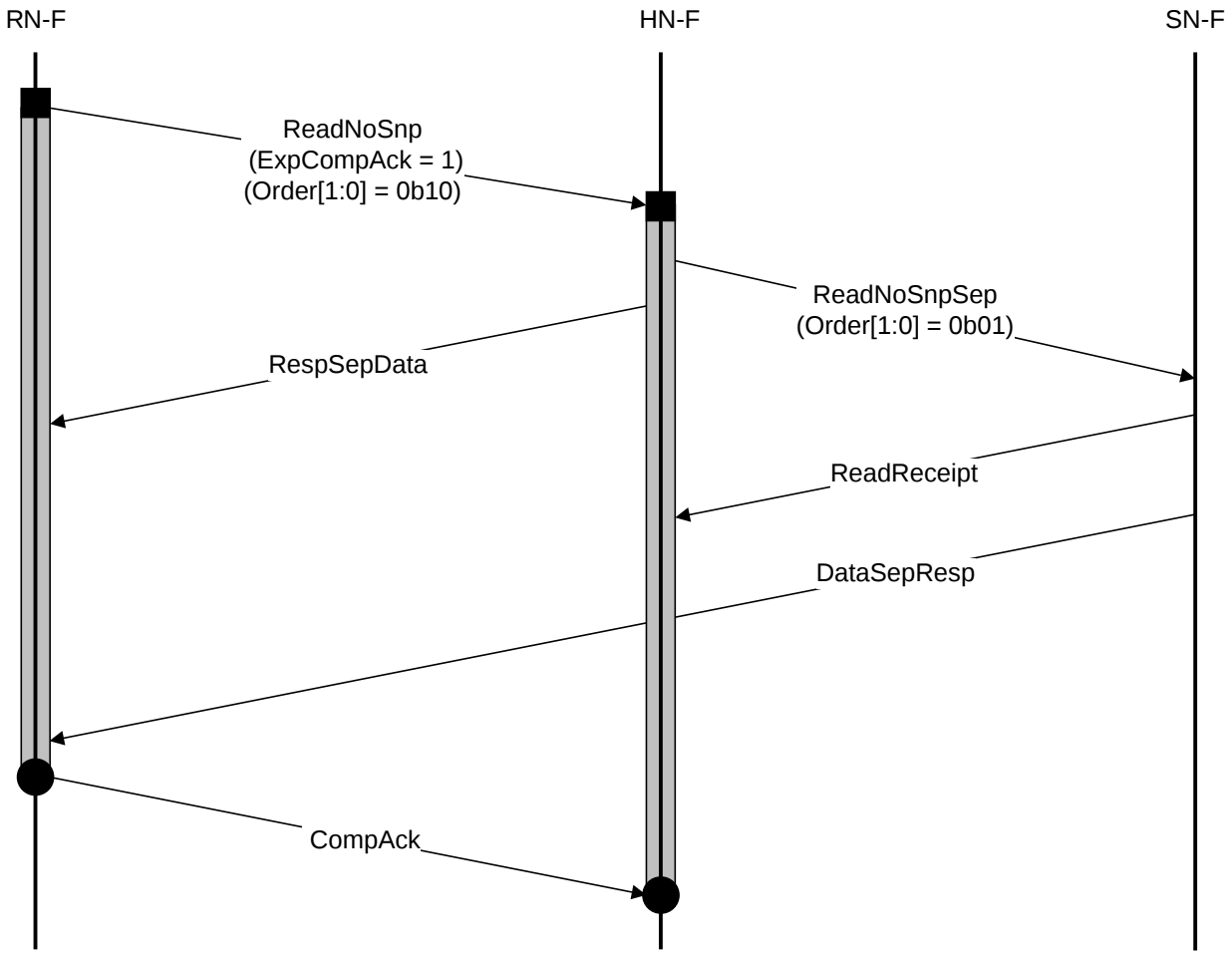

Figure B5.10: DMT Read transaction example with ordering and separate Non-data and Data-only

The steps in the Read transaction with ordering and separate Non-data and Data-only in Figure B5.10 are as follows:

1. RN-F sends ReadNoSnp to HN-F, with ExpCompAck set to 1. Order[1:0] is set to `0b10`.
2. HN-F sends ReadNoSnpSep to SN-F with Order[1:0] set to `0b01`.
3. HN-F returns RespSepData to RN-F.
4. SN-F returns ReadReceipt to HN-F.
5. SN-F sends DataSepResp to RN-F directly.
6. RN-F issues CompAck to HN-F.

B5.2 Dataless transaction flows

This section gives examples of the interconnect protocol flow for Dataless transactions.

B5.2.1 Dataless transaction without memory update

An example of this type of flow is a MakeUnique transaction.

Figure B5.11 shows the transaction flow.

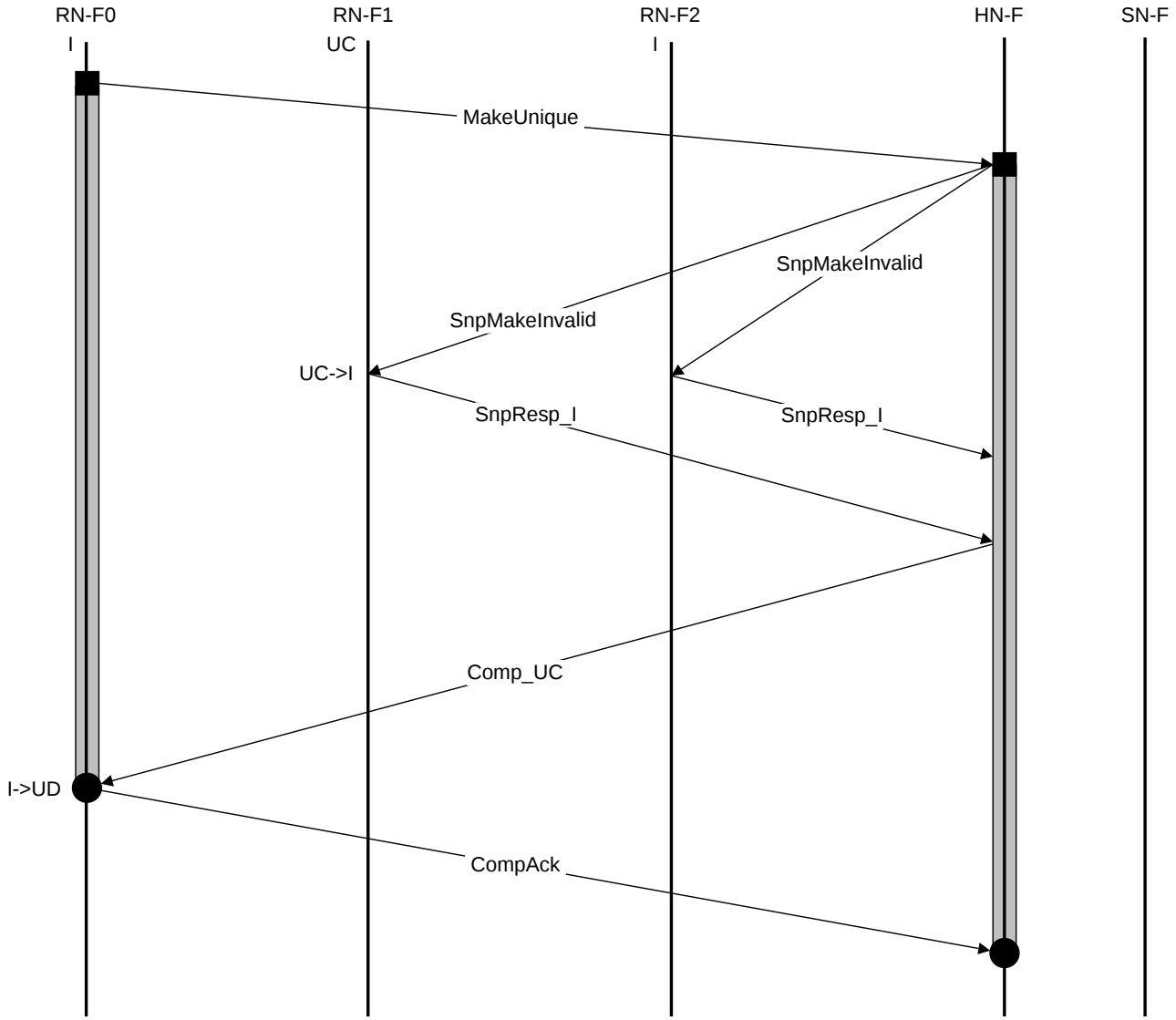

Figure B5.11: MakeUnique without memory update

The steps in the MakeUnique without memory update transaction flow in Figure B5.11 are:

1. RN-F0 sends MakeUnique request to HN-F.
2. HN-F sends SnpMakeInvalid requests to RN-F1 and RN-F2. RN-F1 cache line state transitions from UC to I.
3. RN-F1 and RN-F2 return SnpResp_I to the HN-F.

4. HN-F sends Comp_UC to RN-F0. RN-F0 cache line state transitions from I to UD.
5. RN-F0 issues CompAck response to the HN-F to indicate transaction completion.

B5.2.2 Dataless transaction with memory update

An example of this type of flow is a CleanUnique transaction.

Figure B5.12 shows the transaction flow.

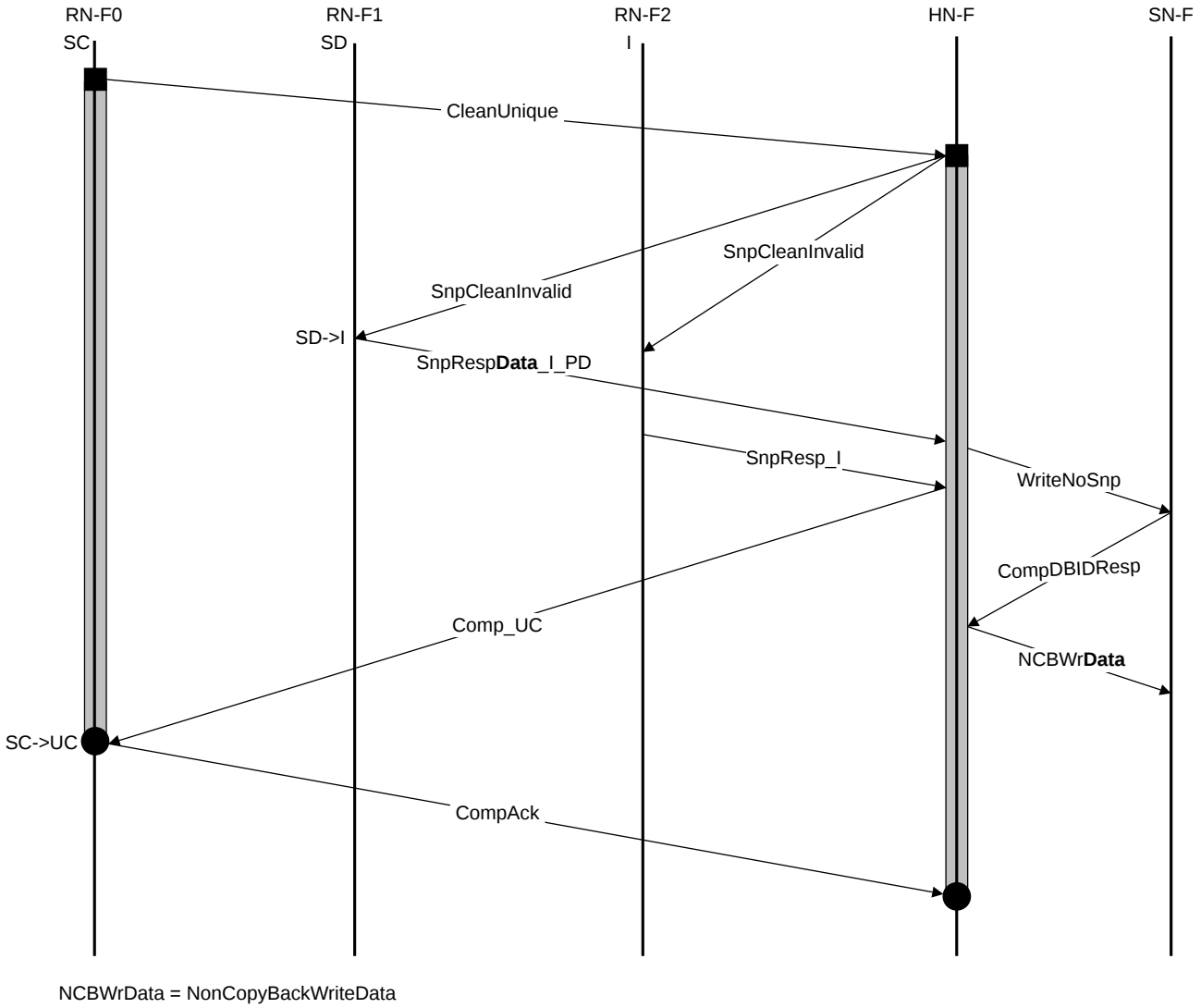

Figure B5.12: CleanUnique with memory update

The steps in the CleanUnique with memory update transaction flow in Figure B5.12 are:

1. RN-F0 sends CleanUnique request to HN-F.
2. HN-F sends SnpCleanInvalid requests to RN-F1 and RN-F2. RN-F1 cache line state transitions from SD to I.
3. RN-F1 returns SnpRespData_I_PD to HN-F. HN-F sends WriteNoSnp to SN-F.

4. RN-F2 returns SnpResp_I to the HN-F. HN-F can now send Comp_UC to RN-F0. RN-F0 cache line state transitions from SC to UC. Meanwhile, SN-F returns CompDBIDResp to HN-F. HN-F subsequently sends NCBWrData to SN-F.
5. RN-F0 issues CompAck response to the HN-F to indicate transaction completion.

B5.2.3 Persistent CMO with snoop and separate Comp and Persist

In this example of CleanSharedPersistSep transaction flow, the *Point of Persistence* (PoP) is at the SN-F.

Figure B5.13 shows the transaction flow.

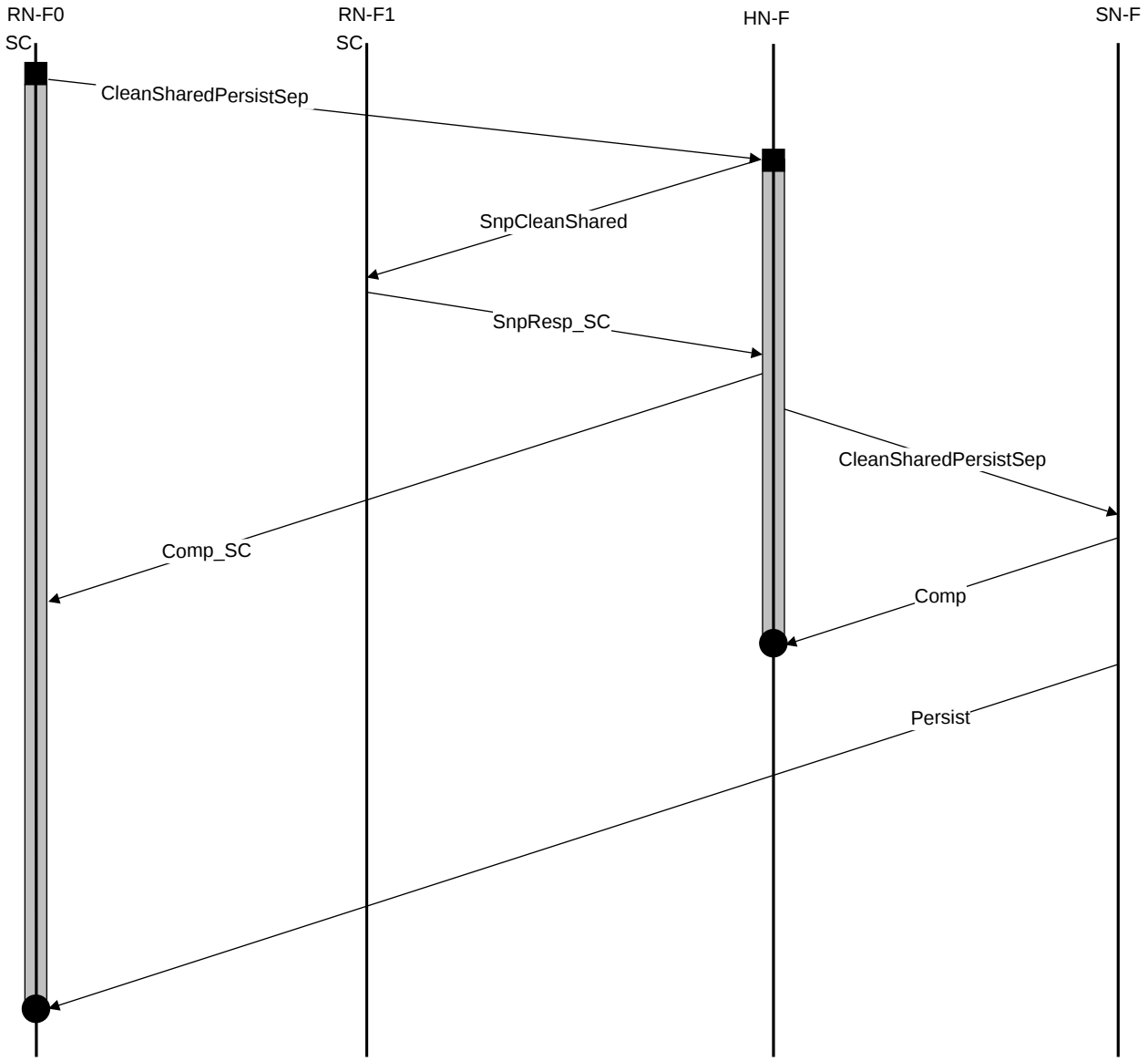

Figure B5.13: CleanSharedPersistSep transaction flow

The steps in the CleanSharedPersistSep transaction flow in Figure B5.13 are:

1. RN-F0 sends CleanSharedPersistSep request to HN-F.
2. HN-F sends SnpCleanShared request to RN-F1.
3. RN-F1 sends SnpResp_SC to HN-F.
4. HN-F sends Comp_SC to RN-F0.
5. HN-F sends CleanSharedPersistSep to SN-F after completing the writing back of all snooped Dirty data, if any.
6. SN-F returns Comp response to HN-F. SN-F directly sends Persist response to RN-F0 to indicate the request has reached PoP and data from any prior writes to the same location is pushed to PoP.

B5.2.4 Evict transaction

Figure B5.14 shows the Evict transaction flow.

Note

The Evict request is a hint. A Comp response can be given by HN-F without updating the Snoop Filter or Snoop Directory.

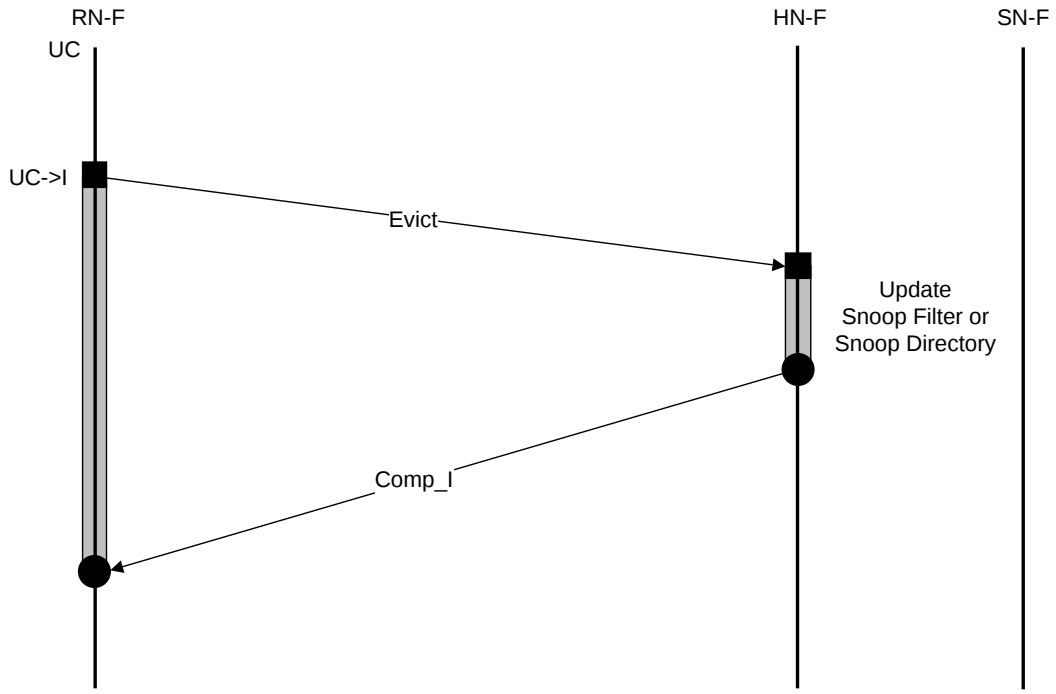

Figure B5.14: Evict transaction flow

The steps in the Evict transaction flow in Figure B5.14 are:

1. RN-F0 cache line state transitions from UC to I and sends Evict request to HN-F. HN-F receives and allocates the request.
2. HN-F returns Comp_I response and deallocates the request. RN-F0 deallocates the request.

Note

The cache state at the Requester must change to Invalid before the Evict message is sent.

B5.3 Write transaction flows

This section gives examples of the interconnect protocol flow for Write transactions.

B5.3.1 Write transaction with no snoop and separate responses

Figure B5.15 shows a WriteNoSnp transaction flow.

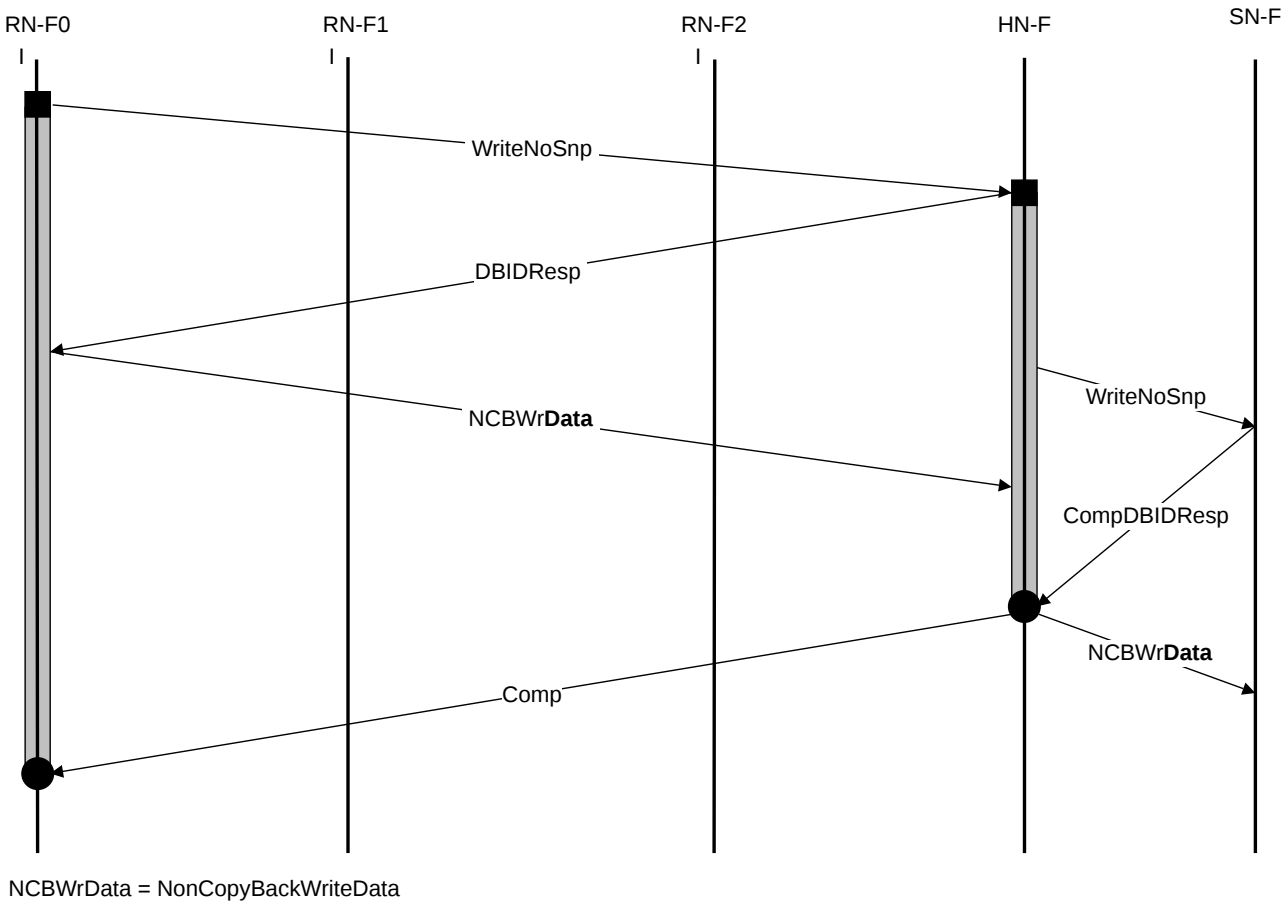

Figure B5.15: WriteNoSnp with separate responses from Home Node to Request Node

The steps in the WriteNoSnp transaction flow in Figure B5.15 are:

1. RN-F0 issues a WriteNoSnp transaction to HN-F. HN-F receives and allocates the request.
2. HN-F sends DBIDResp without Comp. Meanwhile, HN-F sends WriteNoSnp to SN-F.
3. RN-F0 responds with data, NCWrData. SN-F returns CompDBIDResp to HN-F.
4. HN-F sends a Comp after receiving CompDBIDResp from SN-F. HN-F sends NCBWrData to the SN-F.

> **Note**
>
> This flow example shows Comp is sent after CompDBIDResp is received from SN-F. However, HN-F is permitted to send Comp anytime after receiving the WriteNoSnp request from RN-F0.

5. RN-F0 waits for Comp from HN-F and deallocates its request.

Figure B5.15 shows the flow, the copy of data being transferred is marked in bold.

B5.3.2 Write transaction with snoop and separate responses

An example of this type of flow is a WriteUniquePtl transaction.

Figure B5.16 shows the transaction flow. The copy of data being transferred is marked in bold.

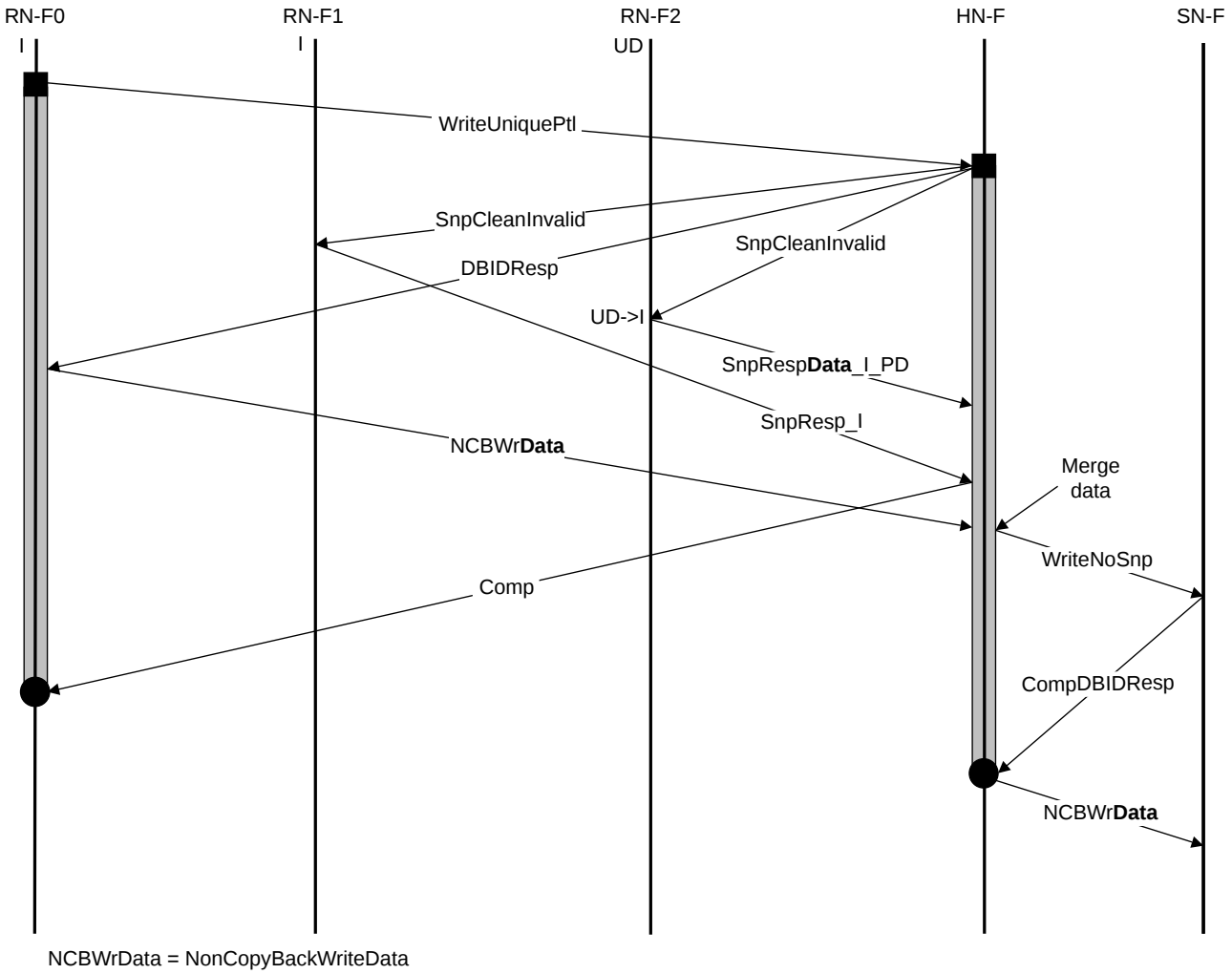

Figure B5.16: WriteUniquePtl with snoop

The steps in the WriteUniquePtl with snoop transaction flow in Figure B5.16 are:

1. RN-F0 sends WriteUniquePtl request to HN-F.
2. HN-F sends SnpCleanInvalid requests to RN-F1 and RN-F2. HN-F also returns DBIDResp to RN-F0. RN-F2 cache line state transitions from UD to I.
3. RN-F1 sends SnpResp_I and RN-F2 sends SnpRespData_I_PD to HN-F.
4. RN-F0 issues NCBWrData to HN-F. HN-F merges the write data with the dirty line and sends WriteNoSnp to SN-F.

5. HN-F sends Comp response to RN-F0.
6. SN-F returns CompDBIDResp to HN-F.
7. HN-F sends NCBWrData to SN-F.

B5.3.3 CopyBack Write transaction to memory

An example of this type of flow is a WriteBackFull transaction.

Figure B5.17 shows the transaction flow. The copy of data being transferred is marked in bold.

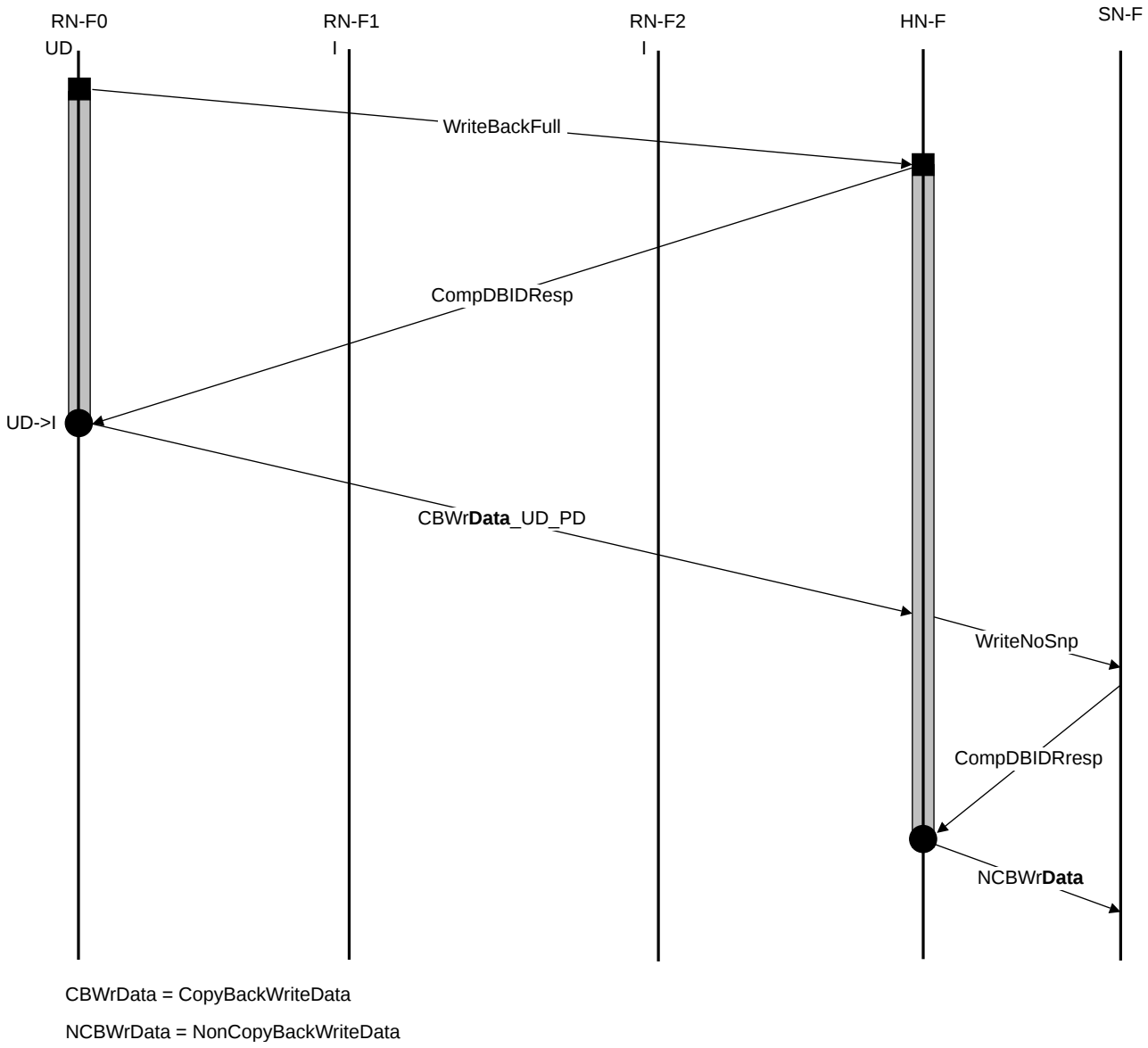

Figure B5.17: WriteBackFull transaction flow

The steps in the WriteBackFull transaction flow in Figure B5.17 are:

1. RN-F0 sends WriteBackFull request to HN-F.
2. HN-F returns CompDBIDResp to RN-F0. RN-F0 cache line state transitions from UD to I.
3. RN-F0 sends CBWrData_UD_PD to HN-F. HN-F sends WriteNoSnp to SN-F.
4. SN-F returns CompDBIDResp to HN-F.
5. HN-F sends NCBWrData to SN-F.

B5.4 Atomic transaction flows

This section shows flows for different Atomic transaction types. It contains the following subsections:

- B5.4.1 *Atomic transactions with data return*
- B5.4.2 *Atomic transaction without data return*
- B5.4.3 *Atomic operation executed at the SN*

B5.4.1 Atomic transactions with data return

This flow is applicable to:

- AtomicLoad
- AtomicCompare
- AtomicSwap

B5.4.1.1 Atomic transaction with snoops and data return

Figure B5.18 shows the atomic operation executed at HN-F.

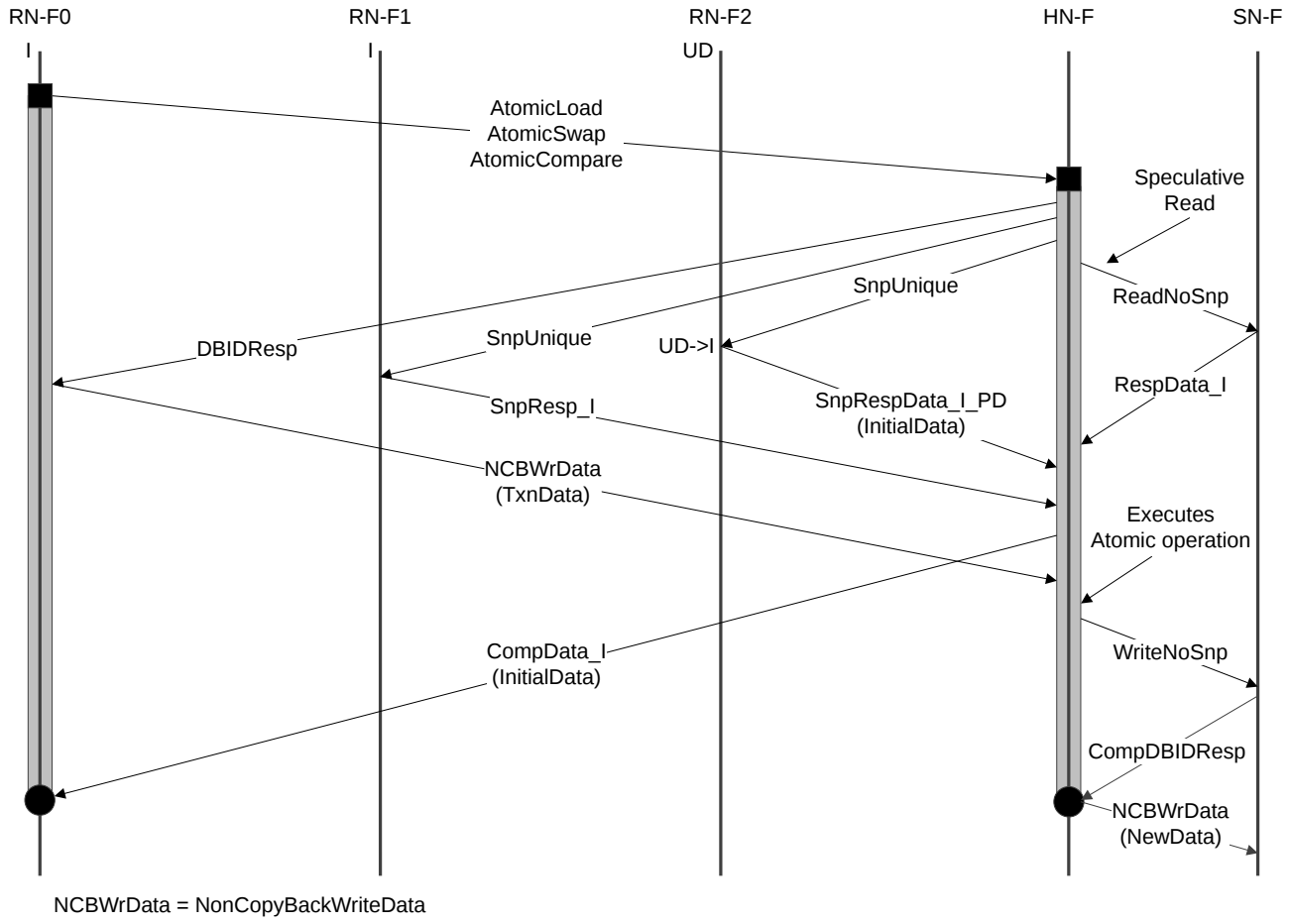

Figure B5.18: AtomicLoad, AtomicSwap, or AtomicCompare executed at HN-F

The steps in Figure B5.18 are:

1. RN-F0 sends an Atomic transaction to HN-F.
2. After receiving the Atomic request, HN-F:
   - Sends DBIDResp to RN-F0 to obtain the Atomic transaction data.
   - Sends SnpUnique Snoop request to other RN-Fs after determining that snoops are required.
   - HN-F is permitted, but not required, to send a speculative ReadNoSnp to SN-F.
3. RN-F2 has the cache line in UD state and responds by sending data and invalidating its own cached copy.
   - The response is SnpRespData_I_PD.
   - This data is marked as (InitialData) in Figure B5.18 to be distinguishable from the data sent by the Requester and the data written to SN-F after the atomic operation is executed.
   - HN-F also receives a second Snoop response, SnpResp_I, from RN-F1.
4. After receiving all Snoop responses, HN-F sends CompData_I to the Requester.
   - The data sent with Comp is the initial copy of the data.
   - This data must not be cached in a coherent state at RN-F0.
5. In response to the DBIDResp sent previously, HN-F receives the NonCopyBackWriteData_I response from the Requester.
   - This data is marked as (TxnData) in Figure B5.18 to be distinguishable from the data sent by RN-F2 in response to the Snoop request from HN-F.
6. Once HN-F receives the NonCopyBackWriteData_I response from the Requester, and the Snoop response with data from RN-F2, the atomic operation is executed.
   - The resulting value after atomic operation execution, marked as (NewData) in Figure B5.18, is written to SN-F.
7. In this example, the read data received due to the speculative read is discarded by HN-F.

Note

In Figure B5.18, the CompData_I response from HN-F can be sent when all Snoop responses are received.

Alternatively, to aid error reporting, CompData_I can be delayed until NonCopyBackWriteData is received from the Requester and the atomic operation is executed.

B5.4.1.2 Atomic transaction without snoops and with data return

Figure B5.19 shows the atomic operation executed at the Home Node.

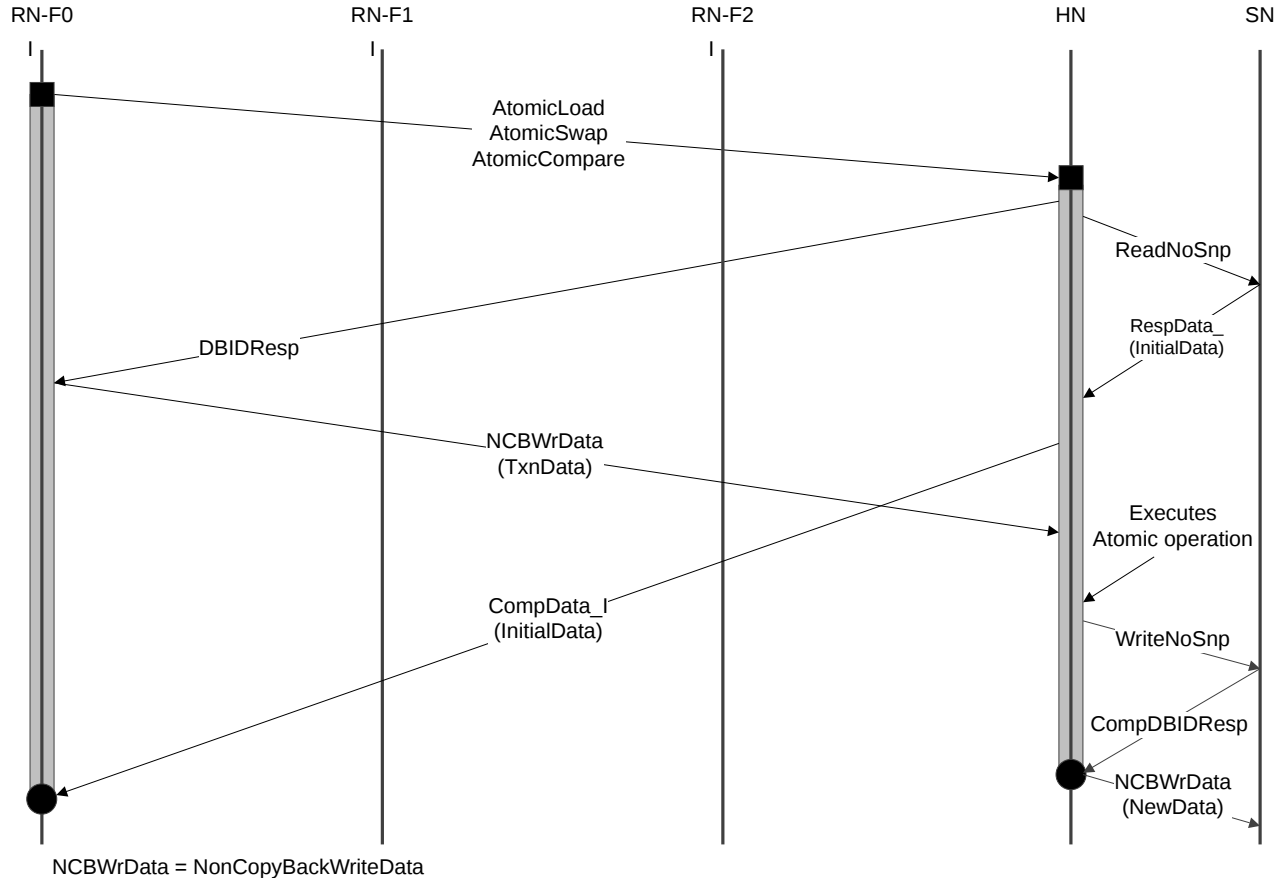

Figure B5.19: AtomicLoad, AtomicSwap, or AtomicCompare executed at Home Node

The steps in the Atomic transaction executed at the Home Node transaction flow in Figure B5.19 are:

1. RN-F0 sends Atomic transaction request to HN.
2. HN returns DBIDResp to RN-F0.
3. HN sends ReadNoSnp request to SN. Meanwhile, RN sends NCBWrData (TxnData) to HN.
4. SN returns RespData_I (InitialData) to HN.
5. HN returns CompData_I (InitialData).
6. HN executes the Atomic operation and sends WriteNoSnp request to SN.
7. SN returns CompDBIDResp to HN.
8. HN sends the result of the atomic operation to SN, marked as (NewData).

B5.4.2 Atomic transaction without data return

This flow is applicable to AtomicStore transactions.

B5.4.2.1 Atomic transaction with snoops and without data return

Figure B5.20 shows the atomic operation executed at HN-F. The flow is similar to the Atomic transaction with snoop and with data return, except that the Comp response to RN-F0 does not include data.

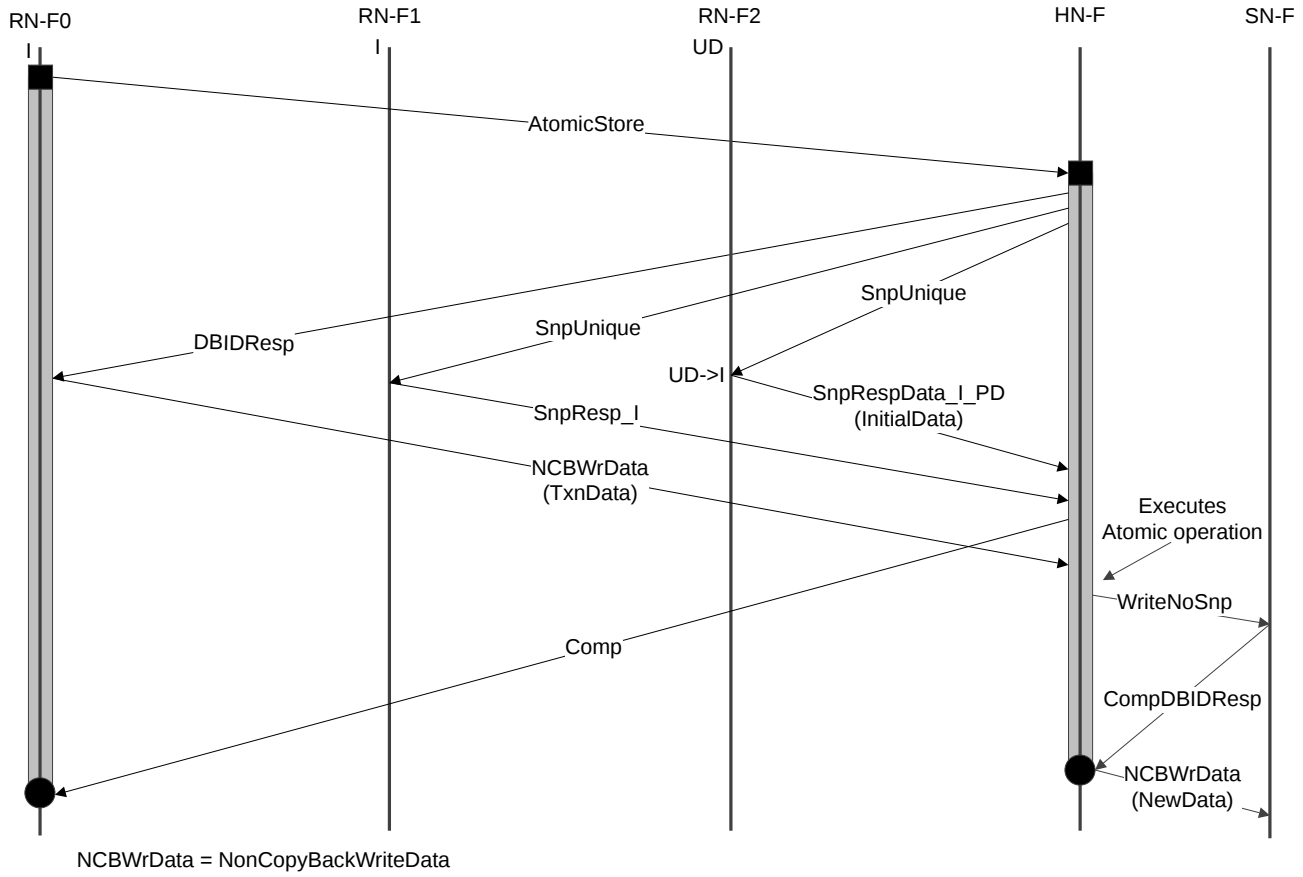

Figure B5.20: AtomicStore executed at HN-F

The steps in the AtomicStore transaction executed at HN in Figure B5.20 are:

1. RN-F0 sends AtomicStore request to the HN-F.

2. HN-F issues DBIDResp to the Requester, RN-F0, and SnpUnique requests to RN-F1 and RN-F2.
3. RN-F1 returns SnpResp_I response to the HN-F. RN-F2 has the cache line in UD state and returns a SnpRespData_I_PD (InitialData) response. This invalidates the cached copy in RN-F2.
4. RN-F0 sends the write data (TxnData) to the HN-F. The HN-F executes the AtomicStore operation locally.
5. HN-F issues a WriteNoSnp request to the SN-F.
6. SN-F returns CompDBIDResp to the Home.
7. HN-F sends the result of the atomic operation to the SN-F, marked as (NewData).

B5.4.2.2 Atomic transaction without snoops and without data return

Figure B5.21 shows the atomic operation executed at the Home Node. The flow is similar to the Atomic transaction without snoop and with data return except that the Comp response to the Request Node does not include data.

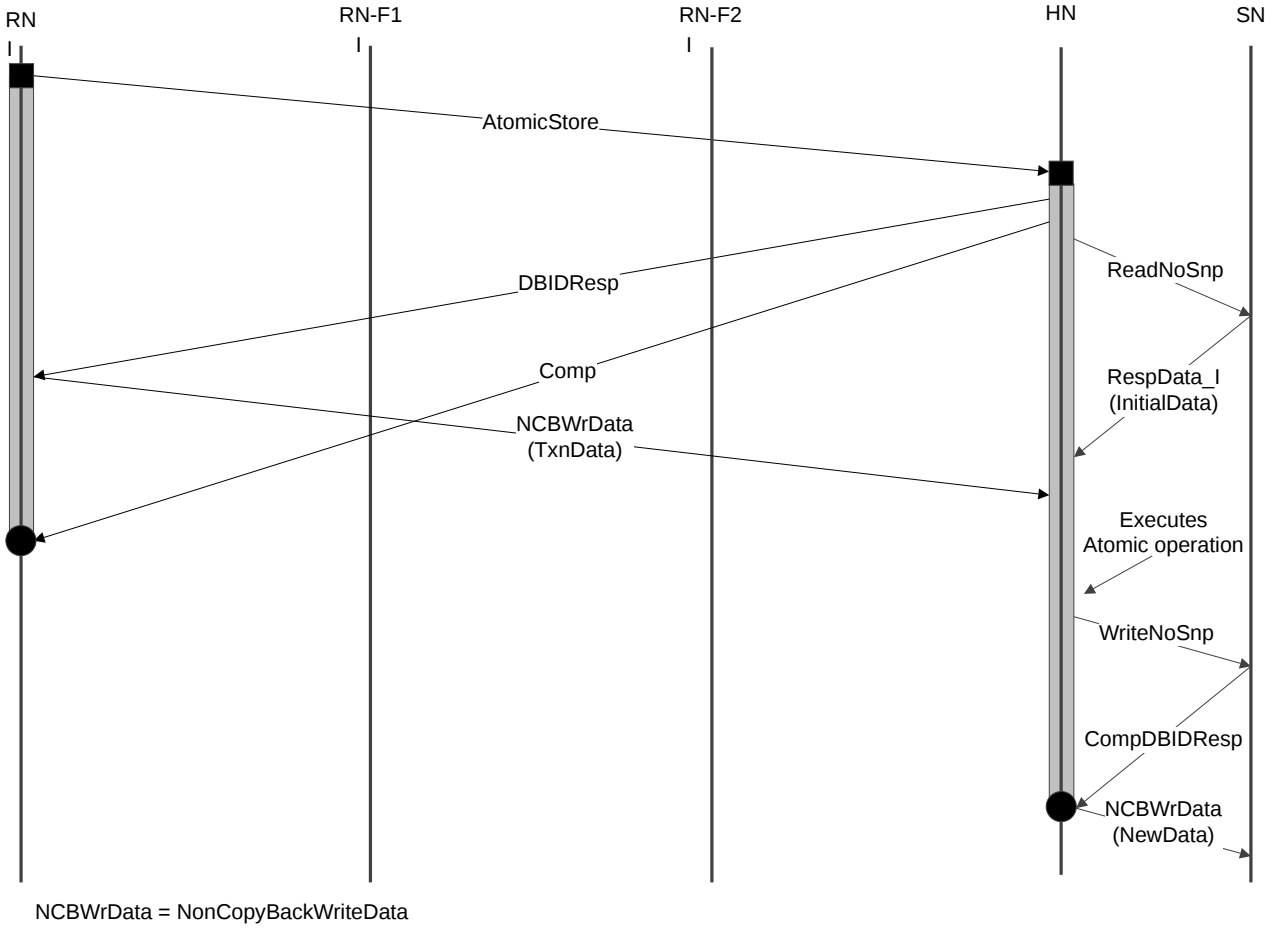

Figure B5.21: AtomicStore executed at Home Node

Note

In Figure B5.21, the read from the Subordinate Node is required to obtain the InitialData and is not speculative.

The Comp response from the Home Node can be combined with the DBIDResp response.

The steps in the AtomicStore transaction executed at the Home Node in Figure B5.21 are:

1. RN sends AtomicStore request to the HN.
2. HN returns DBIDResp response to RN, indicating the RN can send the write data to the HN.
3. HN sends Comp response to RN and sends ReadNoSnp request to the SN.
4. SN returns RespData_I to the HN.
5. Once the HN has the write data from the RN, the atomic operation is executed and WriteNoSnp is sent to SN.
6. SN returns CompDBIDResp to the HN.
7. HN sends the result of the atomic operation to the SN, marked as NewData.

B5.4.3 Atomic operation executed at the SN

Figure B5.22 shows an example Atomic transaction flow where the SN-F is executing the atomic operation.

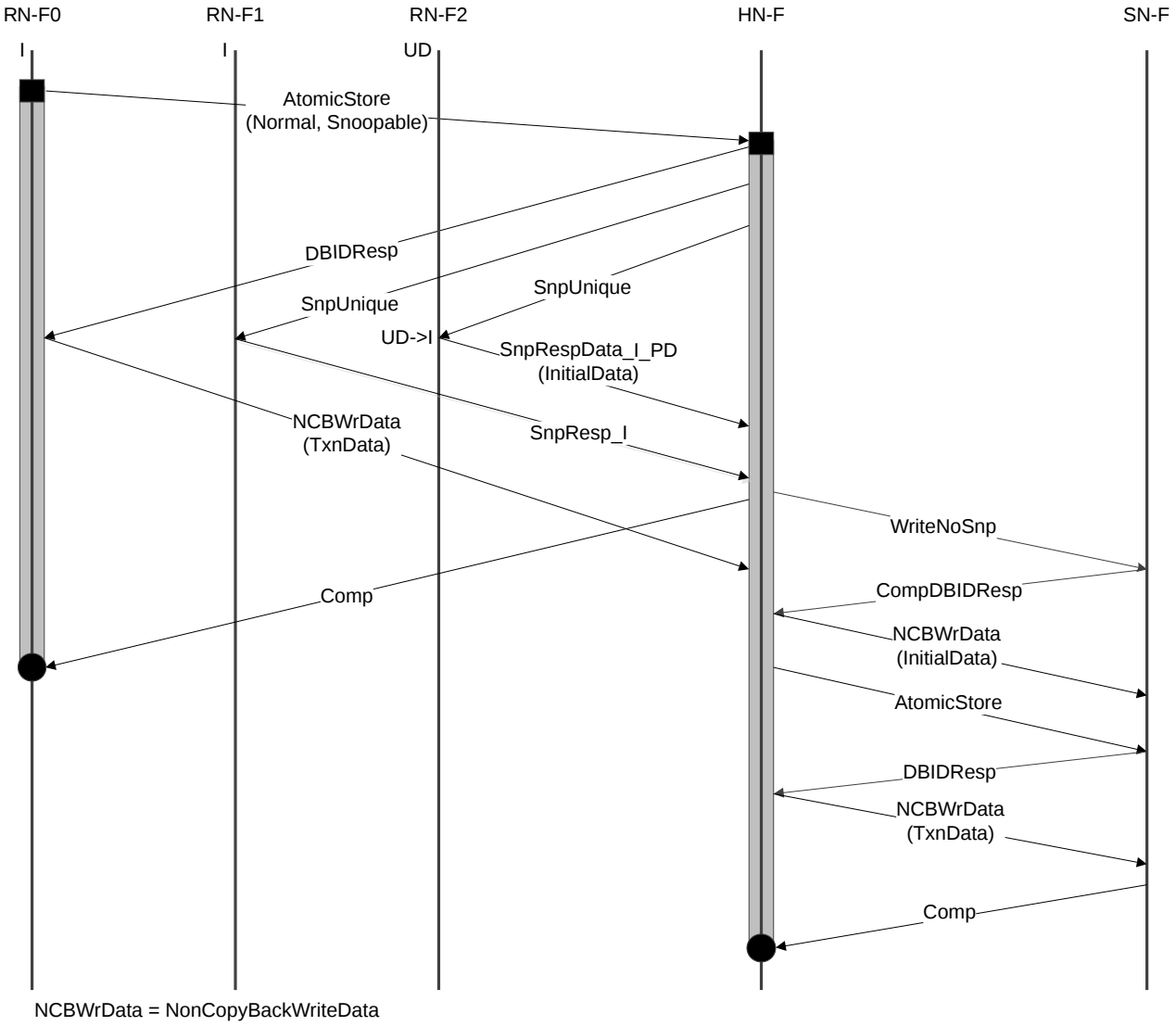

Figure B5.22: AtomicStore executed at SN-F

The steps in the AtomicStore transaction executed at SN-F in Figure B5.22 are:

1. RN-F0 sends an AtomicStore transaction to HN-F.
   - The Atomic request is to a Snoopable address location.
2. After receiving the Atomic request, HN-F:
   - Sends DBIDResp to RN-F0 to obtain the Atomic transaction data.
   - Sends a SnpUnique to other RN-Fs after determining that snoops are required.
3. RN-F2 has the cache line in UD state and responds by sending data and invalidating its own cached copy.
   - The response is SnpRespData_I_PD.
   - This data is marked as (InitialData) in Figure B5.22 to be distinguishable from the data sent by the Requester and the data written to SN-F to execute the atomic operation.
   - HN-F also receives a second Snoop response, SnpResp_I, from the other snooped RN-F.
4. HN-F writes the received data to SN-F using a WriteNoSnp transaction.
5. In response to the DBIDResp sent previously, HN-F receives the NonCopyBackWriteData response from the Requester.
6. HN-F after sending the Snoop response data to SN-F, sends an AtomicStore transaction request to SN-F, and executes the sequence of messages required to complete the Atomic transaction.
7. The HN-F deallocates the request once the Comp response is sent to the Requester and the Comp response for the Atomic transaction is received from SN-F.
   - The Comp response from HN-F can be sent when all the Snoop responses are received.

B5.5 Stash transaction flows

This section shows example interconnect protocol flows for the two Stash transaction types:

- B5.5.1 *Write with Stash hint*
- B5.5.2 *Independent Stash request*

B5.5.1 Write with Stash hint

Figure B5.23 shows an example WriteUniqueStash with Data Pull transaction flow.

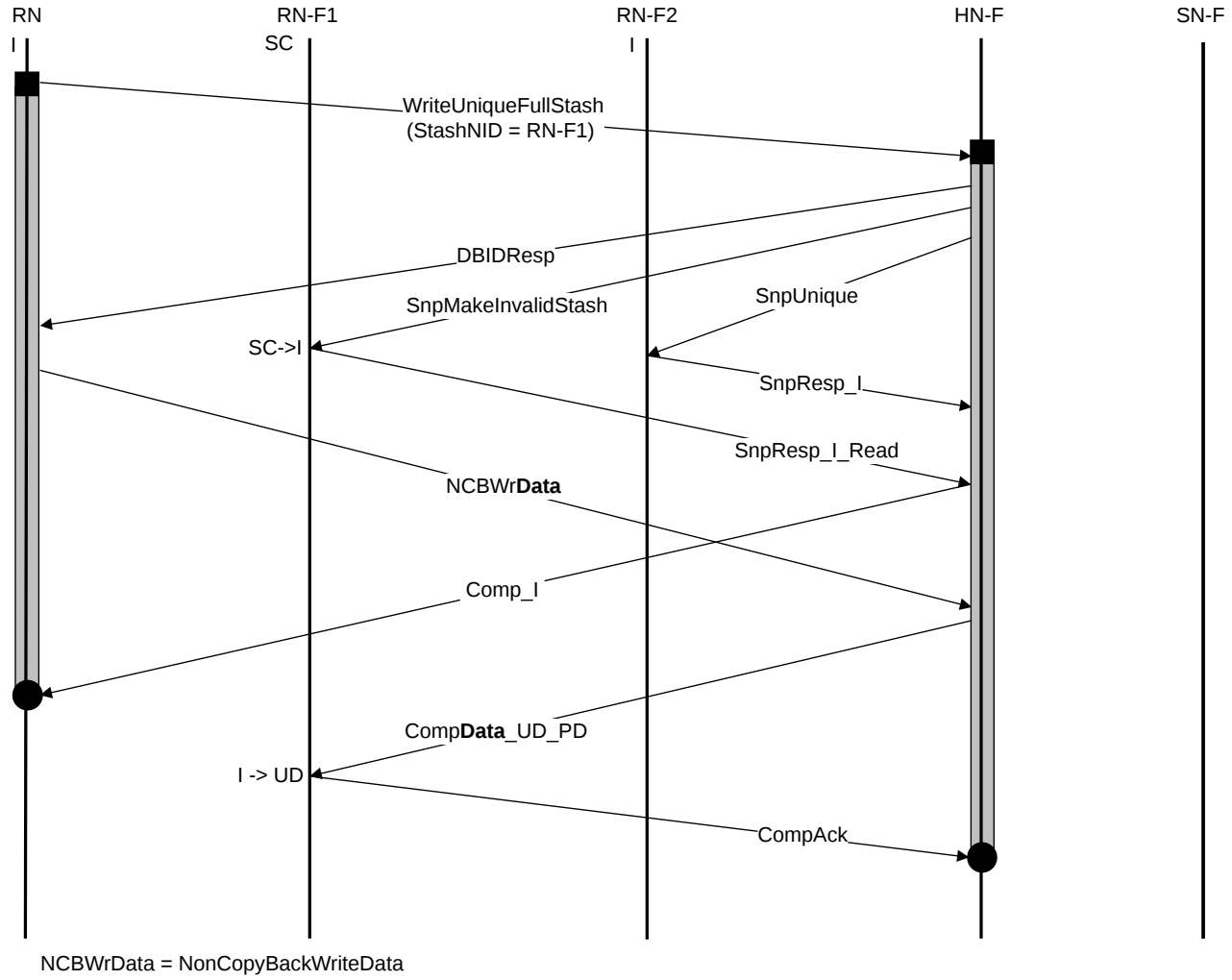

Figure B5.23: WriteUniqueStash with Data Pull

The steps in Figure B5.23 are:

1. The Request Node sends a WriteUniqueFullStash request to HN-F with the Stash target identified as RN-F1. Typically, the requesting Request Node is an RN-I.
2. HN-F sends SnpMakeInvalidStash to RN-F1 and SnpUnique to RN-F2.
3. RN-F1 and RN-F2 send SnpResp response to HN-F. The Snoop response from RN-F1 also includes a Read request, that is, the Data Pull.

4. HN-F treats the Read request from RN-F1 as a ReadUnique, and sends a combined CompData to RN-F1. CompData response includes the data written by the Request Node.
5. RN-F1 sends CompAck to HN-F to complete the Read transaction.

B5.5.2 Independent Stash request

Figure B5.24 shows an example StashOnce with Data Pull transaction flow.

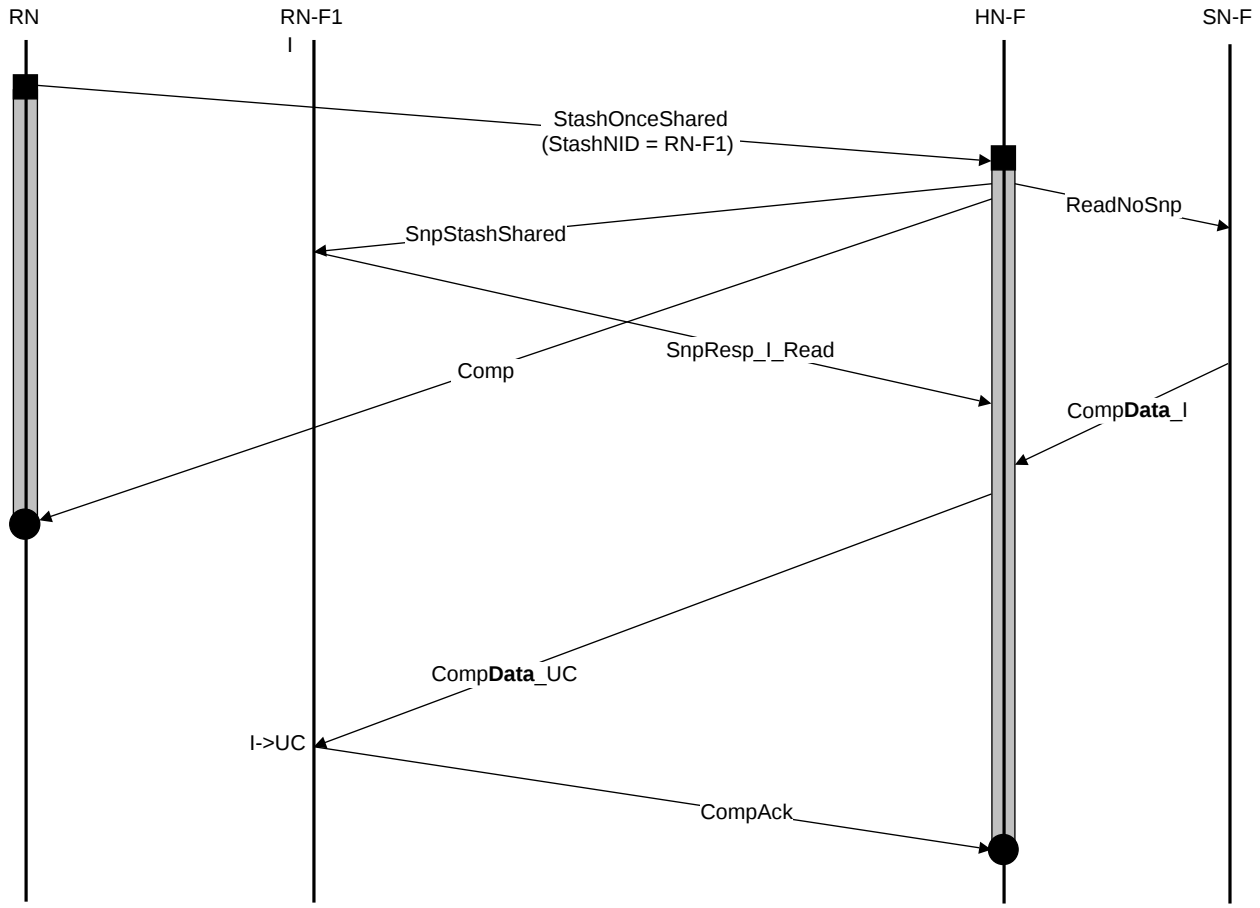

Figure B5.24: StashOnceShared with Data Pull

The steps in Figure B5.24 are:

1. The Request Node sends a StashOnceShared request to HN-F with the Stash target identified as RN-F1.
2. HN-F sends a Comp response after establishing processing order for the received request that is guaranteeing the request is processed before a request to the same address received later from any Requester.
3. HN-F sends a SnpStashShared snoop to RN-F1, and a ReadNoSnp request to SN-F to fetch data.
4. RN-F1 sends SnpResp_I_Read response to HN-F.
5. HN-F treats the Read request from RN-F1 as a ReadNotSharedDirty, and sends a combined CompData to RN-F1.
6. RN-F1 sends CompAck to HN-F to complete the Read transaction.

B5.6 Hazard handling examples

This section shows how CopyBack-Snoop request hazard conditions are handled at the Requester, and how various requests to request and request to snoop request hazard conditions are handled at the HN-F. It contains the following subsections:

B5.6.1 Snoop request

Figure B5.25 shows a Snoop request to an RN-F hazarding a pending CopyBack request at Time C.

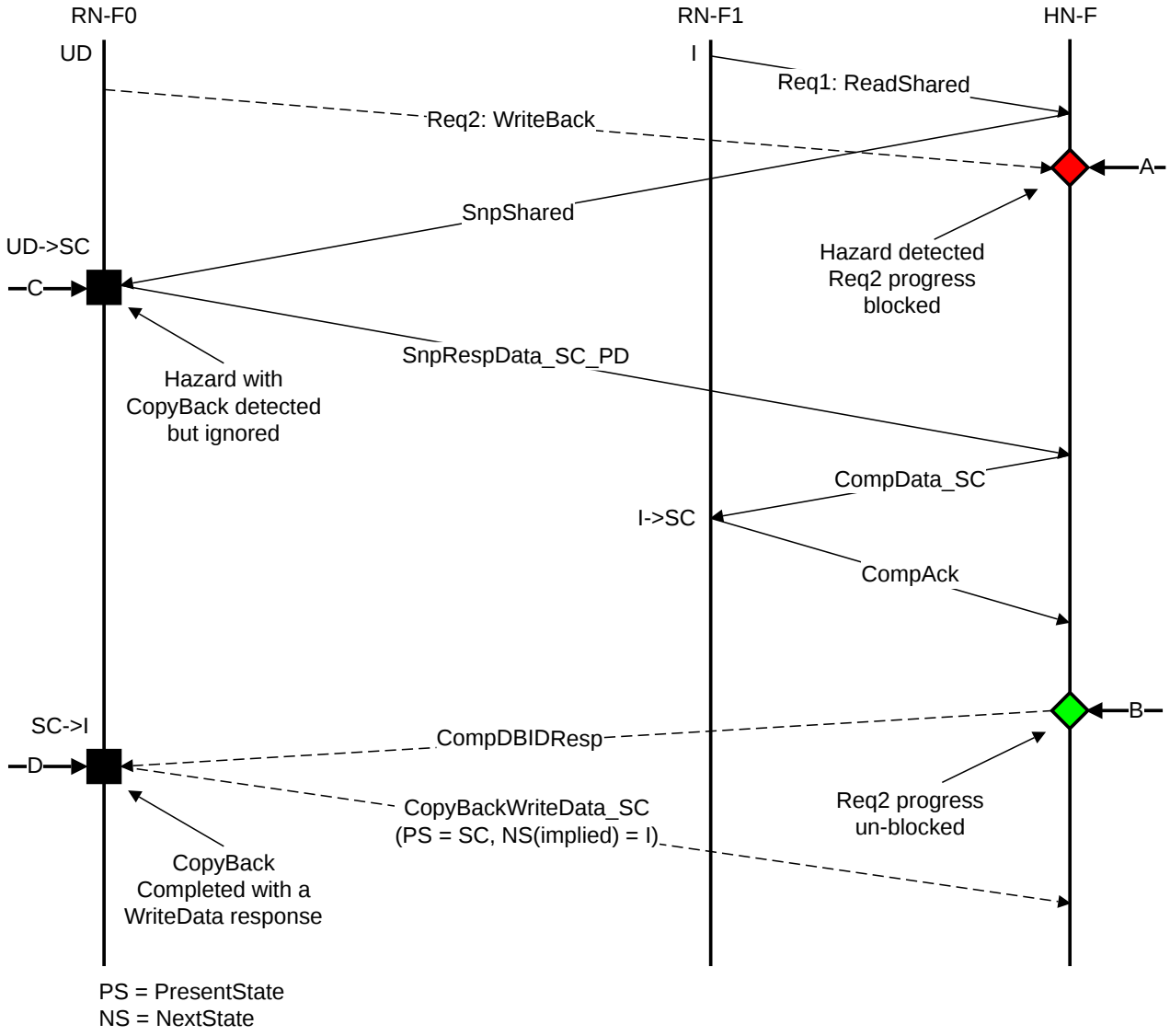

Figure B5.25: CopyBack-Snoop hazard at RN-F example

The steps required to resolve this hazard are:

1. At Time C:
   - The SnpShared transaction ignores the hazard and reads the cache line data.
   - The cache line state is changed from UD to SC.
2. At Time D:
   - The CompDBIDResp for the CopyBack is sent to RN-F0.
   - RN-F0 sends back a CopyBackWriteData_SC response.
   - The cache line state is changed from SC to I.

   The data is clean for coherence and is not required to be sent to the interconnect for correct functionality. However, the protocol requires the CopyBack flow to be consistent irrespective of a snoop hazard. The cache line state in the WriteData response is SC because that is the state of the cache line when the WriteData response is sent.

Note

The response to a Snoop request that hazards with an outstanding Evict must be SnpResp_I.

The only response that can be received for a CopyBack request, while a Snoop response to a Snoop request to the same address is pending, is a RetryAck. This includes data, if applicable.

Figure B5.26 shows a further example of a Snoop request hazarding with an outstanding CopyBack request. In this example, the Snoop request is a SnpOnce request generated as a result of a ReadOnce request from RN-F1. The SnpOnce request receives a copy of the data with the Snoop response but does not change the cache line state. In this case, the final data response from RN-F0 indicates that the data is Dirty and that HN-F must write the data back to memory.

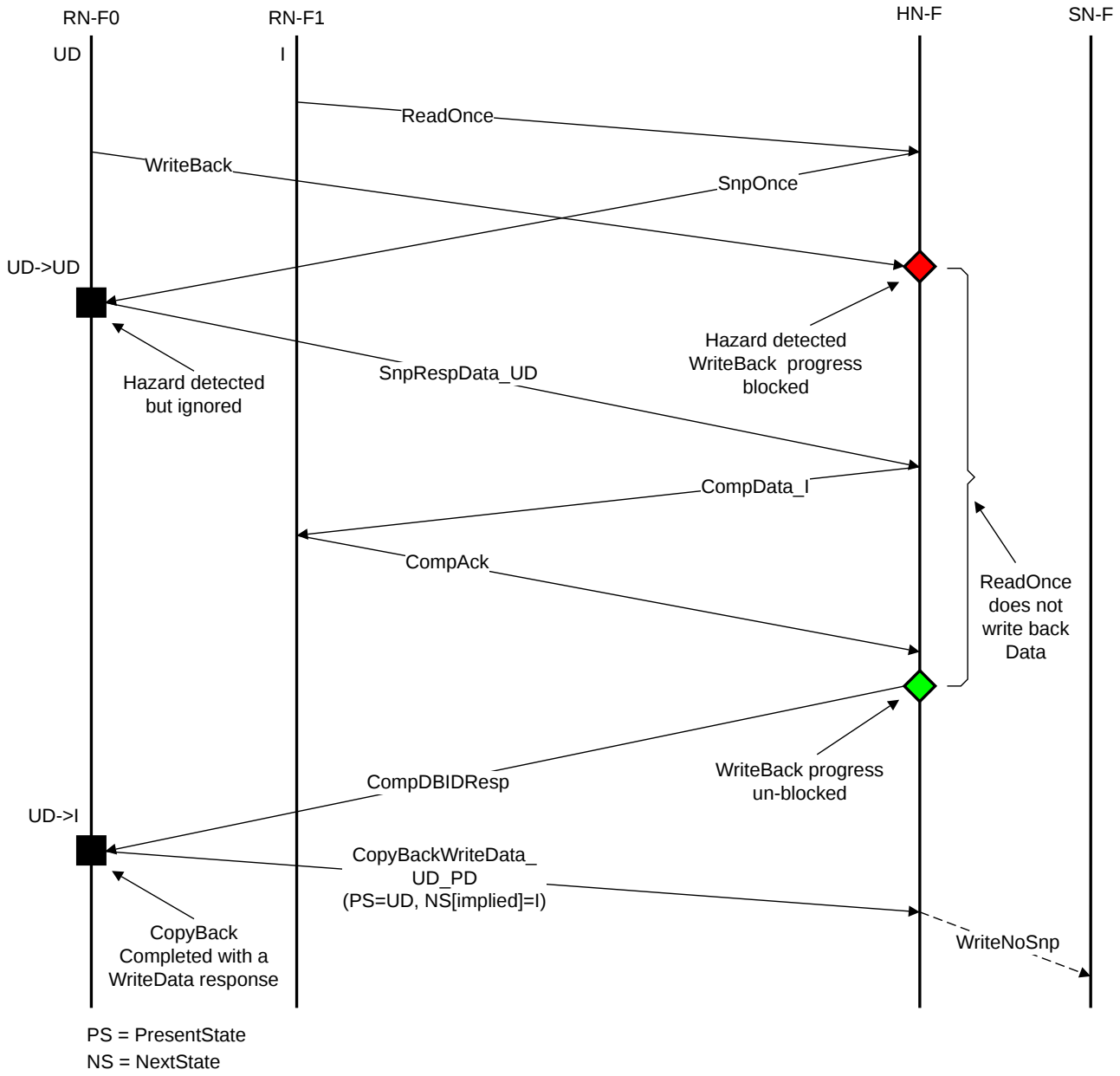

Figure B5.26: CopyBack-Snoop hazard with no cache state change example

B5.6.2 Request

If more than one request to the same cache line is ready to be processed at the HN-F, the HN-F can select the next request in any order. The exception for this is when the two requests have an ordering requirement and are from the same source, therefore the order of processing must match the order of arrival.

Figure B5.27 shows an example where a ReadShared and a ReadUnique, for the same cache line, arrive at the HN-F at approximately the same time.

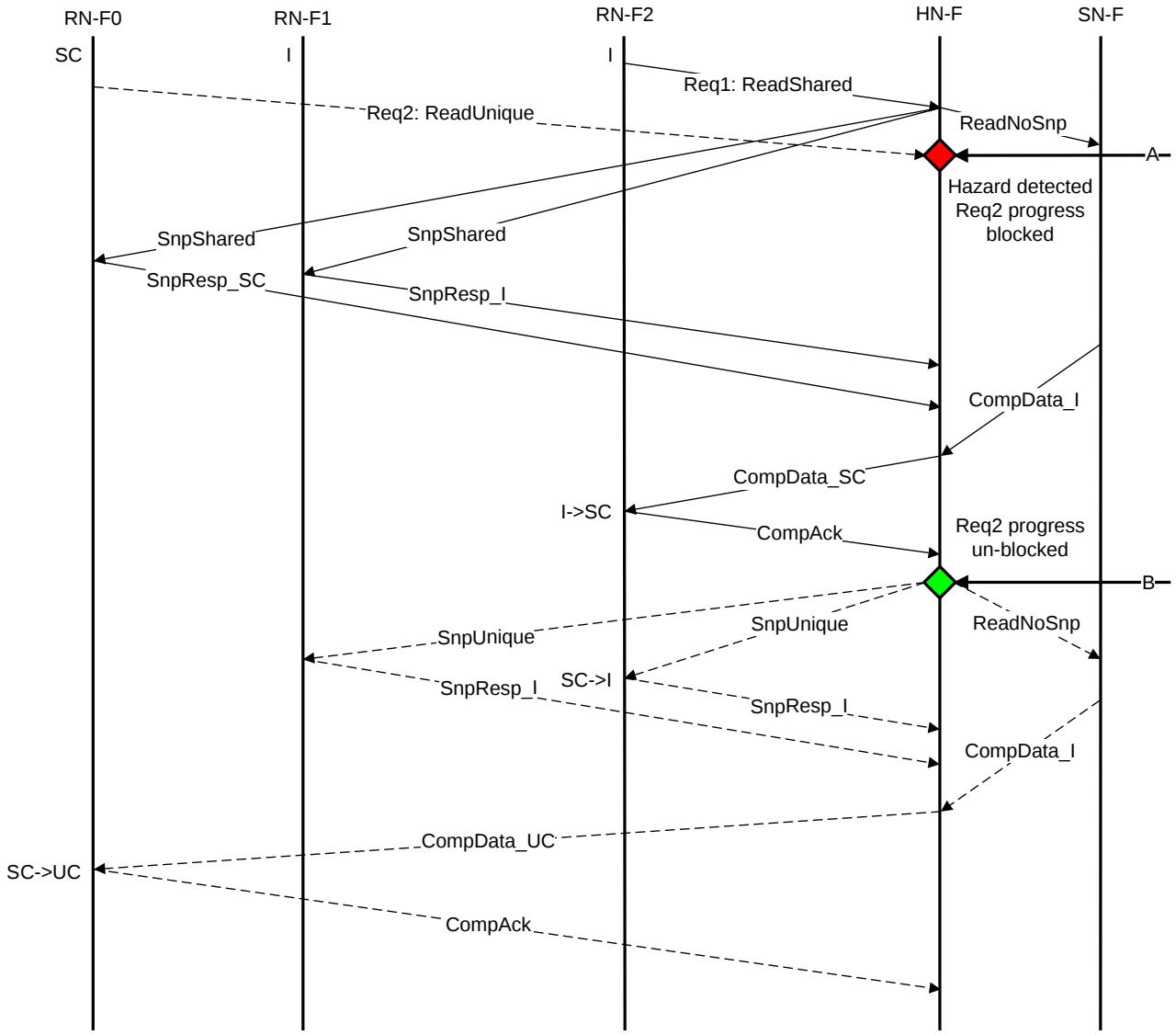

Figure B5.27: Read-Read request hazard example

The steps required to resolve this hazard in Figure B5.27 are:

1. At Time A:
   - ReadUnique from RN-F0 arrives and hazards a ReadShared request from RN-F2 for which the HN-F has already sent Snoop requests.
   - ReadUnique progress is blocked at the HN-F.
2. At Time B:
   - The HN-F has completed the ReadShared transaction request from RN-F2.
   - The ReadShared transaction is considered to be complete and the HN-F unblocks the ReadUnique transaction request from RN-F0.

Except for ReadNoSnp, the flows are similar if the two transactions, that Figure B5.27 shows, are replaced by any Read request type, or Dataless request type:

- A Read transaction request without DMT or DCT or separate Comp and data response is completed at the HN-F when both of the following are true:
  - All CompData is sent and, if applicable, CompAck is received. A CompAck is only required for transactions that assert ExpCompAck in the original request message.
  - A memory update is completed if required.

B5.6.3 Read or Dataless Request

A hazard between a Read or Dataless request and a CopyBack request at the HN-F is treated similarly to the Read-Read hazard described in B5.6.2 *Request*. See also B5.6.1 *Snoop request*.

Figure B5.28 shows the case where a ReadShared and a WriteBack, for the same cache line, arrive at the HN-F at approximately the same time.

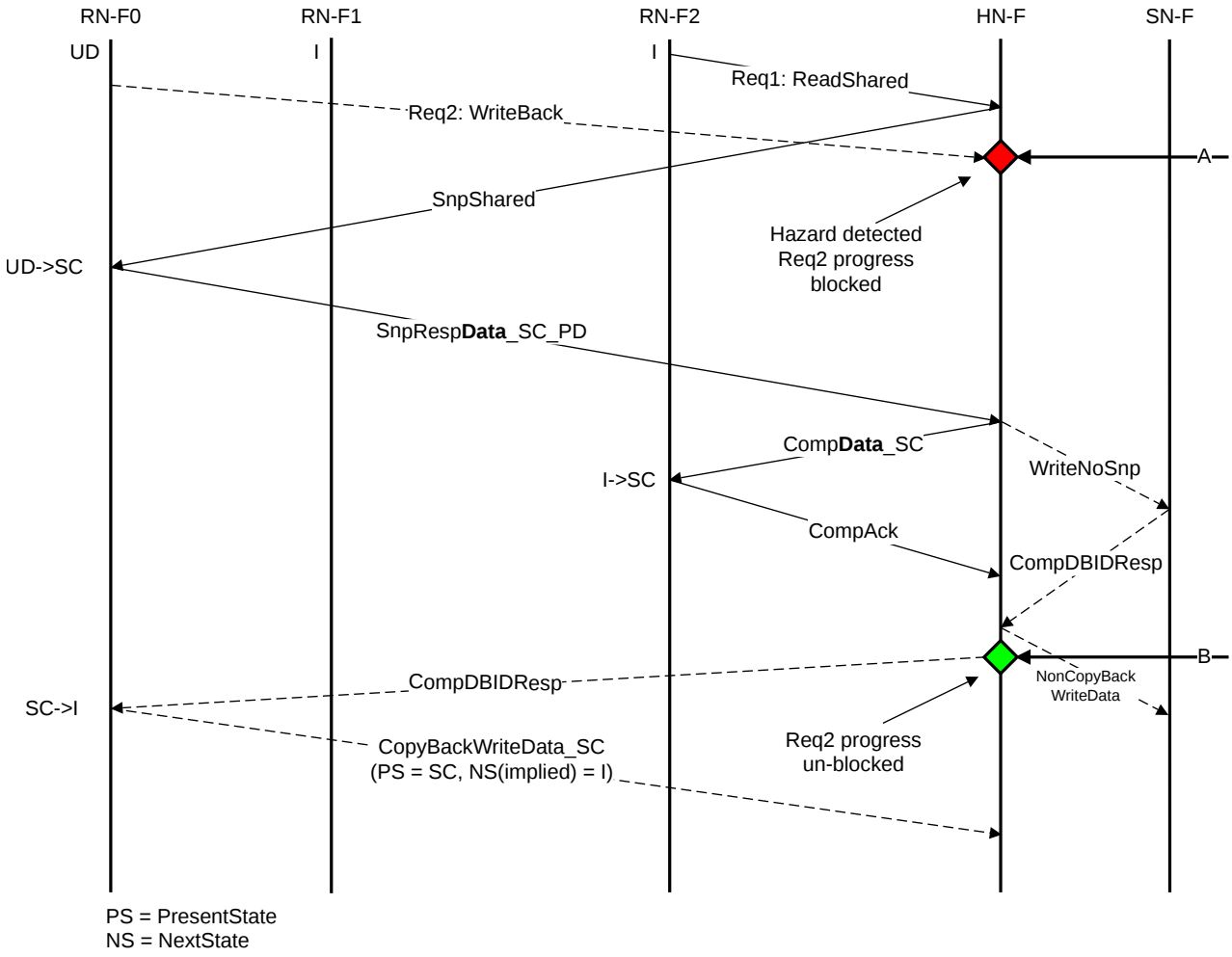

Figure B5.28: Read - CopyBack or Dataless - CopyBack hazard example

The steps required to resolve this hazard in Figure B5.28 are:

1. At Time A:
   - A WriteBack encounters a hazarding condition at the HN-F. The reason for the hazard is a ReadShared transaction that is already in progress.

- The hazard detection results in the WriteBack being blocked.
- The ReadShared transaction receives data with the snoop response and must update memory in addition to sending the data to the Requester.

2. At Time B:
   - The WriteBack is unblocked because the HN-F has sent the data response to the Requester and a WriteData response to memory for the ReadShared transaction.

If the ReadShared request reaches the HN-F, after the HN-F has started processing the WriteBack request, the ReadShared request is blocked until completion of the WriteBack request.

A CopyBack request is completed at HN-F when both of the following are true:

- A data message corresponding to the CopyBack request is received.
- A memory update is completed if necessary.

B5.6.4 Race hazard

After completion, a request could silently evict the cache line from the cache and generate another request to the same address. For example:

1. The regenerated request reaches the HN-F before the CompAck response associated with the earlier request.
2. The HN-F detects an address hazard and blocks the processing of the new request until the CompAck response is received.

In such a scenario, on arrival at HN-F, the CompAck response deallocates the previous request from the HN-F and unblocks the processing of the new request.

Chapter B6
Exclusive accesses

This chapter describes the mechanisms that the architecture includes to support Exclusive accesses. It contains the following sections:

B6.1 Overview

The principles of Exclusive accesses are that a *Logical Processor* (LP) performing an exclusive sequence does the following:

- Performs an Exclusive Load from a location.
- Calculates a value to store to that location.
- Performs an Exclusive Store to the location.

Exclusive accesses are supported to Snoopable and Non-snoopable memory locations.

If the location is updated by a different LP since the Exclusive Load, the Exclusive Store must fail. In this case, the store does not occur and the LP does not update the value held at the location.

Table B6.1 shows the descriptions for Exclusive terms.

Table B6.1: Description of Exclusive terms

Term	Description of Exclusive terms
Exclusive Load	The action of an LP executing an appropriate program instruction, such as LDREX, requires: – Obtaining the data from the location to which the LP wants to perform an exclusive sequence. – Indicating that the LP is starting an exclusive sequence.
Exclusive Load transaction	A transaction issued on the interface to obtain data for an Exclusive Load, if the data is not available in the cache at the LP. Not every Exclusive Load requires an Exclusive Load transaction.
Exclusive Store	The action of an LP executing an appropriate program instruction, such as STREX, requires: – Determining if the exclusive sequence has passed or failed. – If appropriate, updating the data at the location. Can pass or fail and this result is known to the executing processor. When an Exclusive Store passes, the data value at the address location is updated. When an Exclusive Store fails, this indicates that the data value at the address location has not been updated, and the exclusive sequence must be restarted.
Exclusive Store transaction	A transaction issued on the interface that could be required to complete an Exclusive Store. Not every Exclusive Store requires an Exclusive Store transaction. An Exclusive Store transaction can pass or fail and this result is made known to the LP using the transaction response.

B6.2 Exclusive monitors

The progress of an exclusive sequence is tracked by an exclusive monitor. The location of the monitor and the request type generated to support the Exclusive accesses depends on the memory attributes of the address.

The attributes of Exclusive accesses must be such that they are guaranteed to be observed by an exclusive monitor. For example, if there is a cache between the Requester and monitor, the Exclusive accesses should be Non-cacheable.

B6.2.1 Snoopable memory location

For a Snoopable memory location two monitors are defined:

LP monitor Each LP within an RN-F must implement an exclusive monitor that observes the location used by an exclusive sequence. The LP monitor is set when the LP executes an Exclusive Load. The LP monitor is reset when either:

- The location is updated by another LP, indicated by an Invalidating snoop request to the same address.
- There is a store to the location by the same LP. If the store from the same LP is Non-exclusive, resetting the monitor is IMPLEMENTATION DEFINED.

PoC monitor An HN-F must implement a PoC monitor that can pass or fail an Exclusive Store transaction. A pass indicates that the transaction has been propagated to other coherent RN-Fs. A fail indicates that the transaction has not been propagated to other coherent RN-Fs and therefore the Exclusive Store cannot pass. The monitor is used to ensure an Exclusive Store transaction from an LP is only successful if the LP could not have received a snoop transaction relating to an Exclusive Store to the same address from another LP, after its own Exclusive Store transaction is issued. The minimum requirement of the PoC monitor is to record when any LP performs a Snoopable transaction related to an exclusive sequence. If an LP has performed a transaction related to an exclusive sequence, and subsequently performs an Exclusive Store transaction before a successful Exclusive Store transaction from another LP is scheduled, the Exclusive Store transaction must be successful. The monitor must support the parallel monitoring of all exclusive-capable Logical Processors in the system. When the HN-F receives a transaction associated with an Exclusive Load or an Exclusive Store, the monitor registers that the LP is attempting an exclusive sequence. When the HN-F receives an Exclusive Store transaction:

- If the PoC monitor has registered that the LP is performing an exclusive sequence, that is, the LP has not been reset by an Exclusive Store transaction from another LP, the Exclusive Store transaction is successful and is permitted to proceed. In such a case, registered attempts of all other LPs must be reset. It is recommended, but not required, that the PoC monitor for the successful LP is left as registered.
- If the PoC monitor has not registered that the LP is performing an exclusive sequence, that is, the LP has been reset by an Exclusive Store from another LP, the Exclusive Store transaction is failed and is not permitted to proceed. The monitor must register that the LP is attempting an Exclusive sequence.

Note

A successful Exclusive Store transaction from an LP does not have to reset that the LP is performing an exclusive sequence. The LP can continue to perform a sequence of Exclusive Store transactions, which are all successful, until another LP performs a successful Exclusive Store transaction. For store transactions in which the LP is not identifiable, the store must be handled as from a different LP than the one which set the monitor.

From initial system reset, the first LP to perform an Exclusive Store transaction can be successful, but it is not required. At that point, all other LPs must subsequently register the start of their exclusive sequence for their Exclusive Store transaction to be successful.

When an Exclusive Store transaction from one LP passes and the registered attempts of all other LPs is reset, the other LPs can only register a new exclusive sequence after the CompAck response is observed for the Exclusive Store transaction that passed.

Note

An LP and PoC monitor pair are required to support an Exclusive access to a Snoopable memory location.

B6.2.2 Additional address comparison

The PoC monitor can be enhanced to include some address comparison. A full address comparison is not required and it is permitted to only record a subset of address bits. This approach reduces the chances of an Exclusive Store transaction failing because of an Exclusive Store transaction from another LP to a different address location. The number of bits of address comparison used is IMPLEMENTATION DEFINED.

Where an additional address comparison monitor is used, the monitored address bits are recorded at the start of an exclusive sequence on either a Load Exclusive or Store Exclusive transaction. The monitor is reset by a successful Exclusive Store transaction from another LP to a matching address.

A monitor that includes an additional address comparison must still include a minimum monitor of a single bit for every Exclusive-capable LP to ensure forward progress.

An Exclusive Store transaction is permitted to progress if one of the following occurs:

- The address monitor has registered an exclusive sequence for a matching address from the same LP and has not been reset by an Exclusive Store transaction from a different LP with a matching address.
- The minimum single-bit monitor has been set by an exclusive sequence from the same LP and it has not been reset by an Exclusive Store transaction from a different LP to any address.

Note

The term matching address is used to describe where a monitor only records a subset of address bits. The address bits that are recorded are identical, but the address bits that are not recorded can be different.

An implementation does not require an address monitor for each Exclusive-capable LP. Because the address monitor provides a performance enhancement, it is acceptable to have fewer address monitors and for the use of these to be IMPLEMENTATION DEFINED. For example, address monitors can be used on a first-come first-served basis, or by allocation to particular Logical Processors. Alternatively, a more complex algorithm could be implemented.

Additional PoC exclusive monitor functionality can be provided to prevent interference, or denial of service, caused by one agent in the system issuing many Exclusive access transactions. It is recommended that the progress on Exclusive requests from one PAS is independent of progress on Exclusive requests from any other PAS.

B6.2.3 Alternatives to a PoC monitor

An HN-F is permitted to use the following mechanisms instead of a PoC monitor to determine the results of an Exclusive access:

- A precise Snoop Filter, to track if the Requester at the time of Exclusive Store processing retains a copy of the cache line.

- Snooping by the Home Node, to determine if the Requester still holds a copy of the cache line.

B6.2.4 Non-snoopable memory location

For a Non-snoopable memory location, a single monitor is used:

System monitor The System monitor tracks Exclusive accesses to a Non-snoopable region. This monitor type is set by a ReadNoSnp(Excl) transaction and reset by an update to the location by another LP. System monitors can be placed at a PoS or at endpoint devices. Potentially, the number of devices in the system is much larger than the number of PoS and placing System monitors at a PoS can:

- Reduce System monitor duplication.
- Reduce the time taken for the system to detect failure of an Exclusive access.

A System monitor must be located to observe all transactions to the monitored location.

B6.3 Exclusive transactions

The following transaction types support Exclusive accesses through an Excl bit:

- Exclusive Load transaction to a Snoopable location:
  - ReadClean
  - ReadNotSharedDirty
  - ReadShared
  - ReadPreferUnique
- Exclusive Store transaction to a Snoopable location:
  - CleanUnique
  - MakeReadUnique
- Exclusive Load transaction to a Non-snoopable location:
  - ReadNoSnp
- Exclusive Store transaction to a Non-snoopable location:
  - WriteNoSnp

The communicating node pairs are:

- For Exclusives to a Snoopable location:
  - RN-F to ICN(HN-F)
- For Exclusives to a Non-snoopable location:
  - RN-F, RN-D, RN-I to ICN(HN-F, HN-I)
  - ICN(HN-F) to SN-F
  - ICN(HN-I) to SN-I

An exclusive transaction must use the correct LPID value, see B2.4.7 *Logical Processor Identifier, LPID*.

B6.3.1 Responses to Exclusive requests

Transaction responses to Exclusive requests are similar to the normal responses to reads and writes with the following exceptions:

- ReadClean, ReadNotSharedDirty, and ReadShared Exclusive transactions:
  - Must not use separate Comp and data response.
  - Requests that do not fail must not use either DMT or DCT.
- ReadNoSnp Exclusive transactions that do not fail, must not use DMT.
- WriteNoSnpFull and WriteNoSnpPtl transactions, must not use DWT if the Exclusive monitor is located at the Home and the Exclusive check fails.

However, the response for the following Exclusive transactions must also indicate if the exclusive request has passed or failed:

- ReadClean
- ReadNotSharedDirty
- ReadShared
- ReadNoSnp
- CleanUnique
- WriteNoSnpFull
- WriteNoSnpPtl

The RespErr field in the response is used for this purpose. See B13.10.47 *Response Error, RespErr*. The RespErr field value of `0b01`, Exclusive Okay, indicates a pass and a RespErr field value of `0b00`, Normal Okay, indicates an Exclusive access failure.

The Exclusive Okay response must only be given for a transaction that has the Excl attribute set.

Not all memory locations are required to support Exclusive accesses. An Exclusive Load transaction to a location that does not support Exclusive accesses must not be given an Exclusive Okay response.

It is IMPLEMENTATION DEFINED whether or not Exclusive Store transaction to a location that does not support Exclusive accesses updates the location.

It is recommended that an Exclusive Store transaction is not performed to a location that does not support Exclusive accesses.

ReadPreferUnique and MakeReadUnique do not use RespErr to determine the pass or fail of an Exclusive operation. An Exclusive MakeReadUnique response with Shared cache state indicates the failure of an Exclusive access. When the response cache state is Unique, the Requester must use its Local Monitor state to determine if the Exclusive access is a pass.

Table B6.2 shows the Snoopable attributes of the request, the relevant monitor type and possible reasons for fail conditions and response requirements.

Table B6.2: Responses to an Exclusive access request

Request type	Snoopable	Monitor type	Fail condition	Response
ReadNoSnp(Excl)	No	System	Target does not support Exclusive accesses	Target must return a data response
WriteNoSnp(Excl)	No	System	Address content modified	The Requester must still complete the write flow by sending the Data message
			Address not present due to monitor overflow	
			Target does not support Exclusive accesses	
ReadClean(Excl) ReadNotSharedDirty(Excl) ReadShared(Excl)	Yes	LP, PoC	Target does not support Exclusive accesses	Target must return a data response
ReadPreferUnique	Yes	LP, optional PoC	None	If a PoC monitor is available, the appropriate monitor bit must be set
CleanUnique(Excl)	Yes	LP, PoC	Address content modified	Target must return a Comp response
			Address not present due to monitor overflow	
			Target does not support Exclusive accesses	
MakeReadUnique(Excl)	Yes	LP, optional PoC	Address content modified	Target uses PoC monitor, Precise snoop filter, or SnpQery to determine response

B6.3.1.1 MakeReadUnique(Excl)

When performing an Exclusive Store operation, MakeReadUnique(Excl) is the preferred alternative over the CleanUnique(Excl) request.

The permitted responses to the MakeReadUnique(Excl) request are included in Table B4.38 and Table B4.39.

B6.3.1.1.1 Result of an Exclusive access

The result of an Exclusive access is determined in the following manner:

- RespErr value of Exclusive Okay is not permitted in response to a MakeReadUnique(Excl).
- The Requester must use its local exclusive monitor along with the cache state in the response to determine the success of an Exclusive Store.
  - When the cache state in the response is Shared, the Requester must assume that the Exclusive access has failed.
  - When the cache state in the response is Unique, the Requester must use its own local monitor to determine the result of the Exclusive access.

B6.3.1.1.2 Home behavior

After checking the PoC monitor, if the Home determines the Exclusive Store passes, all other cached copies of the cache line must be invalidated. Subsequently, the Home sends a Comp response with the cache state of Unique to the Requester. The cache state can include [_PD] if a cached copy in SD state was invalidated and the responsibility of writing back of the Dirty data is being passed to the Requester.

To avoid an Exclusive access livelock, the monitor must be capable of simultaneously tracking each individual LP Exclusive thread in the system.

A Home is permitted to use a precise Snoop Filter to determine the success or failure of an Exclusive access. If the Snoop Filter indicates that the Requester has lost the cache line, the Exclusive access is assumed to have failed.

In the absence of precise caching information from the Snoop Filter, the Home can use the SnpQuery snoop to determine the presence and state of the cache line at the Requester.

When a Home determines that an Exclusive Store transaction has failed, the following rules must be followed:

- If the Requester has lost the cache line, the Home is expected to send SnpPreferUniqueFwd or SnpPreferUnique to get a copy of the cache line. The Home is permitted to send SnpNotSharedDirty(Fwd), or SnpClean(Fwd), or SnpShared instead. The snoop must not be SnpSharedFwd nor any Invalidating snoop.
- The sender of the data is permitted to return, as appropriate, a UC or UD state in the response to the Requester if no other cached copies exist. The Completer must return data with SC state, if other copies exist. SD state is not permitted in the response to the Requester. This is because the Home cannot determine from the MakeReadUnique(Excl) request whether the Requester accepts data in SD state.

Data is provided in response to a MakeReadUnique(Excl) transaction when a cache line is lost to a snoop between the issuing of the MakeReadUnique(Excl) transaction and the point at which the Home actions the transaction or the Home cannot determine that the data has been retained.

B6.3.2 System responsibilities

A system that implements the CHI protocol has the following responsibilities:

- Should include a monitor per LP for the efficient handling of Exclusive accesses.
- Must have a starvation prevention mechanism for all exclusive requests, whether using the monitor mechanism or some other means.

- It is recommended that progress on Exclusive requests from one PAS is independent of progress on Exclusive requests from any other PAS.

B6.3.3 Exclusive accesses to Snoopable locations

This section describes the behavior of an LP when performing Exclusive accesses to a Snoopable address location.

B6.3.3.1 Snoopable Exclusive Load

The LP starts an exclusive sequence with an Exclusive Load. The start of the exclusive sequence must set the LP exclusive monitor.

An LP wanting to perform an Exclusive access to a Snoopable location could already hold the cache line in its local cache:

- If the LP holds the cache line in a Unique state, it is permitted, but not recommended, that an Exclusive Load transaction is performed.
- If the LP holds the cache line in a Shared state, it is permitted, but not required, that an Exclusive Load transaction is performed.
- If the LP does not hold a copy of the cache line, it is recommended that the LP uses an Exclusive Load transaction to obtain the cache line. It is permitted to use ReadClean, ReadShared, ReadNotSharedDirty, or ReadPreferUnique without the Excl attribute asserted.

B6.3.3.2 Snoopable Exclusive Load to Snoopable Exclusive Store

Typically, after the execution of an Exclusive Load, an LP calculates a new value to store to the location before attempting the Exclusive Store.

It is not required that an LP always completes an Exclusive sequence. For example, the value obtained by the Exclusive Load can indicate that a semaphore is held by another LP and that the value cannot be changed until the semaphore is released by the other LP. Therefore, a new Exclusive sequence can be started with no attempt to complete the current Exclusive sequence.

During the time between the Exclusive Load and the Exclusive Store, the LP exclusive monitor must monitor the location to determine whether another LP could have updated the location.

B6.3.3.3 Snoopable Exclusive Store

An LP must not permit an Exclusive Store transaction to be in progress at the same time as any transaction that registers that it is performing an exclusive sequence. The LP must wait for all messages for any such transaction to be exchanged, or to receive a RetryAck response, before issuing an Exclusive Store transaction. The transactions that register that an LP is performing an exclusive sequence are:

- Exclusive Load transactions to any location.
- Exclusive Store transactions to any location.

When an LP executes an Exclusive Store the following behavior is required:

- If the LP exclusive monitor has been reset the Exclusive Store must fail and the LP must not issue an Exclusive Store transaction. The LP must restart the exclusive sequence.

Note

When the LP monitor has been reset, not issuing a transaction for an Exclusive Store that must eventually fail avoids unnecessary invalidation of other copies of the cache line.

- If the cache line is held in a Unique state and the LP exclusive monitor is set, the Exclusive Store has passed and the LP can update the location without issuing a transaction.
- If the cache line is held in a Shared state and the LP exclusive monitor is set, the LP must issue an Exclusive Store transaction. A CleanUnique or MakeReadUnique transaction with the Excl attribute asserted must be used. The LP exclusive monitor must continue to operate and check that the cache line is not updated while the CleanUnique or MakeReadUnique transaction is in progress. An Exclusive CleanUnique transaction response is handled in the following way. The transaction receives a Normal Okay or an Exclusive Okay response. If the transaction receives an Exclusive Okay response, this indicates that the transaction has passed and has completed invalidating all other copies of the cache line. After an exclusive transaction completes with an Exclusive Okay response, the LP must again check the LP exclusive monitor:
  - If the LP exclusive monitor is set, the Exclusive Store has passed and the update is performed.
  - If the LP exclusive monitor is not set, an update to the cache line is indicated to have occurred between the point that the Exclusive Store transaction was issued and the point of completion. The Exclusive Store must fail and the exclusive sequence must be restarted.
  - If the LP has not been able to track the exclusive nature of the cache line, because the cache line has been evicted, the Exclusive Store must fail and the exclusive sequence must be restarted.

If the Exclusive Store transaction receives a Normal Okay response, this indicates another LP has been permitted to progress a transaction associated with an Exclusive Store. The transaction associated with the Exclusive Store, from this LP, has failed and has not propagated to other Logical Processors in the system. When an Exclusive Store transaction completes with a Normal Okay response, the options are:

- The LP can fail the Exclusive Store and restart the exclusive sequence with or without checking the state of the cache line when the access completed.
- The LP can check the LP exclusive monitor, and if the LP exclusive monitor has been reset, the LP must fail the Exclusive Store and restart the exclusive sequence.
- The LP can check the LP exclusive monitor, and if the LP exclusive monitor is set, the LP can repeat the Exclusive Store transaction.

For handling of an Exclusive MakeReadUnique at the Home Node, see B6.3.1.1.2 *Home behavior*.

B6.3.4 Exclusive accesses to Non-snoopable locations

The following restrictions apply to Exclusive accesses to Non-snoopable locations:

- The address of an Exclusive access must be aligned to the total number of bytes in the transaction.
- The number of bytes to be transferred in an Exclusive access must be a legal data transfer size. This is 1 byte, 2 bytes, 4 bytes, 8 bytes, 16 bytes, 32 bytes, or 64 bytes.

Failure to observe these restrictions results in behavior that is UNPREDICTABLE.

For Exclusive read and Exclusive write transactions to be considered a pair, the following criteria must apply:

- The addresses of the Exclusive read and the Exclusive write must be identical.
- The value of the control signals, that is MemAttr and SnpAttr of the Exclusive read and the Exclusive write transaction, must be identical.
- The data size in the Exclusive read and the Exclusive write must be identical.
- The LPID value of the Exclusive read must match the LPID value of the Exclusive write transaction.

The minimum number of bytes to be monitored during an exclusive operation is defined by the transaction size. The System monitor can monitor a larger number of bytes, up to 64 bytes, which is the maximum size of an Exclusive access. However, this can result in a successful Exclusive access being indicated as failing because a neighboring byte was updated while the Exclusive access was in progress.

Multiple Exclusive transactions to Non-snoopable memory locations, either read or write, to the same or different addresses, from the same LP must not be outstanding at the same time.

If the Subordinate Node does not support Exclusive accesses, as indicated by an Exclusive Fail on the Exclusive ReadNoSnp, the write updates the location if the write is given an Exclusive Fail response.

If the Subordinate Node does support Exclusive accesses, as indicated by an Exclusive Pass on the Exclusive ReadNoSnp, the write does not update the location if the write is given an Exclusive Fail response.

Chapter B7
Cache Stashing

This chapter describes the cache stashing mechanism whereby data that is written from a Request Node can be installed in a peer cache. It contains the following sections:

B7.1 Overview

Cache stashing is a mechanism to install data within particular caches in a system. Cache stashing ensures that data is located close to its point of use, improving the system performance.

Cache stashing is permitted to Snoopable memory only.

The CHI protocol supports two main forms of cache stashing transaction:

WriteUniqueStash
: Write with stash hint. This is used when the cache in which the data should be allocated is known at the point in time that the data is written. A write with stash hint can be a [Full] or [Ptl] cache line write, and this affects the Snoop transactions that are used. See B7.2 *Write with Stash hint*.

StashOnce
: Independent stash request. This is used when the request to stash data into a particular cache is separated from the writing of the data. An independent Stash transaction can indicate if the cache line should be held in a Unique or Shared state by using a StashOnceUnique or StashOnceSepUnique, or a StashOnceShared or StashOnceSepShared transaction respectively, which corresponds to whether the next expected use of the cache line is for storing or for reading. See B7.3 *Independent Stash request*.

Both forms of cache stashing can target installation of data at different cache levels. The Stash target cache can be a peer cache, specified by using the peer cache target NodeID, or an LP cache within the peer node, if the peer node has multiple logical processors. The LP is identified by the LPID in the target cache field. See B7.4 *Stash target identifiers*.

The Cache stashing requests can also target the cache below the peer cache in the cache hierarchy, which can be an interconnect cache or a system cache. This is done by not specifying the peer cache NodeID. See B7.4.2 *Stash target not specified*.

In all cases of cache stashing, the Stashing is only a performance hint and it is permitted for the Stash request receiver to not perform the stashing behavior.

B7.1.1 Snoop requests and Data Pull

The following Snoop requests are used to notify a Stash target:

- SnpUniqueStash
- SnpMakeInvalidStash
- SnpStashUnique
- SnpStashShared

Table B7.1 shows the Snoop requests associated with each of the Stash requests.

Table B7.1: Stash request and the corresponding Snoop request

Stash request	Snoop request
WriteUniquePtlStash	SnpUniqueStash or SnpMakeInvalidStash[a]
WriteUniqueFullStash	SnpMakeInvalidStash
StashOnceUnique StashOnceSepUnique	SnpStashUnique

Continued on next page

Table B7.1 – *Continued from previous page*

Stash request	Snoop request
StashOnceShared StashOnceSepShared	SnpStashShared

[a] Possible if Home already has the latest copy of the line.

A Snoopee that receives a Stash Snoop request does one of the following:

- Provides a Snoop response that also acts as a Read request for the associated cache line. Including a Read request with Snoop response is referred to as a DataPull. Table B7.2 shows the type of Read request that is implied by a DataPull in the response to each Stash Snoop request.

Table B7.2: Snoop response with Data Pull and implied Read request

Snoop request	Implied Read request
SnpUniqueStash	ReadUnique
SnpMakeInvalidStash	ReadUnique
SnpStashUnique	ReadUnique
SnpStashShared	ReadNotSharedDirty

- Provides a Snoop response without a DataPull response so ignoring the cache stash hint.

The value of the DataPull field in the SnpResp and SnpRespData responses indicates if DataPull is requested. See B13.10.37 *Data Pull, DataPull* for legal values for DataPull.

The use of DataPull to complete a Snoop request with Stash is optional.

If the Snoopee is not able to support the DataPull transaction flow, the stash operation is permitted to be ignored.

B7.2 Write with Stash hint

The rules for sending and processing a WriteUniqueFullStash and WriteUniquePtlStash request at the Stash requester, the Home, and the Stash target node are as follows:

Requester responsibilities:

- Sends a WriteUniqueFullStash or WriteUniquePtlStash request depending on whether a full cache line or a partial cache line is to be written.
- The request is expected to include a Stash target.

Home responsibilities:

- Permitted to send a RetryAck response to a WriteUniqueStash request and follow the Retry transaction flow.
- Sends SnpUniqueStash to the identified Stash target.
- Sends SnpUnique to all other Requesters that share the cache line.
- Permitted to send SnpMakeInvalidStash and SnpMakeInvalid instead of SnpUniqueStash and SnpUnique respectively for WriteUniqueFullStash.
- Send Comp to the Requester after the coherency action is completed.
- Permitted to ignore the stash hint in the Write request and process the request as a regular WriteUnique.
- Handles a request without a Stash target in the manner described in B7.4.2 *Stash target not specified*.
- Permitted to use DMT to get data from SN-F to the Stash target in response to a DataPull request, when the data is neither available at Home nor obtained from any caches.
- Permitted to use separate Non-data and Data-only response to the Stash target in response to a DataPull request.

The Stash target responsibilities:

- The responses that include Data Pull are:
  - SnpResp_I_Read
  - SnpRespData_I_Read
  - SnpRespData_I_PD_Read
  - SnpRespDataPtl_I_PD_Read
- Must not request Data Pull if:
  - Snoop has an address hazard with an outstanding request.
  - The Stash target has an outstanding request to the same address that has received a DBIDRespOrd but has not completed.
- When requesting Data Pull:
  - The Stash target must guarantee the Read data is accepted without any structural or protocol dependencies that could result in deadlock.
  - The Read request is treated by Home as ReadUnique.
  - The Stash target must populate the DBID field in the response with the TxnID that is to be used by Home for the Read transaction. If the Snoop response with DataPull includes data, the DBID field value in all data packets must be the same.
- Permitted to ignore the Stash hint and handle the snoop as SnpUnique.

B7.3 Independent Stash request

The second mechanism for implementing cache stashing is to permit the Stash request to be sent separated in time from the writing of Stash data. Examples of when such a mechanism is useful are:

- When the data that is being written is not required by the target immediately. This delayed Stash avoids polluting the cache with data that is not used immediately.
- When the data is already in the system and the data has to be prefetched into caches.
- When the process using the data being written is not scheduled when the data is written, and therefore the precise target of the Stash data is not known until later.

In these cases, a Requester can use StashOnce or StashOnceSep requests to request Home or a peer node to fetch a cache line.

The rules for sending and processing an independent Stash request at the Stash requester, Home, and the Stash target are as follows:

Requester Node responsibilities:

- Sends StashOnceUnique or StashOnceSepUnique to Home if the stashed cache line is to be modified.
- Sends StashOnceShared or StashOnceSepShared to Home if the stashed cache line is not to be modified.
- Sends StashOnceSep only if the Requester is capable of handling the StashDone response.
- The StashOnce and StashOnceSep requests provide a Stash target when the data is to be stashed in a peer cache.
- The StashOnce and StashOnceSep requests do not provide a Stash target when the data is to be allocated in the next level cache.
- The Requester, upon receiving the Comp response, is permitted to release some request resources, while keeping track of the number of outstanding StashDone responses. This count of the number of outstanding StashDone responses can be done per Stash group, using the Requester defined Stash Group ID, specified by the StashGroupID field value.

Home Node responsibilities:

- Permitted to send a RetryAck response to a Stash request and follow the Retry transaction flow.
- Send a SnpStashUnique to the target RN-F for StashOnceUnique and StashOnceSepUnique.
- Send a SnpStashShared to the target RN-F for StashOnceShared and StashOnceSepShared.
- Permitted to not send a Snoop request in response to a Stash request.
- Must send a Comp response, even if the Stash request is abandoned.
- For the StashOnce, the Home must send a Comp only after establishing processing order for the received request that is guaranteeing that any request to the same address received later from any Requester is ordered behind this request. The Comp response must come from the PoC.
- For the StashOnceSep request, the Home must send a Comp only after establishing the guarantee that a RetryAck response will not be sent. The StashDone response must only be sent after establishing the guarantee that the request is ordered at the Home.
- Fetches the addressed cache line from memory into the shared system cache when a Stash request without a Stash target is received.
- Permitted to send Comp after receiving the Stash request, and before sending any SnpStash* or receiving the Snoop response.
- Send Comp with a Non-Invalid state, if the cache line is cached in the cache at the next level.

- Send Comp_I if the cache line is not cached, or it is not known if the cache line is cached, in the cache at the next level.
- Send a Comp_I response if either the cache look up at Home is a miss or Home did not look up the cache before responding.
- Permitted to use DMT to get data from SN-F to the Stash target in response to a Data Pull request.
- Permitted to use separate Non-data and Data-only response to the Stash target in response to a Data Pull request.

Stash target responsibilities:

- The snoop must not change the state of the cache line at the Stash target.
- The snoop is treated as a hint at the Stash target to obtain a copy of the cache line.
- Must not request DataPull if:
  - Snoop has an address hazard with an outstanding request.
  - Response is sent before performing a local cache lookup.
  - The snoop is SnpStashShared and the cache has a copy of the cache line.
  - The Stash target has an outstanding request to the same address that has received a DBIDRespOrd but not yet completed.
- When requesting DataPull:
  - The Stash target must guarantee the Read data is accepted without any structural or protocol dependencies that could result in deadlock.
  - The DataPull request is treated by Home as ReadNotSharedDirty for SnpOnceShared.
  - The DataPull request is treated by Home as ReadUnique for SnpOnceUnique.
  - The Home must treat the Stash request and the DataPull request atomically as a single request. That is, the Home must not order any other request to the same address between the Stash request and the corresponding DataPull request.
  - The Stash target must populate the DBID field in the response with the TxnID that is to be used by Home for the Read transaction.
- A DataPull request can be sent, but is not required to be sent, when the snoop is SnpStashUnique and a shared copy is present.
- The Stash target is permitted, but not required, to wait until the local cache lookup is complete before sending the Snoop response.
- The cache state in the Snoop response is not required to be precise:
  - An imprecise response must be SnpResp_I.
  - Any state other than I in the response must be precise.

Note

For StashOnce*, care is needed to avoid any action that could result in the deallocation of the cache line from the cache where it is expected to be used.

A StashOnce*Unique transaction can cause the invalidation of a copy of the cache line and care must be taken to ensure such transactions do not interfere with Exclusive access sequences.

The requirements regarding the sending of Stash type snoops from a Home, and the permitted responses by the target, are the same as the request/response rules for StashOnceSep transactions. See B2.3.4 *Stash transactions*.

B7.4 Stash target identifiers

For all Stash requests, both options of specified and non-specified Stash target are supported.

B7.4.1 Stash target specified

If the Stash target is available in the Stash request, Home sends the snoop with a stash hint to the specified target. The specified target can be a Request Node or an LP within a Request Node.

B7.4.2 Stash target not specified

The Home Node that receives a WriteUniquePtlStash or WriteUniqueFullStash request without a Stash target does the following:

- If the cache line is cached in a Unique state at a Request Node, the Home can treat that Request Node as the Stash target.
- If the cache line is not cached in a Unique state, the Home must only send SnpUnique as required, and must not send SnpUniqueStash to any Request Node.
- For WriteUniquePtlStash, if the cache line is not in any cache, it is recommended that he Home prefetches and allocates the cache line in the system cache. It is permitted, but not recommended, to perform a partial write to main memory.
- For WriteUniqueFullStash, if the cache line is not in any cache, Home is permitted to allocate the cache line in the shared system cache.

The Home Node that receives a StashOnce or StashOnceSep request without a Stash target does the following:

- If the cache line is not cached in any peer cache, it is recommended that the cache line is allocated in the shared system cache.
- If the cache line is cached in a peer cache, it is IMPLEMENTATION DEFINED if a snoop is sent to transfer a copy of the cache line and allocate the cache line in the shared system cache. For StashOnceUnique and StashOnceSepUnique, it is IMPLEMENTATION DEFINED if all cached copies are invalidated before allocating the cache line in the shared system cache.

B7.5 Stash messages

Stash messages are classified as:

- Write requests:
  - WriteUniqueFullStash
  - WriteUniquePtlStash

See B4.2.3 *Write transactions*.

- Dataless requests:
  - StashOnceUnique
  - StashOnceSepUnique
  - StashOnceShared
  - StashOnceSepShared

See B4.2.2 *Dataless transactions*.

- Snoop requests:
  - SnpUniqueStash
  - SnpMakeInvalidStash
  - SnpStashUnique
  - SnpStashShared

See B4.3 *Snoop request types*.

- Stash responses:
  - Comp
  - StashDone
  - CompStashDone

See B2.3.4 *Stash transactions*.

B7.5.1 Supporting REQ packet fields

The fields defined in the REQ packet to support Stash requests are:

- StashNID, StashLPID
- StashNIDValid, StashLPIDValid

Table B7.3 shows the valid StashNIDValid and StashLPIDValid encodings.

Table B7.3: Valid StashNIDValid and StashLPIDValid encodings

StashNIDValid	StashLPIDValid	Comments
0	0	Stash target is not specified
0	1	Reserved
1	0	Only a target Request Node is specified
1	1	Both target Request Node and LPID are specified

See B13.10 *Protocol flit fields*.

B7.5.2 Supporting SNP packet fields

The fields defined in the SNP packet to support Stash requests are:

- StashLPID
- StashLPIDValid

See B13.10 *Protocol flit fields*.

B7.5.3 Supporting RSP packet field

The field defined in the RSP packet to support Stash requests is DataPull.

See B13.10 *Protocol flit fields*.

B7.5.4 Supporting DAT packet field

The field defined in the DAT packet to support Stash requests is DataPull.

See B13.10 *Protocol flit fields*.

Chapter B8
DVM Operations

This chapter describes *Distributed Virtual Memory* (DVM) operations that the protocol uses to manage virtual memory. It contains the following sections:

B8.1 Introduction to DVM transactions

DVM transactions are an optional feature used to pass messages that support the maintenance of a virtual memory system.

DVM transactions support the following operations:

- Non-sync transaction flow, including:
  - TLB Invalidate
  - Branch Predictor Invalidate
  - Physical Instruction Cache Invalidate
  - Virtual Instruction Cache Invalidate
- Sync transaction flow, including:
  - Synchronization

DVM transactions only operate on read-only structures, such as Instruction cache, Branch Predictor, and TLB, and therefore only invalidation operations are required. The concept of cleaning does not apply to a read-only structure. This means that it is functionally correct to invalidate more entries than the DVM message requires, although the extra invalidations can affect performance.

Support for DVM operations on an interface is defined by the DVM_Support property. See B16.1.19 *DVM_Support* for more information.

B8.2 DVM transaction flow

The following sections describe the Non-sync and Sync DVM transaction flows and flow control:

- B8.2.1 *Non-sync type DVM transaction flow*
- B8.2.2 *Sync type DVM transaction flow*
- B8.2.3 *Flow control*

B8.2.1 Non-sync type DVM transaction flow

Figure B8.1 shows the steps in a Non-sync type DVM transaction.

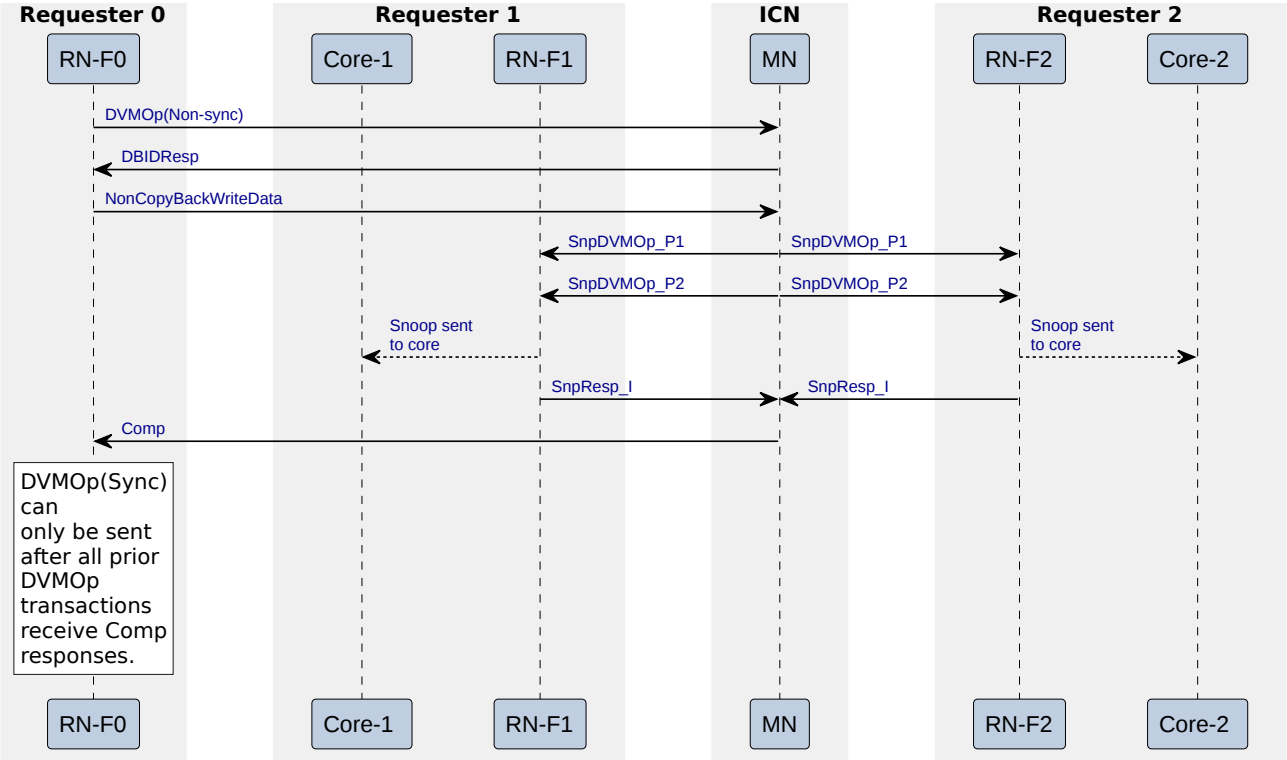

Figure B8.1: Non-sync type DVM transaction flow

The required steps that Figure B8.1 shows are:

1. RN-F0 sends a DVMOp(Non-sync) to the Miscellaneous Node using the appropriate write semantics for the DVMType.
2. The Miscellaneous Node accepts the DVMOp(Non-sync) request and provides a DBIDResp response.
3. The RN-F0 sends an 8-byte data packet on the data channel.
4. The MN broadcasts the SnpDVMOp snoop request to the remaining RN-F and RN-D nodes in the system. The MN is permitted, but not required, to send the SnpDVMOp to the RN that issued the original DVMOp. The SnpDVMOp is sent on the Snoop channel, and requires two Snoop requests. The two parts of the SnpDVMOp are labeled by the suffix _P1 and _P2.

> **Note**
>
> Both parts of the message must carry the same TxnID. The Request Node must have resources available to accept the SnpDVMOp. See B8.2.3 *Flow control*.

5. After completing the required actions, each recipient of the SnpDVMOp sends a single SnpResp response to the Miscellaneous Node.

> **Note**
>
> Sending of a SnpResp implies that the target Request Node has forwarded the SnpDVMOp to the required Request Node structures and has freed up the resources needed to accept another DVM operation. Sending of a SnpResp does not imply that the requested DVM operation has completed. See B8.2.2 *Sync type DVM transaction flow*.

6. After receiving all the SnpResp responses, the Miscellaneous Node sends a Comp response to the requesting node.

B8.2.1.1 DVM early Comp for Non-sync DVMOps

The Miscellaneous Node in the interconnect is permitted to send Comp for a Non-sync DVMOp without waiting to complete the required snooping of Request Nodes. Such a Miscellaneous Node must take the responsibility of ordering this Non-sync DVMOp against any Sync DVMOp received later from the same source. If a Miscellaneous Node cannot provide such an ordering guarantee, snooping must be complete before sending a Comp response for a Non-sync DVMOp.

A Miscellaneous Node that is enabled to send early Comp for a Non-sync DVMOp is permitted to opportunistically combine Comp and DBIDResp responses into a single CompDBIDResp response.

> **Note**
>
> Sending of a Comp response early for Non-sync DVMOp reduces round-trip latency for DVMOp completion. This enables a greater number of DVMOp transactions to be pipelined from a single source.
>
> Such an early completion also enables a Sync DVMOp, which is waiting for completions of all related DVMOp transactions sent earlier from the same Request Node.

The Miscellaneous Node must still wait for the Snoop response for a Sync DVMOp to be received before sending a Comp to the Requester for that Sync DVMOp.

B8.2.2 Sync type DVM transaction flow

Figure B8.2 shows the flow in a Sync DVM transaction.

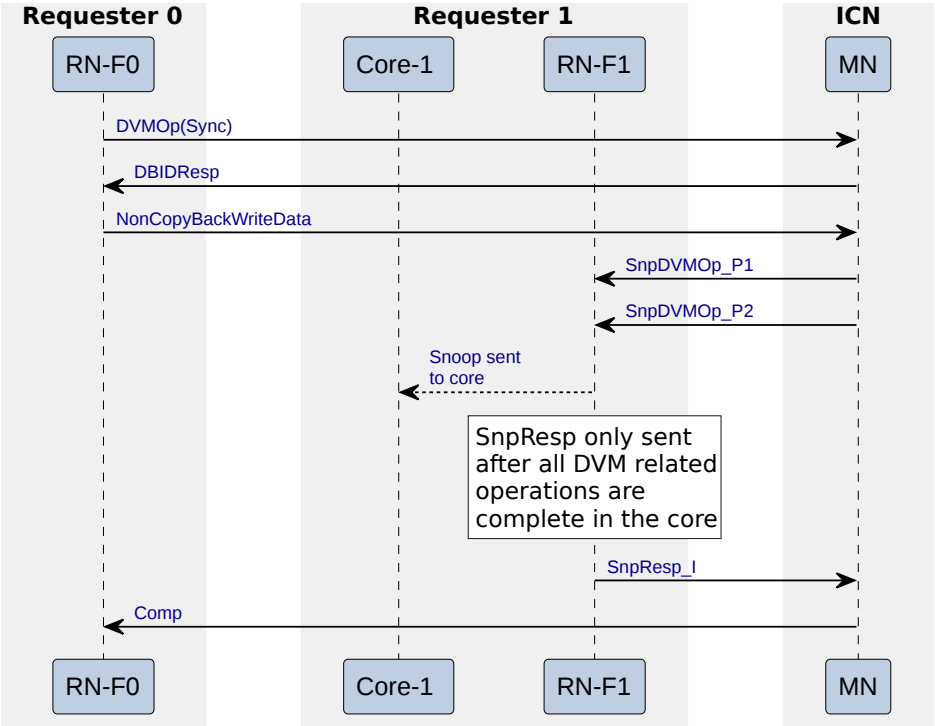

Figure B8.2: Sync DVM transaction flow

The required steps that Figure B8.2 shows are:

1. RN-F0 sends a DVMOp(Sync) to the Miscellaneous Node.

> **Note**
>
> All previous DVMOp requests whose completion needs to be guaranteed by the DVMOp(Sync) must have received a Comp response before the Request Node can send a DVMOp(Sync).

2. The Miscellaneous Node accepts the DVMOp(Sync) request and sends a DBIDResp response to the Requester.
3. The RN-F0 sends a data packet on the data channel with a data size of 8 bytes.
4. The Miscellaneous Node sends the SnpDVMOp to RN-F1. The SnpDVMOp is sent on the Snoop channel, and requires two Snoop requests. The two parts of a SnpDVMOp are labeled by the suffix _P1 and _P2.
5. After completing the DVM Sync operation, RN-F1 sends a SnpResp response to the Miscellaneous Node.

> **Note**
>
> Sending of a SnpResp implies that all DVM-related operations have completed at the Request Node structures and the target Request Node has freed up the resources needed to accept another SnpDVMOp.

6. After receiving the SnpResp, the Miscellaneous Node sends a Comp response to RN-F0.

B8.2.3 Flow control

This section describes the flow requirements for DVMOp and SnpDVMOp.

B8.2.3.1 DVMOp

The flow requirements for a DVMOp include:

- A DVMOp can receive a RetryAck response from a Miscellaneous Node.
  - A DVMOp that receives a RetryAck response must wait for a PCrdGrant response from the Miscellaneous Node that has the appropriate PCrdType.
- All previous DVMOp requests whose completion needs to be guaranteed by the DVMOp(Sync) must have received a Comp response before the Request Node can send the DVMOp(Sync).
- For DVMOp(Non-sync) operations, the interconnect must guarantee forward progress. This requires that there should be at least one tracker entry in the Miscellaneous Node reserved for DVMOp(Non-Sync).
- It is permitted to overlap a DVMOp(Non-sync) and a DVMOp(Sync) from the same Request Node, if the DVMOp(Sync) is not required to guarantee completion of the DMVOp(Non-sync).

B8.2.3.2 SnpDVMOp

The flow requirements for a SnpDVMOp include:

- Each SnpDVMOp transaction requires two SnpDVMOp request packets.
  - Both SnpDVMOp request packets corresponding to a single transaction must use the same TxnID.
  - The two SnpDVMOp request packets corresponding to a single transaction can be sent or received in any order.
  - To prevent deadlocks, due to the two-part SnpDVMOp requests that use the Snoop channel, a SnpDVMOp transaction must only be sent when the receiving Request Node has pre-allocated resources to accept both parts of the SnpDVMOp transaction.
  - A Request Node must provide a response to a SnpDVMOp transaction only after both SnpDVMOp request packets are received corresponding to that transaction.
  - A Request Node must provide a response to a SnpDVMOp only when a further SnpDVMOp from a Miscellaneous Node can be accepted.
- Each RN-F and RN-D in the system specifies the number of SnpDVMOp transactions it can accept concurrently.
  - Each RN-F and RN-D in the system must be able to accept at least one SnpDVMOp(Non-Sync) transaction in addition to a SnpDVMOp(Sync) transaction.
  - The minimum number of SnpDVMOp transactions that must be accepted concurrently is two. This is the default number for Request Nodes that do not specify a number.
- A SnpDVMOp(Sync) operation at a Request Node that has issued a StashOnceSep request must wait until all outstanding StashDone responses are received for StashOnceSep requests that use invalidated page entries.

For SnpDVMOp(Non-sync):

- Multiple SnpDVMOp(Non-sync) transactions can be outstanding from a Miscellaneous Node.

For SnpDVMOp(Sync):

- Only one SnpDVMOp(Sync) can be outstanding from a Miscellaneous Node to a Request Node.
- When responding to a SnpDVMOp(Sync), a Request Node can depend on the forward progress of any transaction, except for the completion of an outstanding DVMOp(Sync) request.

B8.3 DVMOp field value restrictions

Field value restrictions during a DVMOp transaction are shown:

- Request messages in Table B8.1
- Response messages, DBIDResp, Comp, and SnpResp in Table B8.2
- Snoop messages in Table B8.3
- Data messages in Table B8.4

B8.3.1 Request DVMOp field value restrictions

Table B8.1 shows the Request message field value restrictions for a DVMOp transaction.

Table B8.1: Request message field value restrictions for DVMOp

Field name	Restriction
QoS	None. Can take any value.
TgtID	Expected to be node ID of Miscellaneous Node. Can be remapped to correct TgtID by the interconnect.
SrcID	Source ID of the Requester that initiated the DVM message.
TxnID	An ID generated by the Requester. Must follow the same rules as any other transaction.
ReturnNID StashNID SLCRepHint	Must be all zeros.
StashNIDValid Endian Deep	Must be 0.
ReturnTxnID StashLPIDValid StashLPID	Must be all zeros.
Opcode	Must be DVMOp.
Size	Must be 8-byte.
Addr	Req_Addr_Width. See DVM Operations.
NS	Must be 0.
NSE	Must be 0.
LikelyShared	Must be 0.
AllowRetry	Can take any value, because a DVMOp can be given a Retry.

Continued on next page

Table B8.1 – *Continued from previous page*

Field name	Restriction
Order	Must be all zeros.
PCrdType	Must be all zeros if AllowRetry is asserted, otherwise the Credit type value.
MemAttr	Must be all zeros.
SnpAttr	None. Can take any value. See DVM domain.
DoDWT	
PGroupID	Not applicable. Must be all zeros.
StashGroupID	
TagGroupID	
LPID	None. Can take any value.
Excl	Must be 0.
SnoopMe	
CAH	
ExpCompAck	Must be 0.
TagOp	Must be 0.
TraceTag	None.
MPAM	Must be all zeros.
PBHA	Must be all zeros.
RSVDC	None. Can take any value.

B8.3.2 Response DVMOp field value restrictions

Table B8.2 shows the Response DBIDResp, SnpResp, and Comp and CompDBIDResp message field value restrictions during a DVMOp transaction.

Table B8.2: Restrictions for response message field values during DVMOp

Field	DBIDResp messages	Comp and CompDBIDResp messages	SnpResp messages
QoS	None. Can take any value.	None. Can take any value.	None. Can take any value.
TgtID	Must be ID of the original Requester	Must be ID of the original Requester	Must be ID of the Miscellaneous Node that is handling DVMOps
SrcID	Must be ID of the Miscellaneous Node that is handling DVMOps.	Must be ID of the Miscellaneous Node that is handling DVMOps.	Must be the node ID that is responding to snoops
TxnID	Must match TxnID of the original request	Must match TxnID of the original request	Must match the TxnID of the SnpDVMOp snoop request

Continued on next page

Table B8.2 – *Continued from previous page*

Field	DBIDResp messages	Comp and CompDBIDResp messages	SnpResp messages
Opcode	Must be DBIDResp	Must be Comp or CompDBIDResp	Must be SnpResp
RespErr	Must be all zeros	Must be `0b00`, `0b10` or `0b11`	Must be `0b00` or `0b11`
Resp	Must be all zeros	Must be all zeros	Must be all zeros
FwdState DataPull	Must be all zeros	Must be all zeros	Must be all zeros
CBusy	Expected to be populated by the Completer	Expected to be populated by the Completer	Expected to be populated by the Completer
DBID PGroupID StashGroupID TagGroupID	Generated by the Miscellaneous Node that is handling DVMOps	Generated by the Miscellaneous Node that is handling DVMOps	None. Can take any value.
PCrdType	Must be all zeros	Must be all zeros	Must be all zeros
TagOp	Must be all zeros	Must be all zeros	Must be all zeros
TraceTag	None	None	None

B8.3.3 Snoop DVMOp field value restrictions

Table B8.3 shows the Snoop message field (SnpDVMOp) value restrictions during a DVMOp transaction.

Table B8.3: Snoop message field value restrictions for DVMOp

Field name	Restriction
QoS	None. Can take any value.
SrcID	Must be node ID of Miscellaneous Node.
TxnID	An ID generated by Miscellaneous Node.
FwdNID PBHA	None. Used as Range and Num[4:0] fields. See Table B8.14.
FwdTxnID StashLPIDValid StashLPID	Not applicable. Must be 0.
VMIDExt	Must be used for VMID[15:8] in Part 1 of the SnpDVMOp request. None, can take any value in Part 2 of the SnpDVMOp request.
Opcode	Must be SnpDVMOp.

Continued on next page

Table B8.3 – *Continued from previous page*

Field name	Restriction
Addr	(Req_Addr_Width) - 3. See DVM Operations.
NS	Must be 0.
NSE	Must be 0.
DoNotGoToSD	Must be 0.
RetToSrc	Must be 0.
TraceTag	None.
MPAM	Must be all zeros.

Both SnpDVMOp request packets, corresponding to a single DVMOp, must have the same value in the following fields:

- TxnID
- Opcode
- SrcID

B8.3.4 Data DVMOp field value restrictions

Table B8.4 shows the Data message field value restrictions for a DVMOp transaction.

Table B8.4: Data message field value restrictions for DVMOp

Field name	Restriction
QoS	None. Can take any value.
TgtID	Must be the same as SrcID returned in the DBIDResp response.
SrcID	Must be ID of the original Requester.
TxnID	Must be the same as DBID of the DBIDResp response.
HomeNID PBHA	Must be all zeros.
Opcode	Must be NonCopyBackWriteData.
RespErr	Must be `0b00` or `0b10`.
Resp	Must be all zeros.
FwdState DataSource	Must be all zeros.
DataPull	Must be 0.
CBusy	Must be all zeros.

Continued on next page

Table B8.4 – *Continued from previous page*

Field name	Restriction
DBID	None. Can take any value.
CCID	Must be all zeros.
DataID	Must be all zeros.
TagOp	Must be all zeros.
Tag	Must be 0.
TU	Must be 0.
TraceTag	None.
CAH	Must be 0.
RSVDC	None.
BE	Only BE[7:0] must be asserted.
Data	Unused bits must be 0 for Data[63:0] and Data [n:64] = Can take any value.
DataCheck	Must be the appropriate value for the Data field.
Poison	None. Can take any value.

B8.4 DVM messages

This section provides additional information on the various DVM messages, their associated fields, and flit packing. It contains the following subsections:

B8.4.1 DVM message payload

The payload of a DVM operation from the Request Node to the Miscellaneous Node is distributed within:

- The Addr field in the DVM request from the Request Node.
- The lower 8 bytes of Data in the NonCopyBackWriteData packet.

The payload of a DVM operation from the Miscellaneous Node to the Request Node is distributed over two SnpDVMOp request packets using the Addr fields.

It is recommended, but not required, that the payload of a SnpDVMOp message matches the payload of the DVMOp message from which it originates.

Table B8.5 shows the various fields in the payload and their encodings.

Table B8.5: DVMOp fields and encodings

Field	Bits	Function
AddrV	1	Indicates if the message includes an address. `0b0` No address included `0b1` Address included
VIV	2	*Virtual Index* (VI) Valid. `0b00` VI not valid `0b01` Reserved `0b10` Reserved `0b11` VI is valid
VMIDV	1	`0b1` indicates that the *Virtual Machine Identifer* (VMID) is valid.
ASIDV	1	`0b1` indicates that the *Address Space Identifer* (ASID) is valid.
Security	2	Indicates which Security state the invalidation applies to. See Table B8.8 for the encodings for each DVMType.

Continued on next page

Table B8.5 – *Continued from previous page*

Field	Bits	Function
Exception	2	Indicates that the transaction applies to: `0b00` Hypervisor and all Guest OS `0b01` EL3[a] `0b10` Guest OS `0b11` Hypervisor
DVMType	3	Indicates the DVM operation type as: `0b000` TLB Invalidate (TLBI) `0b001` Branch Predictor Invalidate (BPI) `0b010` Physical Instruction Cache Invalidate (PICI) `0b011` Virtual Instruction Cache Invalidate (VICI) `0b100` Synchronization `0b101-0b111` Reserved
VMID	8	Virtual Machine Identifier VMID[7:0]
ASID	16	Address Space Identifier
Stage	2	Indicates Stage 1 or Stage 1 invalidation: `0b00` DVMv7: Any transaction DVMv8: Both Stage 1 and Stage 2 invalidation `0b01` Stage 1 invalidation only `0b10` Stage 2 invalidation only `0b11` *Granule Protection Table*(GPT)
Leaf	1	Indicates whether only leaf entries are invalidated: `0b0` Invalidate all associated translations. `0b1` Invalidate Leaf Entry only.
Range[b]	1	Range can take any value. When: **Range = 1** The transaction is a range-based TLBI operation. **Range = 0** The transaction is a non-range based TLBI operation. Range is inapplicable and must be zero for non-TLBI DVM transactions.
Num[b]	5	Used as a constant multiplication factor in the range calculation. All binary values are valid.
Scale[b]	2	Used as a constant in address range exponent calculation. All binary values are valid.
TTL[b]	2	Hint of *Translation Table Level* (TTL) which includes the addresses to be invalidated. See Table B8.6 and Table B8.7 for details.

Continued on next page

Table B8.5 – *Continued from previous page*

Field	Bits	Function
TG[b]	2	*Translation Granule* (TG) For non-range TLB Invalidations, TG and TTL indicate the table level hint, see Table B8.7. For TLB Invalidations by range, TG indicates the granule size: `0b00` Reserved `0b01` 4K `0b10` 16K `0b11` 64K
BaseAddr	37-41	Shifted base address of the range, shifted based on TG: **4K** BaseAddr is VA[MaxVA:12] **16K** BaseAddr is VA[MaxVA:14], VA[13:12] must be set to zero **64K** BaseAddr is VA[MaxVA:16], VA[15:12] must be set to zero
VA or	49-53	Virtual Address
PA	44-52	Physical Address
VI	16	Virtual Index
VMIDExt	8	Virtual Machine Identifier VMID[15:8]

Continued on next page

Table B8.5 – *Continued from previous page*

Field	Bits	Function	
IS[c]	4	*Invalidation Size* (IS) encoding for GPT TLBI by PA operations:	
		0b0000	4K
		0b0001	16K
		0b0010	64K
		0b0011	2MB
		0b0100	32MB
		0b0101	512MB
		0b0110	1GB
		0b0111	16GB
		0b1000	64GB
		0b1001	512GB
		0b1010-0b1111	Reserved

[a] DVMv8 only.

[b] Inapplicable and must be set to 0 in non-TLBI DVM operations.
See B8.4.3.1 *TLB Invalidate by Range* and B8.4.3.2 *Level Hint in TLBI operations* for details of how these fields are used in TLBI operations.

[c] Inapplicable and must be set to 0 in non-GPT TLBI by PA DVM operations. See B8.4.3.3 *Invalidation Size in GPT TLBI by PA operations*.

TTL and TG fields

For TLB Invalidations by address range, the TTL field can indicate which level of translation table walk holds the leaf entry for the address being invalidated. The encodings are shown in Table B8.6.

Table B8.6: Leaf entry hint for range-based TLB Invalidations

TTL	Meaning
0b00	No level hint information.
0b01	The leaf entry is on level 1 of the translation table walk.
0b10	The leaf entry is on level 2 of the translation table walk.
0b11	The leaf entry is on level 3 of the translation table walk.

For TLB Invalidations by non-range address, the TTL and TG fields indicate which level of translation table walk holds the leaf entry for the address being invalidated.

The encodings are shown in Table B8.7.

Table B8.7: Leaf entry hint for non-range TLB Invalidations

TG	TTL	Meaning
0b00	0b00	No level hint
	0b01	Reserved
	0b10	Reserved
	0b11	Reserved
0b01	0b00	The leaf entry is on level 0 of the translation table walk.
	0b01	The leaf entry is on level 1 of the translation table walk.
	0b10	The leaf entry is on level 2 of the translation table walk.
	0b11	The leaf entry is on level 3 of the translation table walk.
0b10	0b00	Reserved
	0b01	The leaf entry is on level 1 of the translation table walk.
	0b10	The leaf entry is on level 2 of the translation table walk.
	0b11	The leaf entry is on level 3 of the translation table walk.
0b11	0b00	Reserved
	0b01	The leaf entry is on level 1 of the translation table walk.
	0b10	The leaf entry is on level 2 of the translation table walk.
	0b11	The leaf entry is on level 3 of the translation table walk.

Security field

The Security field has different meanings depending on the DVMType, as shown in Table B8.8.

Table B8.8: Security field encodings for each DVMType

Security	TLBI	BPI	PICI		VICI
			Invalidation All	Invalidation by PA	
00	Realm	Secure and Non-secure	Root, Realm, Secure, and Non-secure	Root	Secure and Non-secure
01	Non-secure address from Secure context	Reserved	Realm and Non-secure	Realm	Reserved
10	Secure	Reserved	Secure and Non-secure	Secure	Secure
11	Non-secure	Reserved	Non-secure	Non-secure	Non-secure

ASID field

The ASID field contains an 8-bit or 16-bit Address Space Identifier.

- Armv7 supports an 8-bit ASID.
- From Armv8, an 8-bit and 16-bit ASID are supported.

It cannot be determined from a DVM message whether the message uses an 8-bit or 16-bit ASID.

All 8-bit ASID messages are required to set the ASID[15:8] bits to zero.

It is expected that most systems will use a single ASID size across the entire system, either 8-bit ASID or 16-bit ASID.

In a system that contains a mix of 8-bit ASID and 16-bit ASID components, it is expected that all maintenance is done by an agent that uses 16-bit ASID. This ensures that the agent can perform maintenance on both the 8-bit ASID and 16-bit ASID components.

The interoperability requirements include:

- For an 8-bit ASID agent sending a message to a 16-bit ASID agent, a message appears as a 16-bit ASID with the upper 8 bits set to zero.
- For a 16-bit ASID agent sending a message to an 8-bit VMID agent:
  - If the upper 8 bits are zero, the message was received correctly.
  - If the upper 8 bits are non-zero, over-invalidation occurs as the 8-bit ASID agent ignores the upper 8 bits.

VMID field

The VMID field contains an 8-bit or 16-bit Virtual Machine Identifier.

- Armv7 and Armv8 support an 8-bit VMID.
- From Armv8.1, an 8-bit and 16-bit VMID are supported.

It cannot be determined from a DVM message whether the message uses an 8-bit or 16-bit VMID.

All 8-bit VMID messages are required to set the VMID[15:8] field to zero.

It is expected that most systems use a single VMID size across the entire system, either 8-bit VMID or 16-bit VMID.

In a system that contains both 8-bit VMID and 16-bit VMID components, it is expected that all maintenance is done by an agent that uses 16-bit VMID. This ensures that the agent can perform maintenance on both the 8-bit VMID and 16-bit VMID components.

The interoperability requirements include:

- For an 8-bit VMID agent sending a message to a 16-bit VMID agent, a message appears as a 16-bit VMID with the upper 8 bits set to zero.
- For a 16-bit VMID agent sending a message to an 8-bit VMID agent:
  - If the upper 8 bits are zero, the message was received correctly.
  - If the upper 8 bits are non-zero, over-invalidation occurs as the 8-bit VMID agent ignores the upper 8 bits.

From Armv8.1, VMIDExt is used to transport the upper byte of 16-bit VMIDs.

DVM domain

The SnpAttr bit in a DVM request is used to differentiate between Inner and Outer domain.

Table B8.9 shows the SnpAttr value encoding in DVM transactions.

Table B8.9: SnpAttr value encoding in DVM transactions

SnpAttr	Domain value
0	Inner domain
1	Outer domain

Two additional optional interface broadcast pins are defined, **BROADCASTTLBIINNER** (BTI) and **BROARDCASTTLBIOUTER** (BTO). They determine broadcasting of TLBI operations in the interconnect. See B16.2 *Optional interface broadcast signals*.

B8.4.2 DVM message packing

Table B8.10 shows the distribution of the payload in the DVMOp request from the Request Node, using 8-byte write semantics, and the distribution of the payload in the SnpDVMOp requests from the Miscellaneous Node.

In the DVMOp, the combination of the address field in the request and the 8-byte write data transports the complete payload. REQ.Addr[3] is not used in the request and must be set to 0.

In the two SnpDVMOp requests the combination of the two address fields transports the complete payload. SNP.Addr[0] is used in a SnpDVMOp request to indicate which part of the payload is being transported.

The valid combinations of Maximum PA (MPA) and Maximum VA (MVA) address bits are:

- MPA = 44 : MVA = 49
- MPA = 45 : MVA = 51
- MPA = 46 to 52 : MVA = 53

See B8.4.3.1 *TLB Invalidate by Range* and B8.4.3.2 *Level Hint in TLBI operations*.

Table B8.10: Security field encodings for each DVMType

X in REQ.Addr[x] DAT.Data[x]	Num bits	DVMOp REQ	DVMOp DAT		Num bits	SnpDVMOp Request Part 1	SnpDVMOp Request Part 2
2:0	3	-	Num[2:0][a]	0	1	0	1
3	1	0	Num[3][a]				
4	1	AddrV	Scale[0] PA[6] VA[6]	1	1	AddrV	Scale[0] PA[6] VA[6]
5	1	VMIDV VIV[0]	Scale[1] PA[7] VA[7]	2	1	VMIDV newline VIV[0]	Scale[1] PA[7] VA[7]
6	1	ASIDV VIV[1]	IS[0] TTL[0] PA[8] VA[8]	3	1	ASIDV VIV[1]	IS[0] TTL[0] PA[8] VA[8]

Continued on next page

Table B8.10 – *Continued from previous page*

X in REQ.Addr[x] DAT.Data[x]	Num bits	DVMOp REQ	DVMOp DAT	X in SNP.Addr[x]	Num bits	SnpDVMOp Request Part 1	SnpDVMOp Request Part 2
7	1	Security[0]	IS[1] TTL[1] PA[9] VA[9]	4	1	Security[0]	IS[1] TTL[1] PA[9] VA[9]
8	1	Security[1]	IS[2] TG[0] PA[10] VA[10]	5	1	Security[1]	IS[2] TG[0] PA[10] VA[10]
9	1	Exception[0]	IS[3] TG[1] PA[11] VA[11]	6	1	Exception[0]	IS[3] TG[1] PA[11] VA[11]
10	1	Exception[1]	VA[12] PA[12]	7	1	Exception[1]	VA[12] PA[12]
13:11	3	DVMType[2:0]	VA[15:13] PA[15:13]	10:8	3	DVMType[2:0]	VA[15:13] PA[15:13]
21:14	8	VMID[7:0] VI[27:20]	VA[23:16] PA[23:16]	18:11	8	VMID[7:0]	VA[23:16] PA[23:16]
37:22	16	ASID[15:0] VI[19:12][b]	VA[39:24] PA[39:24]	34:19	16	ASID[15:0]	VA[39:24] PA[39:24]
39:38	2	Stage	VA[41:40] PA[6]	36:35	2	Stage	VA[41:40] PA[41:40]
40	1	Leaf	VA[42] PA[42]	37	1	Leaf	VA[42] PA[42]
41	1	Range[c]	VA[43] PA[43]	38	1	VA[46]	VA[43] PA[43]
42	1	Num[4][a]	VA[44] PA[44]	39	1	VA[47]	VA[44] PA[44]
43	1	-	VA[45] PA[45]	40	1	VA[48]	VA[45] PA[45]
44	1	-	VA[46] PA[46]	41	1	VA[50]	VA[49] PA[46]
45	1	-	VA[47] PA[47]	42	1	VA[52]	VA[51] PA[47]
48:46	3	-	VA[50:48] PA[50:48]	45:43	3	-	PA[50:48]
49	1	-	VA[51] PA[51]	46	1	-	PA[51]
50	1	-	VA[52]	47	1	-	-
55:51	5	-	-	48	1	-	-
63:56	8	-	VMID[15:8][d]				

[a] For Part 2 of a SnpDVMOp request, Num[4:0] from the corresponding DVMOp request is carried on the FwdNID field of the Snoop flit alongside the Addr field. See Table B8.14.

[b] When used as Virtual Index VA (VI VA), REQ.Addr[37:30], DAT.Data[37:30], and SNP.Addr[34:27] can take any value.

[c] For Part 1 of a SnpDVMOp request, Range from the corresponding DVMOp request is carried on the FwdNID field of the Snoop flit alongside the Addr field. See Table B8.14.

[d] For Part 1 of a SnpDVMOp request, VMID[15:8] from the corresponding DVMOp request is carried on the VMIDExt field of the Snoop flit alongside the Addr field. See Table B13.8.

B8.4.3 TLB Invalidate

This section details the TLB Invalidate (TLBI) message.

For a TLBI message, some fields have a fixed value, as shown in Table B8.11

Table B8.11: Fixed field values for a TLBI message

Field	Value	Status
DVMType	0b000	TLBI

The entries on which the TLBI must operate depends on the fields in the message.

Table B8.12 shows all the supported TLBI operations.

The Arm column indicates the minimum Arm architecture version required to support the message.

Table B8.12: TLBI messages

Operation	Arm	Exception	Security	VMIDV	ASIDV	Leaf	Stage	AddrV
EL3 TLBI all	v8	0b01	0b10	0b0	0b0	0b0	0b00	0b0
EL3 TLBI by VA	v8	0b01	0b10	0b0	0b0	0b0	0b00	0b1
EL3 TLBI by VA, Leaf only	v8	0b01	0b10	0b0	0b0	0b1	0b00	0b1
Secure Guest OS TLBI by Non-secure IPA	v8.4	0b10	0b01	0b1	0b0	0b0	0b10	0b1
Secure Guest OS TLBI by Non-secure IPA, Leaf only	v8.4	0b10	0b01	0b1	0b0	0b1	0b10	0b1
Secure TLBI all	v7	0b10	0b10	0b0	0b0	0b0	0b00	0b0
Secure TLBI by VA	v7	0b10	0b10	0b0	0b0	0b0	0b00	0b1
Secure TLBI by VA, Leaf only	v8	0b10	0b10	0b0	0b0	0b1	0b00	0b1
Secure TLBI by ASID	v7	0b10	0b10	0b0	0b1	0b0	0b00	0b0
Secure TLBI by ASID and VA	v7	0b10	0b10	0b0	0b1	0b0	0b00	0b1
Secure TLBI by ASID and VA, Leaf only	v8	0b10	0b10	0b0	0b1	0b1	0b00	0b1
Secure Guest OS TLBI all	v8.4	0b10	0b10	0b1	0b0	0b0	0b00	0b0
Secure Guest OS TLBI by VA	v8.4	0b10	0b10	0b1	0b0	0b0	0b00	0b1
Secure Guest OS TLBI all, Stage 1 only	v8.4	0b10	0b10	0b1	0b0	0b0	0b01	0b0
Secure Guest OS TLBI by Secure IPA	v8.4	0b10	0b10	0b1	0b0	0b0	0b10	0b1
Secure Guest OS TLBI by VA, Leaf only	v8.4	0b10	0b10	0b1	0b0	0b1	0b00	0b1
Secure Guest OS TLBI by Secure IPA, Leaf only	v8.4	0b10	0b10	0b1	0b0	0b1	0b10	0b1
Secure Guest OS TLBI by ASID	v8.4	0b10	0b10	0b1	0b1	0b0	0b00	0b0
Secure Guest OS TLBI by ASID and VA	v8.4	0b10	0b10	0b1	0b1	0b0	0b00	0b1

Continued on next page

Table B8.12 – *Continued from previous page*

Operation	Arm	Exception	Security	VMIDV	ASIDV	Leaf	Stage	AddrV
Secure Guest OS TLBI by ASID and VA, Leaf only	v8.4	0b10	0b10	0b1	0b1	0b1	0b00	0b1
All OS TLBI all	v7	0b10	0b11	0b0	0b0	0b0	0b00	0b0
Guest OS TLBI all, Stage 1 and 2	v7	0b10	0b11	0b1	0b0	0b0	0b00	0b0
Guest OS TLBI by VA	v7	0b10	0b11	0b1	0b0	0b0	0b00	0b1
Guest OS TLBI all, Stage 1 only	v8	0b10	0b11	0b1	0b0	0b0	0b01	0b0
Guest OS TLBI by IPA	v8	0b10	0b11	0b1	0b0	0b0	0b10	0b1
Guest OS TLBI by VA, Leaf only	v8	0b10	0b11	0b1	0b0	0b1	0b00	0b1
Guest OS TLBI by IPA, Leaf only	v8	0b10	0b11	0b1	0b0	0b1	0b10	0b1
Guest OS TLBI by ASID	v7	0b10	0b11	0b1	0b1	0b0	0b00	0b0
Guest OS TLBI by ASID and VA	v7	0b10	0b11	0b1	0b1	0b0	0b00	0b1
Guest OS TLBI by ASID and VA, Leaf only	v8	0b10	0b11	0b1	0b1	0b1	0b00	0b1
Secure Hypervisor TLBI all	v8.4	0b11	0b10	0b0	0b0	0b0	0b00	0b0
Secure Hypervisor TLBI by VA	v8.4	0b11	0b10	0b0	0b0	0b0	0b00	0b1
Secure Hypervisor TLBI by VA, Leaf only	v8.4	0b11	0b10	0b0	0b0	0b1	0b00	0b1
Secure Hypervisor TLBI by ASID	v8.4	0b11	0b10	0b0	0b1	0b0	0b00	0b0
Secure Hypervisor TLBI by ASID and VA	v8.4	0b11	0b10	0b0	0b1	0b0	0b00	0b1
Secure Hypervisor TLBI by ASID and VA, Leaf only	v8.4	0b11	0b10	0b0	0b1	0b1	0b00	0b1
Hypervisor TLBI all	v7	0b11	0b11	0b0	0b0	0b0	0b00	0b0
Hypervisor TLBI by VA	v7	0b11	0b11	0b0	0b0	0b0	0b00	0b1
Hypervisor TLBI by VA, Leaf only	v8	0b11	0b11	0b0	0b0	0b1	0b00	0b1
Hypervisor TLBI by ASID	v8.1	0b11	0b11	0b0	0b1	0b0	0b00	0b0
Hypervisor TLBI by ASID and VA	v8.1	0b11	0b11	0b0	0b1	0b0	0b00	0b1
Hypervisor TLBI by ASID and VA, Leaf only	v8.1	0b11	0b11	0b0	0b1	0b1	0b00	0b1
Realm TLBI all	v9.2	0b10	0b00	0b0	0b0	0b0	0b00	0b0
Realm Guest OS TLBI all, Stage 1 only	v9.2	0b10	0b00	0b1	0b0	0b0	0b01	0b0
Realm Guest OS TLBI all, Stage 1 and 2	v9.2	0b10	0b00	0b1	0b0	0b0	0b00	0b0
Realm Guest OS TLBI by VA	v9.2	0b10	0b00	0b1	0b0	0b0	0b00	0b1
Realm Guest OS TLBI by VA, Leaf only	v9.2	0b10	0b00	0b1	0b0	0b1	0b00	0b1
Realm Guest OS TLBI by ASID	v9.2	0b10	0b00	0b1	0b1	0b0	0b00	0b0
Realm Guest OS TLBI by ASID and VA	v9.2	0b10	0b00	0b1	0b1	0b0	0b00	0b1

Continued on next page

Table B8.12 – *Continued from previous page*

Operation	Arm	Exception	Security	VMIDV	ASIDV	Leaf	Stage	AddrV
Realm Guest OS TLBI by ASID and VA, Leaf only	v9.2	0b10	0b00	0b1	0b1	0b1	0b00	0b1
Realm Guest OS TLBI by IPA	v9.2	0b10	0b00	0b1	0b0	0b0	0b10	0b1
Realm Guest OS TLBI by IPA, Leaf only	v9.2	0b10	0b00	0b1	0b0	0b1	0b10	0b1
Realm Hypervisor TLBI all	v9.2	0b11	0b00	0b0	0b0	0b0	0b00	0b0
Realm Hypervisor TLBI by VA	v9.2	0b11	0b00	0b0	0b0	0b0	0b00	0b1
Realm Hypervisor TLBI by VA, Leaf only	v9.2	0b11	0b00	0b0	0b0	0b1	0b00	0b1
Realm Hypervisor TLBI by ASID	v9.2	0b11	0b00	0b0	0b1	0b0	0b00	0b0
Realm Hypervisor TLBI by ASID and VA	v9.2	0b11	0b00	0b0	0b1	0b0	0b00	0b1
Realm Hypervisor TLBI by ASID and VA, Leaf only	v9.2	0b11	0b00	0b0	0b1	0b1	0b00	0b1
GPT TLBI by PA	v9.2	0b01	0b10	0b0	0b0	0b0	0b11	0b1
GPT TLBI by PA, Leaf only	v9.2	0b01	0b10	0b0	0b0	0b1	0b11	0b1
GPT TLBI all	v9.2	0b01	0b10	0b0	0b0	0b0	0b11	0b0

B8.4.3.1 TLB Invalidate by Range

When DVM_Support is DVM_v8.4 or greater, TLBI operations by IPA or VA have the option to operate on an address range if the Range field is 0b1.

Range-based TLBI operations include range-specific payload in addition to the payload fields in non-range based TLBI operation.

B8.4.3.1.1 Range-based payload packing for TLBI by VA and IPA operations

The range-based fields for TLBI operations by VA and IPA are:

- Range
- BaseAddr
- TG
- TTL
- Scale
- Num

Note

The TG and TTL fields can be used as Level hints for Non-range based TLBI operations by VA or IPA. See B8.4.3.2 *Level Hint in TLBI operations*.

When the Range field is set to 1 for TLBI operations by VA or IPA, the address range to invalidate is calculated using the following formula, where Translation_Granule_Size in bytes is determined from the TG value, as shown in Table B8.13:

$$BaseAddr \leq AddressRange < BaseAddr + \Big((Num + 1) \times 2^{(5 \times Scale + 1)} \times Translation\_Granule\_Size \Big)$$

Table B8.13: TG field encodings in TLB instructions that apply to a range of addresses

TG	Translation granule size	Translation_Granule_Size
00	Reserved	Not applicable
01	4KB translation granule	4096
10	16KB translation granule	16384
11	64KB translation granule	65536

The range parameters are placed in the request packets as shown in Table B8.10.

The Range and Num values are carried on the FwdNID field in the Snoop flit alongside the Addr field for the corresponding Snoop transaction. Unused bits of FwdNID in SnpDVMOp must be set to 0.

Table B8.14 shows the placement of the Range and Num values in the SnpDVMOp payload.

Table B8.14: Num value placement in SnpDVMOp payload

FwdNID bit	SnpDVMOp request	
	Part 1	Part 2
0	Range	Num[0]
1	0	Num[1]
2	0	Num[2]
3	0	Num[3]
4	0	Num[4]
5	0	0
6	0	0

B8.4.3.2 Level Hint in TLBI operations

Non-range based TLBI operations by VA or IPA operations are permitted to use TG and TTL fields. These operations use TG and TTL as an indication of the level of the page table walk that holds the leaf entry for the address that is being invalidated. For such operations, the following conditions apply:

- The Range field must be set to 0.
- The Num and Scale fields are inapplicable and must be set to 0.

B8.4.3.3 Invalidation Size in GPT TLBI by PA operations

GPT TLBI by PA operations perform range-based invalidation and invalidate TLB entries starting from the PA, within the range as specified in the IS field.

The range-based fields for GPT TLBI operations by PA are IS and PA. If the PA is not aligned to the IS value, no TLB entries are required to be invalidated.

The IS field is applicable only in GPT TLBI by PA operations.

The Range field is applicable and must be set to 1 for GPT TLBI by PA operations.

The Range field is applicable and must be set to 0 for GPT TLBI all operations.

B8.4.4 Branch Predictor Invalidate

This section describes the *Branch Predictor Invalidate* (BPI) operations.

The BPI message is used to invalidate virtual addresses from branch predictors.

The fixed field values for a BPI message are shown in Table B8.15.

Table B8.15: Fixed field values for a Branch Predictor Invalidate message

Field	Value	Status
DVMType	0b001	Branch Predictor Invalidate
Part Num	-	See Table B8.10
VMIDV	0b0	VMID field not valid
ASIDV	0b0	ASID field not valid
Security	0b00	Applies to both Secure and Non-secure
Exception	0b00	Applies to all Guest OS and Hypervisor
VMID VMIDExt	0xXX	VMID not specified
ASID	0xXXXX	ASID not specified
Stage	0b00	Reserved, set to 0
Leaf	0b0	Reserved, set to 0

Note

The use of BPI with a 16-bit ASID is not supported.

Table B8.16 shows the operations supported by BPI.

Table B8.16: Branch Predictor Invalidate operations

Operation	Arm	AddrV
Branch Predictor Invalidate all	v7	0b0
Branch Predictor Invalidate by VA	v7	0b1

B8.4.5 Instruction Cache Invalidate

Instruction caches can use either a PA or a VA to tag the data they contain. A system could contain a mixture of both forms of cache.

The DVM protocol includes instruction cache invalidation operations that use Physical Addresses and operations that use Virtual Addresses.

A component that receives DVM messages must support both forms of message, independent of the style of instruction cache implemented. In an instance where a message is received in a format that is not native to the cache type, over-invalidation might be required.

B8.4.5.1 Physical Instruction Cache Invalidate

This section describes the *Physical Instruction Cache Invalidate* (PICI) operations that the DVM message supports. This message type is also used for Instruction Caches which are *Virtually Indexed Physically Tagged* (VIPT).

The fixed field values for a PICI message are shown in Table B8.17.

Table B8.17: Fixed field values for a Physical Instruction Cache Invalidate message

Field	Value	Status
DVMType	0b010	Physical Instruction Cache Invalidate
Part Num	-	See Table B8.10
Exception	0b00	Applies to all Guest OS and Hypervisor
Stage	0b00	Reserved, set to 0
Leaf	0b0	Reserved, set to 0

All supported PICI operations are shown in Table B8.18.

Table B8.18: Physical Instruction Cache Invalidate operations

Operation	Arm	Security	VIV	AddrV
PICI all Root, Realm, Secure and Non-secure	v9.2	0b00	0b00	0b0
PICI by PA without Virtual Index, Root only	v9.2	0b00	0b00	0b1
PICI by PA with Virtual Index, Root only	v9.2	0b00	0b11	0b1
PICI all Realm and Non-secure	v9.2	0b01	0b00	0b0
PICI by PA without Virtual Index, Realm only	v9.2	0b01	0b00	0b1
PICI by PA with Virtual Index, Realm only	v9.2	0b01	0b11	0b1
PICI all Secure and Non-secure	v7	0b10	0b00	0b0
PICI by PA without Virtual Index, Secure only	v7	0b10	0b00	0b1
PICI by PA with Virtual Index, Secure only	v7	0b10	0b11	0b1
PICI all, Non-secure only	v7	0b11	0b00	0b0
PICI by PA without Virtual Index, Non-secure only	v7	0b11	0b00	0b1
PICI by PA with Virtual Index, Non-secure only	v7	0b11	0b11	0b1

Note

When VIV is 0b11, VI[19:12] is used as part of the PA.

B8.4.5.2 Virtual Instruction Cache Invalidate

This section describes the *Virtual Instruction Cache Invalidate* (VICI) operations.

The fixed field values for a VICI message are shown in Table B8.19.

Table B8.19: Fixed field values for a Virtual Instruction Cache Invalidatee message

Field	Value	Status
DVMType	0b011	Virtual Instruction Cache Invalidate
Part Num	-	See Table B8.10
Stage	0b00	Reserved, set to 0
Leaf	0b0	Reserved, set to 0

Table B8.20 shows the operations supported by VICI.

Table B8.20: Virtual Istruction Cache Invalidate operations

Operation	Arm	Exception	Security	VMIDV	ASIDV	AddV
Hypervisor and all Guest OS VICI all, Secure and Non-secure	v7	0b00	0b00	0b0	0b0	0b0
Hypervisor and all Guest OS VICI all, Non-secure only	v7	0b00	0b11	0b0	0b0	0b0
All Guest OS VICI by ASID and VA, Secure only	v7	0b10	0b10	0b0	0b1	0b1
All Guest OS VICI by VMID, Secure only	v8.4	0b10	0b10	0b1	0b0	0b0
All Guest OS VICI by ASID, VA and VMID, Secure only	v8.4	0b10	0b10	0b1	0b1	0b1
All Guest OS VICI by VMID, Non-secure only	v7	0b10	0b11	0b1	0b0	0b0
All Guest OS VICI by ASID, VA and VMID, Non-secure only	v7	0b10	0b11	0b1	0b1	0b1
Hypervisor VICI by VA, Non-secure only	v7	0b11	0b11	0b0	0b0	0b1
Hypervisor VICI by ASID and VA, Non-secure only	v8.1	0b11	0b11	0b0	0b1	0b1

B8.4.6 Synchronization

This section describes the DVMSync operation.

A *Synchronization* (Sync) message is used when the Requester needs to know when all previous invalidations are complete.

The fixed field values for a Sync message are shown in Table B8.21.

Table B8.21: Sync operation fixed values

Field	Value	Status
DVMType	0b100	Synchronization message
Part Num	-	See Table B8.10
AddrV	0b0	No address information
VMIDV	0b0	No VMID information

Continued on next page

Table B8.21 – *Continued from previous page*

Field	Value	Status
ASIDV	0b0	No ASID information
Security	0b00	Security information is N/A
Exception	0b00	Exception information is N/A
VMID	0xXX	VMID not specified
VMIDExt		
ASID	0xXXXX	ASID not specified
Stage	0b00	Stage information is N/A
Leaf	0b0	Leaf information is N/A

Chapter B9
Error Handling

This chapter describes the error handling requirements. It contains the following sections:

B9.1 Packet level

This section describes the errors at the packet level. It contains the following sections:

B9.1.1 Error types

There are two types of error reporting at packet level.

The packet level error reporting types are:

Data Error, DERR

Used when the correct address location has been accessed, but an error is detected within the data. Typically, this is used when data corruption has been detected by Error Correction Code (ECC) or a parity check. Data Error reporting is supported by the RespErr, Poison, and DataCheck fields in the DAT packet. When processing of a request received by Home that is required to be propagated to the Subordinate results in a DERR, the Home must not stop propagating the request to the Subordinate.

> **Note**
>
> An error in data being evicted from Home, or received in a Snoop response as a result of the request, are examples of the request resulting in a DERR.

Non-data Error, NDERR

Used when an error is detected that is not related to data corruption. This specification does not define all cases when this error type is reported. Typically, this error type is reported for:

- An attempt to access a location that does not exist.
- An illegal access, such as a write to a read-only location.
- An attempt to use a transaction type that is not supported.

Non-data Error reporting is supported by the RespErr field in the RSP and DAT packets. When processing of a request received by Home results in an NDERR, it is permitted, but not required, to propagate the request to the Subordinate. The Home is required to pass-back the NDERR in the response to the Requester.

B9.1.2 Error response fields

The RespErr field is used to indicate error conditions. The RespErr field is included in both response and data packets.

Table B9.1 shows the encoding of the RespErr field. See B6.3.1 *Responses to Exclusive requests* for more details on the Exclusive Okay response.

Table B9.1: Error response field encodings

RespErr[1:0]	Name	Description
0b00	OK	Normal Okay. Indicates that either: – The Normal access was successful. For WriteNoSnpDef, RespErr must be used in combination with Resp to establish the exact response. See Table B4.28. – The Exclusive access failed.
0b01	EXOK	Exclusive Okay. Indicates that either the read or write portion of an Exclusive access has been successful.
0b10	DERR	Data Error
0b11	NDERR	Non-data Error

A single transaction is not permitted to mix OK and EXOK responses.

A transaction with a Data response is required to include NDERR either in none or in all Data packets.

The mixing of OK and DERR responses within a single transaction is permitted.

The mixing of EXOK and DERR responses within a single transaction is permitted.

The mixing of OK and NDERR responses within a single transaction is permitted, which can occur only in transactions with both Data and Non-data responses.

The mixing of EXOK and NDERR is not permitted.

B9.1.3 Errors and transaction structure

All transactions must complete in a protocol-compliant manner, even if they include an error response.

Error handling for a transaction that utilizes DMT is the same as the error handling for the same request without DMT.

Because there is no mechanism to propagate errors on requests or snoops, a request must not use DMT or DCT if an error is detected at the interconnect.

If the transaction contains data packets, the source of the data packets is required to send the correct number of packets, but the data values are not required to be valid.

The Resp field gives the cache states associated with a transaction and can be influenced by an error condition. See B4.5 *Response types* for more details on the legal Resp field values. In a response with a NDERR indication, the cache state encoded in Resp is permitted to be any value, including reserved values.

The Resp field in a response must have the same value for every packet of a Data message regardless of whether or not there is an error condition.

A Requester that receives a response with a NDERR for a Snoopable request must:

- For an Allocating transaction:
  - When the start state is I, the Requester must not allocate the received data.
  - If the request was sent from a Non-Invalid state, the Requester must leave the cached copy unchanged.

  In both cases, the cache state must not be changed.
  - The Allocating transactions are:

        - ReadClean
        - ReadNotSharedDirty
        - ReadShared
        - ReadUnique
        - ReadPreferUnique
        - MakeReadUnique
        - CleanUnique
        - MakeUnique

    - For a deallocating transaction:
        - The Requester must continue as normal, in a protocol-compliant manner.
        - The deallocating transactions are:
            - WriteBack
            - WriteEvictFull
            - Evict
            - WriteEvictOrEvict
    - For Other transactions that do not change allocation:
        - The Requester must not upgrade the cache state.
        - The Requester is permitted to downgrade the cache state.

The cache state in a SnpResp message with a NDERR must be I. The Completer must invalidate local cached copies of the cache line. In addition, when the response to a Forwarding snoop results in a NDERR, the Snoopee must not forward data to the Requester. As a consequence, if the CompData message has already been sent to the Requester, the Snoop response to the Home must not include the NDERR.

B9.1.4 Error response use by transaction type

This section defines the permitted, but not required, use of the error fields for each transaction type.

The tables that follow show the Data and Response packets associated with the following transaction types:

- B9.1.4.1 *Read transactions*
- B9.1.4.2 *Dataless transactions*
- B9.1.4.3 *Write transactions*
- B9.1.4.4 *Atomic transactions*
- B9.1.4.5 *Other transactions*
- B9.1.4.6 *Cache Stashing transactions*
- B9.1.4.7 *Snoop transactions*

The following keys are used by the tables:

OK The RespErr field must contain the OK RespErr value of `0b00`.

Y This value of RespErr is permitted.

N This value of RespErr is not permitted.

- Data or Response packet is not used for this transaction type.

B9.1.4.1 Read transactions

Read transactions can contain multiple CompData data packets.

Read data which is known to be corrupt must have an appropriate Error indication where the error can be Poison, DERR, or NDERR.

When RespSepData includes a NDERR, all corresponding DataSepResp packets must be marked with NDERR.

In a Data response to a Read request, a NDERR response is only permitted either in none or in all data response packets.

Table B9.2 shows the legal RespErr field values associated with Data and Response packets for Read transactions.

Table B9.2: Legal RespErr field values associated with Data and response packets for Read transactions

Read transaction	Associated Data and Response packets					CompAck
	Read Receipt	CompData				
		OK	EXOK	DERR	NDERR	
ReadNoSnp	OK	Y	Y	Y	Y	OK
ReadNoSnpSep	OK	-	-	-	-	-
ReadOnce ReadOnceCleanInvalid ReadOnceMakeInvalid	OK	Y	N	Y	Y	OK
ReadClean ReadNotSharedDirty ReadShared	-	Y	Y	Y	Y	OK
ReadUnique ReadPreferUnique[a] MakeReadUnique[b]	-	Y	N	Y	Y	OK

[a] EXOK response is not permitted even when Excl bit in the request is set to 1.
A response of OK in the returned data response is not to be taken as a failure of ReadPreferUnique.
Failure of the exclusive sequence is only determined from the corresponding MakeReadUnique $Excl$ or CleanUnique $Excl$ transaction completion.

[b] Applicable only when data is returned to the Requester. See Table B9.5 for permitted errors when data is not returned to the Requester.

Table B9.3 shows the legal RespErr field values associated with Data-only and Non-data packets for Read transactions.

Table B9.3: Legal RespErr field values associated with Data-only and Non-data packets for Read transactions

Read transaction	Associated Data-only and Non-data packets							
	DataSepResp				RespSepData			
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
ReadNoSnp	Y	N	Y	Y	Y	N	N	Y
ReadNoSnpSep	Y	N	Y	Y	-	-	-	-
ReadOnce ReadOnceCleanInvalid ReadOnceMakeInvalid	Y	N	Y	Y	Y	N	N	Y

Continued on next page

Table B9.3 – *Continued from previous page*

Read transaction	Associated Data-only and Non-data packets							
	DataSepResp				RespSepData			
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
ReadClean ReadNotSharedDirty ReadShared	Y	N	Y	Y	Y	N	N	Y
ReadUnique ReadPreferUnique[a] MakeReadUnique[b]	Y	N	Y	Y	Y	N	N	Y

[a] A response of OK in the returned data response is not be taken as a failure of ReadPreferUnique.
Failure of the exclusive sequence is only determined from the corresponding MakeReadUnique $Excl$ or CleanUnique $Excl$ transaction completion.

[b] Applicable only when data is returned to the Requester. See Table B9.4 for permitted errors when data is not returned to the Requester.

Table B9.4 shows the legal combincations of RespSepData and DataSepResp in relation to their message origins.

Table B9.4: Legal combinations of RespSepData and DataSepResp according to message origins

RespSepData	DataSepResp	Legal combination when DataSepResp	
		From Home	From Subordinate
OK	OK	Y	Y
	NDERR	Y	Y
	DERR	Y	Y
NDERR	NDERR	Y	-

B9.1.4.2 Dataless transactions

A DERR can be reported for a Dataless transaction when the processing of the transaction by another component encounters a data corruption error. This DERR can be indicated back to the originating component, even though a transfer of data does not occur.

Table B9.5 shows the Dataless transaction packets legal RespErr field values.

Table B9.5: Legal RespErr field values for Dataless transactions

Dataless transaction	Associated Response packets												CompAck
	Comp				Persist				CompPersist				
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR	
CleanUnique	Y	Y	Y	Y	-	-	-	-	-	-	-	-	OK
MakeReadUnique[a]	Y	N	Y	Y	-	-	-	-	-	-	-	-	OK
MakeUnique	Y	N	Y	Y	-	-	-	-	-	-	-	-	OK
CleanShared	Y	N	Y	Y	-	-	-	-	-	-	-	-	-
CleanSharedPersist	Y	N	Y	Y	-	-	-	-	-	-	-	-	-
CleanSharedPersistSep	Y	N	Y	Y	Y	N	Y	Y	Y	N	Y	Y	-
CleanInvalid CleanInvalidPoPA MakeInvalid	Y	N	Y	Y	-	-	-	-	-	-	-	-	-
Evict	Y	N	N	Y	-	-	-	-	-	-	-	-	-

[a] Applicable only when data is not returned to the Requester. See Table B9.2 and Table B9.3 for permitted errors when data is returned to the Requester.

Table B9.6 shows the permitted RespErr values in responses to StashOnce transactions.

Table B9.6: Legal RespErr field values for Dataless Stash transactions

Dataless Stash transaction	Associated Response packets											
	Comp				StashDone				CompStashDone			
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
StashOnceUnique StashOnceShared	Y	N	Y	Y	-	-	-	-	-	-	-	-
StashOnceSepUnique StashOnceSepShared	Y	N	Y	Y	Y	N	Y	Y	Y	N	Y	Y

B9.1.4.3 Write transactions

A Write transaction can include either a NDERR or a DERR. Errors can be signaled in both directions, from the Requester to the Completer, and from the Completer back to the Requester.

Write data which is known to be corrupt must have an appropriate Error indication where the error can be Poison or DERR.

For a Write transaction an error can be signaled from the Completer back to the Requester using either the combined CompDBIDResp or using the Comp response. It is permitted, but not required, for the Completer to signal an error even before the WriteData for the transaction is observed. This can occur when the processing of the transaction, such as the cache lookup, encounters a data corruption error.

Table B9.7 shows the Write transaction response packets legal RespErr field values.

Table B9.7: Legal RespErr field values for Write transactions

Write transaction	Associated Response packets									CompAck
	DBIDResp*	Comp				CompDBIDResp				
		OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR	
WriteNoSnp	OK	Y	Y	Y	Y	Y	Y	Y	Y	OK
WriteNoSnpDef	OK	Y[a]	N	Y	Y	Y[a]	N	Y	Y	-
WriteUnique	OK	Y	N	Y	Y	Y	N	Y	Y	OK
WriteNoSnpZero WriteUniqueZero	OK	Y	N	Y	Y	Y	N	Y	Y	-
WriteBack WriteClean WriteEvictFull	-	-	-	-	-	Y	N	Y	Y	-
WriteEvictOrEvict	-	Y	N	N	Y	Y	N	Y	Y	OK

[a] In WriteNoSnpDef transactions, used in combination with Resp[2:0] to communicate Successful, Unsupported, and Defer responses. See Table B4.28.

A Requester that detects an error in the write data to be sent can include an Error indication with the write data packet. This indicates that the data value is known to be corrupt.

Table B9.8 shows the Write transaction Data packets legal RespErr field values.

Table B9.8: Legal RespErr field values in Data packets for Write transactions

Write transaction	Associated Data packets											
	WriteData				WriteDataCancel				NonCopyBackWriteDataCompAck			
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
WriteNoSnp	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
WriteNoSnpDef	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
WriteUnique	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N

Continued on next page

Table B9.8 – *Continued from previous page*

Write transaction	Associated Data packets											
	WriteData				WriteDataCancel				NonCopyBackWriteDataCompAck			
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
WriteBack WriteClean WriteEvictFull WriteEvictOrEvict	Y	N	Y	N	-	-	-	-	-	-	-	-

Table B9.9 shows the Write transaction TagMatch packet legal RespErr field values when MTE is supported on the interface of the Requester.

Table B9.9: Legal RespErr field values in TagMatch packets for Write transactions

Write transaction	Associated Response packets			
	TagMatch			
	OK	EXOK	DERR	NDERR
WriteNoSnp WriteUnique	Y	N	Y	Y
WriteBack WriteClean WriteEvictFull WriteEvictOrEvict WriteNoSnpZero WriteUniqueZero WriteNoSnpDef	-	-	-	-

For permitted RespErr field values in a Combined Write transaction response:

- See Table B9.7 for the corresponding responses for the Write request.
- See Table B9.5 for the corresponding responses for the CMO request.

The permitted error responses in the CompCMO are the same as those in the Comp in Table B9.5.

B9.1.4.4 Atomic transactions

It is permitted, but not required, for a Completer to give a Comp response before receiving all the write data associated with a transaction and has performed the required operation. This behavior is not compatible with a component that wants to signal a Data Error associated with the write data, and such components must use a delayed form of Comp or CompData response.

In Atomic transactions, Read data or Write data which is known to be corrupt must have an appropriate Error indication. The Read Data Error is either Poison, DERR, or NDERR. The Write Data Error is either Poison or DERR.

A DERR or NDERR can be signaled at the following points within a transaction:

- With the CompDBIDResp response.
- For an AtomicStore transaction, with the Comp response.
- For an AtomicLoad, AtomicSwap, and AtomicCompare transaction, with the CompData response.

Note

If a read data at Home due to a non-store Atomic request results in a DERR or NDERR, such an error can be propagated onto the DBIDResp or CompDBIDResp response for that request.

For Atomic transactions that are not able to complete, a NDERR must be used. The transaction structure, including all write data transfers, read data transfers, and other responses must still take place.

There is no need to specify an error associated with the execution of an atomic operation, such as overflow. All atomic operations are fully specified for all input combinations.

A transaction includes both outbound and inbound data, but only has a single Error field. For Atomic transactions, it is permitted for the Error field to indicate an error on either write data or read data. There is no mechanism supported within the transaction to differentiate between the potential different causes of an error. A fault log, or a similar structure, could be able to provide such information, but this is not a requirement of the CHI protocol.

The permitted RespErr values in Atomic transactions are an amalgamation of those permitted in Read and Write transactions.

A DERR can vary between data packets.

Table B9.10 shows the Atomic transaction Response packets legal RespErr field values.

Table B9.10: Legal RespErr field values in Response packets for Atomic transactions

Atomic transaction	Associated Response packets								
	DBIDResp	Comp				CompDBIDResp			
		OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
AtomicStore	OK	Y	N	Y	Y	Y	N	Y	Y
AtomicLoad AtomicSwap AtomicCompare	OK	Y	N	Y	Y	-	-	-	-

Table B9.11 shows the Atomic transaction Data packets legal RespErr field values.

Table B9.11: Legal RespErr field values in Data packets for Atomic transactions

Atomic transaction	Associated Response packets							
	WriteData				CompData			
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
AtomicStore	Y	N	Y	N	-	-	-	-

Continued on next page

Table B9.11 – *Continued from previous page*

Atomic transaction	Associated Response packets							
	WriteData				CompData			
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
AtomicLoad AtomicSwap AtomicCompare	Y	N	Y	N	Y	N	Y	Y

Table B9.12 shows the Atomic transaction TagMatch packet legal RespErr field values when MTE is supported.

Table B9.12: Legal RespErr field values in TagMatch packets for Atomic transactions

Atomic transaction	Associated Response packets			
	TagMatch			
	OK	EXOK	DERR	NDERR
AtomicStore AtomicLoad AtomicSwap AtomicCompare	Y	N	Y	Y

B9.1.4.5 Other transactions

This section describes the error handling requirements for the DVMOp and PrefetchTgt transactions.

B9.1.4.5.1 DVMOp

A DVMOp transaction can include a NDERR in the Comp response. The interconnect can consolidate error responses from all the snoop responses for a DVMOp and include a single error response in the final Comp message to the Requester. The DBIDResp packet must only use the OK response. Even though the Sender of a WriteData response might not use DERR, the packet can be marked as DERR if it encounters errors during transmission. See B9.2.3 *Interoperability of Poison and DataCheck*.

Table B9.13 shows the DVM transaction Response packets legal RespErr field values.

Table B9.13: Legal RespErr field values in Response packets for DVM transactions

DVM transaction	Associated Response packets								
	DBIDResp	Comp				CompDBIDResp			
		OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
DVMOp	OK	Y	N	Y	Y	Y	N	Y	Y

Table B9.14 shows the DVM transaction Data packets legal RespErr field values.

Table B9.14: Legal RespErr field values in Data packets for DVM transactions

DVM transaction	Associated Data packets			
	NCBWrData			
	OK	EXOK	DERR	NDERR
DVMOp	Y	N	Y	N

B9.1.4.5.2 PrefetchTgt

A PrefetchTgt transaction request to a non-supporting address must be discarded.

Note

A component is permitted, but not required, to record and report such an error.

B9.1.4.6 Cache Stashing transactions

If the specified Stash target does not support receiving Stash snoops, the Home must disregard the Stash hint and complete the transaction without Stashing. Examples of such Stash targets are RN-I, RN-D, legacy RN-F, or a Non-request node. In these circumstances, Home must not signal an error to the Requester. Such a wrongly specified Stash target can be attributed to a software-based error.

If the Home does not support Stash requests, the transaction must be completed in a protocol-compliant manner without signaling an error.

B9.1.4.7 Snoop transactions

A Snoop transaction response that includes data can indicate a DERR. A Snoop transaction response that includes data can mix OK and DERR responses for different packets within the transaction.

Data, in response to a Snoop request that is known to be corrupt, must have an appropriate Error indication where the error can be Poison or DERR.

A Snoop transaction response that does not include data can indicate an NDERR.

A Snoopee that responds with an NDERR:

- Must not include data with the response.

- Must invalidate any locally cached copies.
- Must set the cache state in the response to Invalid.
- Must not forward a CompData response to the Requester in response to a Forwarding snoop. As a consequence, if the CompData message is already sent to the Requester, the Snoop response to the Home must not include a NDERR.

Table B9.15 shows the Snoop request Response packets legal RespErr field values.

Table B9.15: Legal RespErr field values in Response packets for Snoop transactions

Snoop transaction	Associated Data and Response packets							
	SnpResp				SnpRespData			
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
SnpOnce	Y	N	N	Y	Y	N	Y	N
SnpClean								
SnpNotSharedDirty								
SnpShared								
SnpUnique								
SnpPreferUnique								
SnpUniqueStash								
SnpCleanShared								
SnpCleanInvalid								
SnpStashUnique	Y	N	N	Y	-	-	-	-
SnpStashShared								
SnpMakeInvalid								
SnpMakeInvalidStash								
SnpQuery								
SnpDVMOp								

It is recommended, but not required, that a DERR on a Clean cache line is dropped and the error is not propagated to the memory, nor included in the response to the Requester.

A DERR on a Dirty cache line must be propagated to the memory, and in the response to the Requester.

A DERR in response to the DataPull request is not expected to be transferred to the Comp response to the Stash request.

For a Forwarding snoop transaction, when simultaneously forwarding data to the Requester and returning Data to Home, it is permitted, but not required, for only one response to include an indication of a DERR if the other response does not encounter the error.

Table B9.16 shows the Forwarding Snoop response packets legal RespErr field values.

Table B9.16: Legal RespErr field values in Response packets for Forwarding Snoop transactions

Snoop transaction	Associated Response packets							
	SnpResp				SnpRespFwded			
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
SnpOnceFwd SnpCleanFwd SnpNotSharedDirtyFwd SnpSharedFwd SnpUniqueFwd SnpPreferUniqueFwd	Y	N	N	Y	Y	N	Y	N

Table B9.17 shows the Forwarding Snoop Data response packets legal RespErr field values.

Table B9.17: Legal RespErr field values in Data packets for Forwarding Snoop transactions

Snoop transaction	Associated Data packets							
	SnpRespData SnpRespDataFwded				CompData			
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
SnpOnceFwd	Y	N	Y	N	Y	N	Y	N
SnpCleanFwd SnpNotSharedDirtyFwd SnpSharedFwd SnpUniqueFwd SnpPreferUniqueFwd								

B9.2 Sub-packet level

There are two types of error reporting at the sub-packet level:

- B9.2.1 *Poison*
- B9.2.2 *Data Check*

B9.2.1 Poison

The Poison bit is used to indicate that a set of data bytes have previously been corrupted. Passing the Poison bit alongside the data in the DAT packet permits any future user of the data to be notified that the data is corrupt.

When Poison is supported:

- The DAT packet includes one Poison bit per 64 bits of data.
- Data marked as poisoned:
  - Must not be utilized by any Requester.
  - Is permitted, but not required, to be stored in caches and memory if marked as poisoned.
- The Poison value, once set, must be propagated along with the data.
- When a Poison error is detected, it is permitted, but not required, to over poison the data.
- Poison on MTE tags is not supported.

Poison must be accurate if there are any valid bytes in the 64-bit chunk. This is the Poison granularity. When all 8 bytes in the 64-bit chunk are invalid, the Poison bit can take any value.

A Data_Poison property is used to indicate if a component supports Poison.

Note

Although Poison on tags is not supported, implementations could choose to do one of the following:

- Poison associated with the data results in the tag being poisoned. Depending on the granularity of the poison associated with the tag, the same techniques that could be used to clear poison associated with data may not clear the poison on the tags.
- Poison associated with the data does not result in the tag being poisoned. This means that a corrupted tag could subsequently be used in an MTE Match operation, which could incorrectly fail. The rate at which this occurs should be significantly lower than the rate at which data corruption occurs.
- A mixture of approaches can be used, depending on the caching or storage structures that are used.

Other implementations are possible.

B9.2.2 Data Check

The DataCheck field is used to detect Data Errors in the DAT packet.

When Data Check is supported:

- The DAT packet carries 8 DataCheck bits per 64 bits of data.
- The Data Check bit is a parity bit that generates Odd Byte parity.

Note

Interface parity optionally extends the error detection provided on the DAT channel by the DataCheck field.

The Data_Check and Check_Type properties are used to indicate if DataCheck is supported in the DAT packet.

B9.2.3 Interoperability of Poison and DataCheck

If the recipient of a Data packet does not support the Poison and DataCheck features, the interconnect must enumerate and convert, as necessary, the Poison and DataCheck error responses to a DERR in the DAT packet.

If support for the Poison and DataCheck features is not similar across an interface, the following rules apply:

- Poison must be mapped to DataCheck or DERR if Poison is not supported across the interface. At such an interface, Poison is expected but not required to be mapped to DataCheck instead of DERR, if DataCheck is supported. When converting from Poison to DataCheck, when an 8-byte chunk is marked as Poisoned, all 8 bits of DataCheck corresponding to that chunk must be manipulated to generate a parity error.
- DataCheck must be mapped to Poison or DERR if DataCheck is not supported across the interface. At such an interface, DataCheck is expected but not required to be mapped to Poison instead of DERR, if Poison is supported. When converting from DataCheck to Poison, if one or more DataCheck bits in a given 8-byte chunk generates a parity error, the Poison bit corresponding to that chunk must be set.

Note

The difference between the handling of Poison and DERR is that a Poison error in a received data packet is typically deferred by the receiver, but a DERR error is typically not deferred by the receiver.

It is sufficient for the Sender of a data packet that detects a Poison error to indicate this in the Poison bits. It is not a requirement that the Sender sets the RespErr field value to DERR.

It is sufficient for the Sender of a data packet that detects a DataCheck error to indicate this in the DataCheck field and is not required to set RespErr field value to DERR.

As Poison and DataCheck fields are independently set, one type of error does not require setting of the other.

In a Data packet that has the RespErr field value set to DERR or NDERR, the value of the Poison and DataCheck fields are inapplicable and can take any value.

B9.3 Use of interface parity

For safety-critical applications, it is necessary to detect and possibly correct, transient and functional errors on individual wires within an SoC.

An error in a system component can propagate and cause multiple errors within connected components. *Error Detection and Correction* (EDC) is required to operate end-to-end, covering all logic and wires from source to destination.

One way to implement end-to-end protection, is to employ customized EDC schemes in components and implement a simple error detection scheme between components. Between these components there is no logic and connections are relatively short. This section describes a parity scheme for detecting single-bit errors on the interface between components. Multi-bit errors can be detected if they occur in different parity signal groups.

Figure B9.1 shows locations where parity can be used.

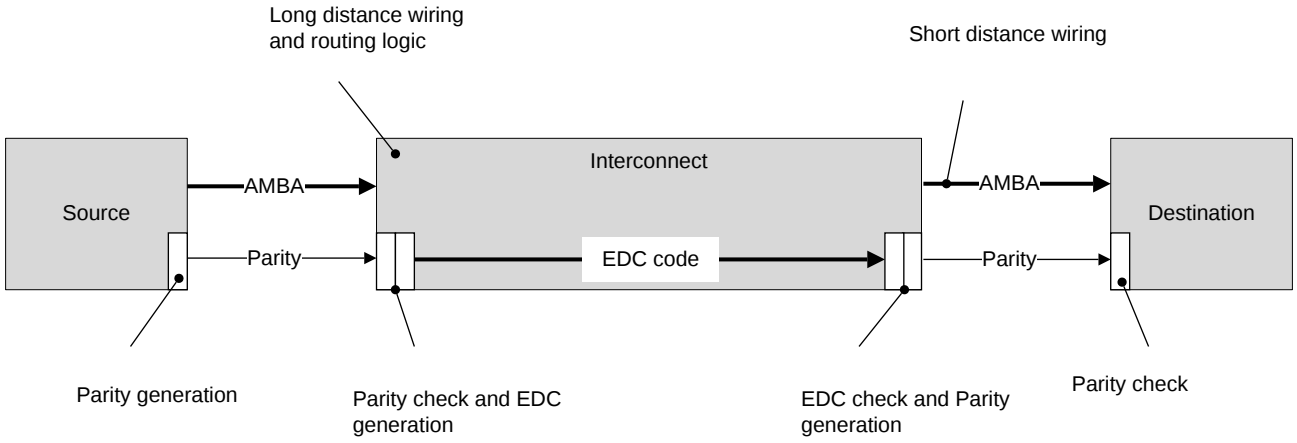

Figure B9.1: Parity use in AMBA

AMBA parity optionally extends the error detection provided on the DAT channel by the DataCheck field to cover the complete flit and control signals on all channels. The protection scheme employed on an interface is defined by the property Check_Type. See B16.1 *Interface properties and parameters*.

B9.3.1 Byte parity check signals

The following attributes are common to all the check signals added for byte parity interface protection:

- Odd parity is used. Odd parity means that check signals are added to groups of signals on the interface and driven such that there is always an odd number of asserted bits in that group.
- Parity signals covering data and payload are defined such that there are no more than 8 bits per group. This limitation assumes that there is a maximum of 3 logic levels available in the timing budget for generating each parity bit.
- Parity signals covering critical control signals, which are likely to have a smaller timing budget available, are defined with a single odd parity bit.
- The least significant check bit of the check signal covers the least significant byte of payload.
- If the bits in a payload do not fill the most significant byte, the most significant bit of the check signal covers fewer than 8 bits.

- Check signals must be driven correctly in every cycle that the Check Enable term is True, see Table B9.18.
- Parity signals must be driven appropriate to all the bits in the associated payload, irrespective of whether those bits are applicable.

B9.3.2 Error detection behavior

This specification is not prescriptive regarding component or system behavior when a parity error is detected. Depending on the system and affected signals, a flipped bit can have a wide range of effects. A flipped bit could:

- Be harmless
- Cause performance issues
- Cause data corruption
- Cause security violations
- Cause a deadlock

Also, an error on one signal could also cause parity failure on other signals.

When an error is detected, the receiver has options to:

- Terminate or propagate the transaction. It is permitted, but not required, to be protocol-compliant when the transaction is terminated.
- Correct the parity check signal or propagate the error.
- Update its memory or leave untouched. It is permitted, but not required, to mark the location as poisoned.
- Signal an error response through other means, for example, with an interrupt.

B9.3.3 Interface parity check signals

Table B9.18 shows the parity check signals and their properties on each of the channels.

Table B9.18: Interface parity check signals

Channel	Check signal	Signals covered	Check signal width	Check granularity	Check enable
Common	**SYSCOREQCHK**	**SYSCOREQ**	1	1	**RESETn** == 1
	SYSCOACKCHK	**SYSCOACK**	1	1	**RESETn** == 1
Common	**TXSACTIVECHK**	**TXSACTIVE**	1	1	**RESETn** == 1
	RXSACTIVECHK	**RXSACTIVE**	1	1	**RESETn** == 1
Common	**TXLINKACTIVEREQCHK**	**TXLINKACTIVEREQ**	1	1	**RESETn** == 1
	TXLINKACTIVEACKCHK	**TXLINKACTIVEACK**	1	1	**RESETn** == 1
	RXLINKACTIVEREQCHK	**RXLINKACTIVEREQ**	1	1	**RESETn** == 1
	RXLINKACTIVEACKCHK	**RXLINKACTIVEACK**	1	1	**RESETn** == 1
REQ	**REQFLITPENDCHK**	**REQFLITPEND**	1	1	**RESETn** == 1
	REQFLITVCHK	**REQFLITV**	1	1	**RESETn** == 1
	REQFLITCHK	**REQFLIT**	`ceil(R/8)`[a]	1 to 8	**REQFLITV** == 1
	REQLCRDVCHK	**REQLCRDV**	1	1	**RESETn** == 1
RSP	**RSPFLITPENDCHK**	**RSPFLITPEND**	1	1	**RESETn** == 1
	RSPFLITVCHK	**RSPFLITV**	1	1	**RESETn** == 1
	RSPFLITCHK	**RSPFLIT**	`ceil(T/8)`[b]	1 to 8	**RSPFLITV** == 1
	RSPLCRDVCHK	**RSPLCRDV**	1	1	**RESETn** == 1

Continued on next page

Table B9.18 – *Continued from previous page*

Channel	Check signal	Signals covered	Check signal width	Check granularity	Check enable
SNP	**SNPFLITPENDCHK**	**SNPFLITPEND**	1	1	**RESETn** == 1
	SNPFLITVCHK	**SNPFLITV**	1	1	**RESETn** == 1
	SNPFLITCHK	**SNPFLIT**	`ceil(S/8)`[c]	1 to 8	**SNPFLITV** == 1
	SNPLCRDVCHK	**SNPLCRDV**	1	1	**RESETn** == 1
DAT	**DATFLITPENDCHK**	**DATFLITPEND**	1	1	**RESETn** == 1
	DATFLITVCHK	**DATFLITV**	1	1	**RESETn** == 1
	DATFLITCHK	**DATFLIT**	`ceil(D/8)`[c]	1 to 8	**DATFLITV** == 1
	DATLCRDVCHK	**DATLCRDV**	1	1	**RESETn** == 1

[a] R = Request flit width. See B13.9.1 *Request flit*.

[b] T = Response flit width. See B13.9.2 *Response flit*.

[c] S = Snoop flit width. See B13.9.3 *Snoop flit*.

[d] D = Data flit width. See B13.9.4 *Data flit*.

B9.4 Hardware and software error categories

This specification defines two error categories:

- Software-based
- Hardware-based

B9.4.1 Software-based error

A software-based error occurs when multiple accesses to the same location are made with incorrect or mismatched Snoopable or Memory attributes. See B2.7.7 *Mismatched Memory attributes*.

A software-based error can cause a loss of coherency and the corruption of data values. It is required that the system does not deadlock for a software-based error, and that transactions always progress through a system in a timely manner.

A software-based error, for an access within one 4KB memory region, must not cause data corruption within a different 4KB memory region.

For locations held in Normal memory, the use of appropriate stores and software cache maintenance can be used to return memory locations to a defined state.

When accessing a peripheral device the correct operation of the peripheral cannot be guaranteed. The only requirement is that the peripheral continues to respond to transactions in a protocol-compliant manner. The sequence of events that could be required to return a peripheral device that has been accessed incorrectly to a known working state is IMPLEMENTATION DEFINED.

B9.4.2 Hardware-based error

A hardware-based error is defined as any protocol error that is not a software-based error.

Caution

If a hardware-based error occurs, recovery from the error is not guaranteed. The system could crash, lock-up, or suffer some other non-recoverable failure.

Chapter B10
Realm Management Extension

This chapter describes the *Realm Management Extension* (RME) used in the CHI protocol. It contains the following sections:

B10.1 Introduction

RME is one component of the Arm *Confidential Computer Architecture* (CCA). Together with the other components of the Arm CCA, RME enables support for dynamic, attestable, and trusted execution environments, also called Realms, to be run on an Arm PE.

RME provides hardware-based isolation that allows execution contexts to run in different Security states and share resources in the system.

B10.2 Physical Address Space, PAS

An RME enabled system defines four Physical Address Spaces. The four Physical Address Spaces are:

- Secure
- Non-secure
- Root
- Realm

Note

Prior to CHI-F, only the Secure and Non-secure Physical Address Spaces were defined.

For further information, see B2.7.2 *Physical Address Space, PAS*.

B10.3 Cache maintenance

This section describes additional Cache Maintenance Operations required in an RME-enabled system.

B10.3.1 Cache Maintenance Operations

The additional Cache Maintenance Operations defined are:

- CleanInvalidPoPA
- WriteBackFullCleanInvPoPA
- WriteNoSnpFullCleanInvPoPA
- WriteNoSnpPtlCleanInvPoPA

The additional Cache Maintenance Operations provide support for the dynamic transition of memory granules between Physical Address Spaces.

To support the new Cache Maintenance Operations, a point in the system is defined, the Point of Physical Aliasing (PoPA). The PoPA is a point at which updates to a location in one PAS are visible to all other Physical Address Spaces. Cache Maintenance Operations could be required to more than one PAS to ensure that any data written earlier is fully visible to the intended Physical Address Spaces.

The RME_Support property must be set to True to support the additional Cache Maintenance Operations at an interface. See B16.1.16 *RME_Support*.

For further information, see PoPA and B4.2.2.1 *Cache Maintenance transactions*.

B10.3.2 Remote invalidation

RME requires the ability to invalidate memory mapped as Non-shareable Cacheable remotely. The ability to remotely invalidate memory removes the need to rely on cache maintenance on each PE that may have memory mapped as Non-shareable Cacheable. To support this, the Nonshareable_Cache_Maint property must be set to True.

For further information, see B2.7.7 *Mismatched Memory attributes* and B16.1.17 *Nonshareable_Cache_Maint*.

B10.4 DVM

Additional DVM operations and fields are defined to support:

- The increased number of Physical Address Spaces in an RME-enabled system.
- The invalidation of Realm Protection Unit entries.

The DVM_Support property must be set to DVM_v9.2 to ensure the DVM operations required for RME support are present. See B16.1.19 *DVM_Support*.

For further information, see Chapter B8 *DVM Operations*.

B10.5 MPAM

MPAM defines independent PartID spaces for each PAS, encoded in the 2-bit MPAM Space (MPAMSP) attribute.

Note

Prior to CHI-F, a single bit MPAMNS attribute was present in place of the MPAMSP attribute.

For further information, see B11.4 *MPAM*.

Chapter B11
System Control, Debug, Trace, and Monitoring

This chapter describes mechanisms that provide extra support for the controlling, debugging and tracing of systems, and the monitoring of systems to enhance performance. It contains the following sections:

B11.1 Quality of Service (QoS) mechanism

B11.1.1 Overview

A system could utilize a QoS scheme to achieve:

- A guaranteed maximum latency for transactions in a particular stream.
- Minimum bandwidth guarantees for a stream of requests.
- Best effort value of bandwidth and latency provided to requests of a particular stream.

The low latency, or guaranteed throughput requirements, required to meet system QoS demands are primarily the responsibility of the transaction end points with support from the intermediate interconnect. The protocol supports this by defining a QoS priority value for packets and controlling request flow using a defined credit mechanism.

B11.1.2 QoS priority value

A 4-bit value is used to prioritize the processing of the packets at protocol nodes and within the interconnect. The QoS Priority Value (PV) for packets is assigned by the source of the transaction. In typical usage models, this value depends on the source type and the class of traffic, with ascending values of QoS indicating a higher priority level. The source could also dynamically vary this value, depending on some accumulated latency and required throughput metric.

B11.1.3 Repeating a transaction with a higher QoS value

When a transaction has been sent with a particular QoS value, the same transaction can be sent again with a different QoS value, typically higher. The Completer is required to handle this situation as multiple different requests.

In this instance, if one of the transactions receives a RetryAck response, the transaction can be canceled and the credit returned. See B2.9.1 *Credit Return*.

B11.2 Data Source indication

It is permitted, but not required, for the Completer of a Read request to specify the source of the data. The source is specified in the DataSource field of the CompData, DataSepResp, SnpRespData, and SnpRespDataPtl responses. DataSource field value is valid even in data responses with error.

The DataSource field can also be used to transport information to bias SLC replacement policy. See B11.3 *SLC Replacement Hint*.

B11.2.1 DataSource value assignment

The DataSource value assignment uses the following key:

`R = 0` The memory is local.

`R = 1` The memory is remote.

The DataSource values must be assigned as follows:

- Fixed values are used for DataSource when data comes from memory and are used to indicate the following:

  `0bR0110` PrefetchTgt memory prefetch was useful. Read data was obtained from the Subordinate with lower latency as the PrefetchTgt request already read or initiated a read of data from memory.

  `0bR0111` PrefetchTgt memory prefetch was not useful. Read request went through a complete memory access and therefore did not have any latency reduction due to the PrefetchTgt request sent earlier. The precise reason for signaling that a prefetch was not useful is IMPLEMENTATION DEFINED.

  Note

  There are several reasons why the PrefetchTgt request could not be useful. Examples are that the prefetch was dropped by the Subordinate, the data obtained by the prefetch was replaced in the buffer, or the Read request arrived at the Subordinate before the prefetch.

- When a system has only a local chip and no concept of remote, the DataSource[4] bit can be reused for additional local chip granularity for DataSource.
- For a response from a cache, not memory, the DataSource value is IMPLEMENTATION DEFINED. It is recommended, but not required, to use certain values for DataSource in these cases. A component is permitted, but not required, to have software programmability to override the DataSource value to:
  - Change the groupings to more suitable specific configuration settings.
  - Change the values where the values are not correct.
- A Completer is permitted to not support sending a useful DataSource value, in which case:
  - The Completer, except for a memory SN-F, must return a `0bR0000` value.
  - A memory SN-F component must return `0bR0111` as a default value. Such exceptions must be understood by the system.

B11.2.2 Crossing a chip-to-chip interface

The chip interface module, if one exists, is responsible to map DataSource values in the incoming data packets to different values to identify that the response came from the remote chip.

B11.2.2.1 Suggested DataSource values

Figure B11.1 shows an example multichip configuration and the suggested mapping of DataSource values to different components in the system:

- Each chip in the system has two processors per cluster, with a three-level cache hierarchy.
- The cache in the chip-to-chip interface module is identified as part of the interconnect caches.
- A non-memory component that is not programmed to identify itself as the source of data can return the default value of `0bR0000`.

Table B11.1 lists the suggested DataSource encodings.

Table B11.1: Suggested DataSource value encodings

DataSource[3:0]	DataSource[4]	
	0, In the local chip	**1, In the remote chip**
0b0000	Not supported or Completer is not sendng 'useful' value	
0b0001	Peer processor cache within local cluster	-
0b0010	Local cluster cache	-
0b0011	Interconnect cache	
0b0100	Peer cluster caches	
0b0101	-	
0b0110	PrefetchTgt, memory prefetch was useful	
0b0111	PrefetchTgt, memory prefetch was not useful	
0b1000-0b1001	-	
0b1010	Local cluster cache, unused prefetch[a]	-
0b1011	Interconnect cache, unused prefetch[a]	
0b1100-0b1111	-	

[a] See B11.3 *SLC Replacement Hint*.

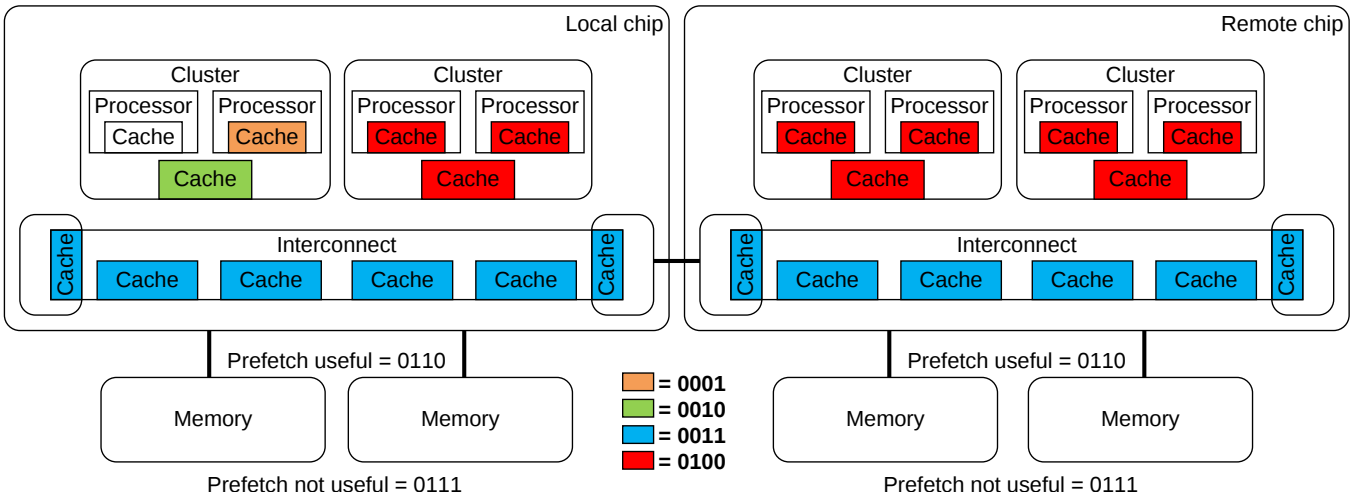

Figure B11.1: Suggested DataSource[3:0] values

B11.2.3 Example use cases

Two examples of how DataSource information can be used by a Requester are:

- To determine the usefulness of a PrefetchTgt transaction in initiating a memory controller prefetch.
  - By monitoring the DataSource value in the data returned from the memory SN-F, the Requester can determine the usefulness of sending PrefetchTgt requests and can modulate the rate, and the sending, of PrefetchTgt requests.
- Can be used by performance profiling and debug software to evaluate and optimize the data sharing pattern.

B11.3 SLC Replacement Hint

The purpose of this feature is to forward cache replacement hints from the Requesters to the caches in the interconnect, the *System Level Caches* (SLC). Typically, a Request Node has the best knowledge of the utility of a cache line. An SLC that is informed of this knowledge can use it to bias its own replacement algorithms and manage cache line replacement in a more efficient manner.

B11.3.1 Characteristics

Although the replacement information is most useful in CopyBack requests, SLC is not restricted to CopyBack transactions. SLCRepHint is extended to all requests from the Request Node to HN-F, except for the following:

- Atomics
- Stash transactions when StashNIDValid is 1
- PrefetchTgt
- PCrdReturn
- DVMOp

This feature is supported by a 7-bit field called SLCRepHint that is included in the REQ channel. SLCRepHint includes two subfields, a 3-bit Replacement field and a 1-bit UnusedPrefetch field.

Figure B11.2 shows the placement of the SLCRepHint subfields.

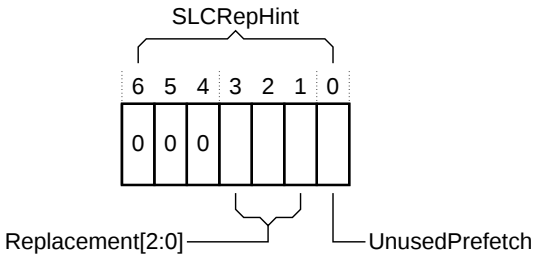

Figure B11.2: SLCRepHint subfield placement

The SLCRepHint field shares the same REQ packet location as ReturnNID and StashNID. When the node ID widths are greater than 7 bits and the field is used as SLCRepHint, the unused bits of the shared field must be set to 0.

Table B11.2 shows the encoding of the UnusedPrefetch subfield.

Table B11.2: UnusedPrefetch subfield encoding

UnusedPrefetch	Description
0	The cache line could have been used since being fetched.
1	The cache line was not used since being fetched.

Note

A Request Node that does not track the usage of the cache line can set the UnusedPrefetch bit value to 0.

The SLCRepHint field is only applicable in requests from a Request Node to HN-F.

The field is inapplicable in requests from the Request Node to HN-I and the Home Node to the Subordinate Node. The field can take any value.

The field is inapplicable in requests from the Request Node to the Subordinate Node, and must be set to 0.

Table B11.3 shows the suggested Replacement subfield encodings and their meaning.

Table B11.3: Suggested Replacement subfield encodings

Replacement[2:0]	Description
0b000	No recommendation (default)
0b100	Most likely to be used again
0b101	More likely to be used again
0b110	Somewhat likely to be used again
0b111	Least likely to be used again
0b011-0b001	Unused

B11.4 MPAM

Memory System Resource Partioning and Monitoring (MPAM) is a mechanism to efficiently utilize the memory resources among users and to monitor the utilization of those resources. The resources are partitioned among users by *Partition Identifier* (PartID) and *Performance Monitoring Group* (PerfMonGroup). A Requester that supports MPAM includes a label within each sent request, identifying the partition to which the request belongs, together with the performance monitoring group within that partition. The Home or the Subordinate uses this information to allocate their resources to this request. Support for MPAM on an interface is defined by the MPAM_Support property. See B16.1.9 *MPAM_Support*.

The MPAM field is applicable only in the REQ and SNP channels:

- On the REQ channel, when a sender does not want to use MPAM for a request, the MPAM values must be set to default settings. See Table B11.5.
- On the SNP channel, MPAM values are only applicable in Stash snoops. In Non-stash type snoops, MPAM values are inapplicable and must be set to default values.

Field width is either 0 bits or 12 bits:

- The width is 0 bits on interfaces that do not support MPAM.
- When the MPAM_Support property is MPAM_9_1, the MPAM field width is 12 bits. The field is further divided into the subfields:
  - PartID = 9 bits
  - PerfMonGroup = 1 bit
  - MPAMSP = 2 bits

Figure B11.3 shows the allocation of the MPAM field bits.

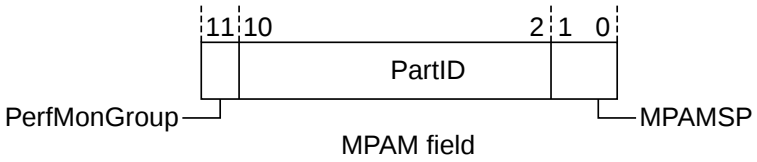

Figure B11.3: MPAM subfields bit allocation

It is IMPLEMENTATION DEFINED how the Receiver uses the MPAM field values.

B11.4.1 MPAMSP

MPAMSP provides the partition identifier namespace selector.

Table B11.4 shows the MPAMSP encodings.

Table B11.4: MPAMSP encodings

MPAMSP	Partition description
00	Secure
01	Non-secure
10	Root
11	Realm

All combinations of values of MPAMSP and PAS are permitted.

Note

MPAMSP replaces MPAMNS from CHI Issue F onwards.

B11.4.2 MPAM value propagation

The Receiver is permitted, but not required, to support the full range of partitions and performance monitor groups received in the request. During the system discovery and configuration process, it is expected that the capabilities of the system are discovered and the range of partitions and the performance monitor groups used match the capabilities of the system.

MPAM field values must be propagated onto an interface that supports MPAM.

It is permitted, but not required, to propagate MPAM field values onto an interface that does not support MPAM.

Table B11.5 shows the default values for MPAM fields when not supported or not propagated.

Table B11.5: Default values for MPAM subfields

MPAM field	Value
PerfMonGroup	0
PartID	0
MPAMSP	Same as PAS value in the request message Same as PAS value in the snoop message

B11.4.3 Stash transaction rules

MPAM values in Stash snoops must be the same as in the request that generated the snoops.

For responses to Stash snoops, when the response includes a DataPull request, the Home must assume the MPAM values in the DataPull request are the same as in the original Stash request.

B11.4.4 Request to Subordinate rules

MPAM values in a request to the Subordinate, where the request is generated by a request to the Home, must be the same as the MPAM values in that request to the Home.

B11.5 Page-based Hardware Attributes

Page-based Hardware Attributes (PBHA) is an optional, IMPLEMENTATION DEFINED feature. Support for PBHA on an interface is defined by the PBHA_Support property. See B16.1.18 *PBHA_Support*. PBHA allows software to set up to 4 bits in translation tables, which is then propagated through the memory system with transactions. See B13.10.31 *Page-based Hardware Attribute, PBHA*.

PBHA values are obtained from page tables during address translations. It is expected, but not required, that all translations to a given PA provide the same PBHA value.

B11.5.1 PBHA field applicability

On the REQ channel, PBHA is applicable in all requests, except for DVMOp and PCrdReturn where it is inapplicable and must be 0.

On the DAT channel, PBHA is only applicable in SnpRespData, SnpRespDataPtl, and SnpRespDataFwded. The PBHA field is inapplicable and must be set to 0 in all other DAT messages.

On the SNP channel, PBHA is only applicable in Stash snoops. The PBHA field is inapplicable and must be set to 0 in all Non-stash snoops.

B11.5.2 Interconnect use of PBHA

If the interconnect forwards a request onto the Subordinate, it is expected to forward the PBHA value along with the request.

If the interconnect caches a line, it is expected to cache the PBHA value along with the line. If the line is evicted, it is expected that the PBHA value is forwarded onto the Subordinate.

B11.5.3 Stash transaction rules

When Home receives a Stash request and generates a Stash snoop to a Requester, it is expected, but not required, that the PBHA value from the request is sent along with the snoop.

The Requester of a DataPull request obtains the address of the request from the received snoop, rather than through an address translation process. Therefore, the Requester is expected, but not required, to obtain the PBHA value from the stash snoop.

B11.5.4 PBHA value consistency

PBHA values can be added to a request during address translation and propagated through a system if PBHA is supported by downstream components. The PBHA value corresponding to a particular address flowing through the system can be precise or imprecise.

To make a PBHA value precise, the PBHA values used in the request and corresponding responses must be the same as the PBHA values obtained from the address translation. Other considerations include:

- The precise PBHA value is available to the request if caches store PBHA values along with the data when the line is written back to memory. Memory accesses that are the result of cache evictions, dedicated Cache Maintenance Operations, or snoop filter back invalidations must use the PBHA that the entry was cached with, not the PBHA value in any associated request.
  - A CMO that operates on a cache line is also expected to operate on any PBHA value stored along with the line.

- Back invalidation from the snoop filter resulting in snoop response with data must obtain PBHA value along with data from the Snoopee. Alternatively, the snoop filter can hold PBHA values and add them to the data received in the snoop response.

Note

A non-exhaustive list of how PBHA values can become imprecise include:

- Address translation is not used by the requests.
- PBHA value is not held in the caches.
- PBHA value is not provided alongside stashing transactions.
- Multiple Virtual Addresses mapped to the same PA where translations provide different PBHA values.
- Appropriate Cache Maintenance Operations have not been performed after the modification of the PBHA value.

B11.6 Completer Busy

The Completer Busy indication is a mechanism for the Completer of a transaction to indicate its current level of activity. Completer Busy provides additional information to a Requester on how aggressive speculative activity can be generated to improve performance.

CBusy Completer Busy. A 3-bit field that is applicable in the appropriate DAT and RSP packets. When separate Data and Comp responses are used for a single Read request, the Busy indication in each response can be independently set.

The CBusy field is not applicable in:

- NonCopyBackWriteData
- NonCopyBackWriteDataCompAck
- CopyBackWriteData
- WriteDataCancel
- CompAck

Note

DataSource can be used as a qualifier to the Completer Busy indication such that some data sources, for example a Forwarding snoop, do not influence the busy indication.

B11.6.1 Use case

It is IMPLEMENTATION DEFINED how a CBusy indication is set by the Completer and how CBusy should be interpreted by a Requester. However, it is recommended that implementations consider mechanisms to avoid a few busy Completers among many skewing the results.

B11.6.1.1 Example use case

The following is an example encoding of the CBusy field.

CBusy[2] When asserted, this indicates multiple cores are actively making requests.

CBusy[1:0] Indicates the degree of fullness of the tracker at the Completer as:

- `00` = Less than 50% full
- `01` = Greater than 50% full
- `10` = Greater than 75% full
- `11` = Greater than 90% full

The prefetcher at the Requester can use the CBusy field values and fine-tune the prefetcher.

Table B11.6 shows the prefetcher mode that is determined by the value of CBusy.

Table B11.6: Prefetcher mode that is determined by CBusy

CBusy[2]	CBusy[1:0]	Prefetcher mode
1	11	Disable inaccurate prefetchers
Can take any value	10	Very conservative mode on inaccurate prefetchers
1	01	Moderately aggressive mode on inaccurate prefetchers
Can take any value	00	Fully aggressive mode on inaccurate prefetchers

B11.7 Trace Tag

A TraceTag bit per channel provides enhanced support for debugging, tracing, and performance measurement of systems.

B11.7.1 TraceTag usage and rules

The rules for when to set and how to propagate TraceTag bit values are:

- The TraceTag bit can be set by the transaction initiator or an interconnect component.
- A component that receives a packet with the TraceTag bit set in every received packet must preserve and reflect the value back in any response packet or spawned packet generated in response to the received packet. Otherwise, the TraceTag bit in the response and spawned packet can take any value. This requirement is true only when the TraceTag bit in the response packet is applicable. Examples of pairs of such response and received packets are CompAck in response to multiple CompData packets and Snoop response to the pair of two DVMOp packets.
- If a received packet spawns multiple responses, such as a Write request resulting in separate Comp and DBIDResp responses, or a Read request generating separate DataSepResp and RespSepData responses, all such spawned responses are required to have the TraceTag bit set if the spawning packet has the TraceTag bit set. If the spawning packet does not have the TraceTag bit set, the value of the TraceTag bit in a spawned packet is independent of the value of the bit in other related spawned packets.
- If a component can receive multiple packets that are associated with a single transaction, then for each packet that it, in turn, generates, the TraceTag value is only required to be set if it is set in the associated received packet. For example:
  - A Write transaction flow at the Request Node could have write data and CompAck as two responses for received packets DBIDResp and Comp respectively. As CompAck is in response to the received Comp only, its TraceTag bit value is only required to depend on the TraceTag bit value in the Comp packet and similarly for the write data and DBIDResp Response-Received Packet pair. A request that receives separate DataSepResp and RespSepData responses and generates a CompAck, is only required to have the TraceTag bit set in CompAck if RespSepData has the TraceTag bit set.
  - The TraceTag bit in the NonCopyBackWriteDataCompAck response from the Request Node must be set if either one of Comp or DBIDResp that caused the WriteData response have the TraceTag bit set.
- When an interconnect receives a packet with the TraceTag bit set, the value must be preserved and not reset the value.

Note

Propagating the value of the TraceTag bit on a resulting cache eviction is IMPLEMENTATION DEFINED.

The precise mechanism to trigger and utilize the TraceTag bit is IMPLEMENTATION DEFINED.

It is expected that the TraceTag bit is limited to single system-wide use at any time.

Some of the ways the trace tag mechanism can be used are:

- Debug, by tracing transaction flows through the system.
- Performance counting
- Latency measurement

Examples, not an exhaustive list, of Request-Response pairs are:

- A Snoop response, with or without data, in response to a Snoop request.

- A Snoop response in response to a SnpDVMOp request.
- Data response from the Subordinate Node in response to a Read request.
- Spawned requests from HN-F:
  - Snoops generated in response to a request from the Request Node.
  - Request to SN-F generated in response to a request from the Request Node.
- Spawned request from HN-I:
  - Read or Write request to SN-I generated in response to a request from the Request Node.
- A CompAck from the Request Node in response to CompData, Comp, or RespSepData.
- A RetryAck response from the Home Node or the Subordinate Node to any request.
- A ReadReceipt response from the Home Node or the Subordinate Node to a Read request or from the Subordinate Node to a ReadNoSnpSep request.
- A DBIDResp response to a Write request.

Chapter B12
Memory Tagging

This chapter describes the Memory Tagging mechanism and contains the following sections:

B12.1 Introduction

The *Memory Tagging Extension* (MTE) is a mechanism that is used to check the correct usage of data held in memory. When a memory location is allocated for a particular use, a memory tag can also be assigned. This memory tag is held alongside that data in memory and is referred to as the Allocation Tag. When the memory location is later accessed, the Requester uses both the address of the location and the tag value that the Requester believes is associated with the location. This tag is referred to as the Physical Address Tag or Physical Tag.

For any access where tag checking is enabled, the Physical Tag is checked against the Allocation Tag. The access always progresses as normal, and the result of the tag check determines whether or not an error condition is signaled.

MTE ensures that a memory access is for its expected purpose, rather than an erroneous or malicious access. MTE can be used at runtime to identify many common programming memory errors, such as buffer-overflow and use-after-free.

The memory tag consists of a 4-bit tag associated with each aligned 16 bytes of data in memory.

The following behavior is supported:

- Memory tagging is permitted only in requests to Normal WriteBack memory.
- Read transactions have an indication in the transaction request that determines whether the Allocation Tag value must be returned alongside the data.
  Checking of the returned Physical Tag against the Allocation Tag is performed by the Requester.
  In the case where a cache holds the data value, but does not hold the Allocation Tag value, a Read transaction that returns both data and tag must be performed. The data returned is not required to be valid.
- Read requests that require tags to be fetched must not use Forwarding snoops.
- StashOnce transactions that request Allocation Tags. The Allocation Tags are expected to be stashed along with the stashing of data.
- Write transactions that have a Physical Tag supplied alongside the write data that must be checked against the Allocation Tag.
  Checking of the Physical Tag against the Allocation Tag is performed by the Completer. In the case of a mismatch a notification of the failure is required.
- Write transactions that update the Allocation Tag to a new value.
  These Write transactions typically update the data at the same time. However, it is permitted to have no BE bits asserted so that only the tag is updated.
- Write transactions which pass a Dirty or Clean cache line to a downstream cache or memory controller without either updating or checking the tags. These Write transactions always include data and provide an indication whether the Allocation Tag value is also being passed with the data.
- Snoop transactions that return data can also return the associated Allocation Tag. If the tags are Dirty, they must be returned. If the tags are Clean, returning them is optional.
- Cache Maintenance Operations must operate on both the data and the corresponding memory tags.

B12.2 Message extensions

The following extensions to the CHI message definitions are used to support Memory Tagging:

Tag Provides sets of 4-bit tags, each associated with an aligned 16 bytes of data.

- Applicable on the DAT channel only.
- Size is `Data_Width/32 bits`.

See Tag.

TU Tag Update. Indicates which of the Allocation Tags must be updated.

- Applicable on the DAT channel only.
- Size is `Data_Width/128 bits`.

See TU.

TagOp Tag Operation. Indicates the operation to be performed on the tags present in the corresponding DAT channel.

- Applicable on the REQ, DAT, and RSP channels.
- Size is 2 bits.
- Table B13.31 shows the value encodings.

Table B12.1: TagOp encodings and Tag operations

TagOp[1:0]	Tag operation
0b00	*Invalid*
0b01	*Transfer*
0b10	*Update*
0b11	*Match* or *Fetch*

See TagOp.

Note

For clarity, in the following sections TagOp values are italicized to differentiate them from data and cache line values.

B12.3 Tag coherency

This section summarizes the tag coherency features.

Allocation Tags that are cached are kept hardware coherent. The coherence mechanism is the same as data coherence.

Applicable tag cached states are: Invalid, Clean, and Dirty. A cache line that is either Clean or Dirty is Valid.

Constraints on the combination of data cache state and tag cache state are:

- Tags can be Valid only when data is Valid.
- Tags can be Invalid when data is Valid.
- Both data and tags are Unique when a cache line is in a Unique state.
- Both data and tags are Shared when a cache line is in a Shared state.
- When a cache line with Dirty tags is evicted:
  - Both data and tags must be treated as Dirty.
  - The tags must be either written back to memory or passed Dirty [_PD] by Home to another cache.
- When Clean tags are evicted from a cache, they can be sent to other caches or dropped silently.
- When Clean tags are evicted with Dirty data, Clean tags can be transferred downstream of PoC along with Dirty data.

B12.4 Read transaction rules

A read can optionally fetch tags along with the data. The need to return tags along with read data is determined from the value of TagOp in the request.

B12.4.1 TagOp values

When TagOp value in the request is *Transfer*:

- Tags are required to be returned along with the data.
- The state of the returned tags must be the appropriate permitted cache state for the request being used.
- The number of tags to be returned is determined by the size of Data that is returned. For all Snoopable requests four tags per access must be returned.
- When required by the access, matching of Physical Tags against the Allocation Tags that are received along with read data is done at the Requester.

When TagOp value in the Request is *Fetch*:

- Tags are required to be returned along with data.
- Returned data is not required to be valid. The Requester must ignore the received data irrespective of the tag match result.
- All tags corresponding to a cache line must be returned.
- The state of the returned tags must be Clean or Dirty.
- If Dirty tags are returned, they must be preserved, unless updated, and written back to memory.

When TagOp value in the request is *Invalid*:

- It is permitted, but not required, that tags are returned along with the data.
- If tags are returned along with the data, they must be Clean.

B12.4.1.1 Converting tags from Shared to Unique

To update either the data or the tags or both, when both the data and the tags are present at the Requester in Shared state, and the Requester requires to move the cache line to a Unique state, the MakeReadUnique transaction with a TagOp value of *Transfer* is expected to be used. The Requester is permitted to use ReadUnique with a TagOp value of *Transfer*.

B12.4.1.2 Fetching tags when Data is present

If a Requester has a cached copy of a cache line, with data Valid but Allocation tags are not Valid, and the Requester requires a Tag Match to be performed, the Requester must use a Read request to fetch the required tags.

In the above scenario:

- If a Requester can guarantee to write a full cache line irrespective of the Tag Match result, it it permitted to use either:
  - ReadNoSnp with *Fetch* if the target memory location is Non-snoopable. The returned data is not required to be valid and must be dropped. Clean tags must be returned. All tags within the cache line must be Valid.
  - ReadUnique with *Fetch*, if the target memory location is Snoopable. The returned data is not required to be valid and must be dropped. Clean or Dirty tags must be returned. All tags within the cache line must be Valid.

- If a Requester cannot guarantee to write a full cache line irrespective of the Tag Match result, it is permitted to use either:
  - ReadClean with *Transfer*
  - ReadUnique with *Transfer*

See Table B4.37 for Requester cache state transitions.

When responding to a ReadClean, a Home that uses a Snoop Filter to track the cached state at the Requester, must not downgrade the state of the cache line in the Snoop Filter based on the state in the response to the Requester. That is, the Snoop filter must not change a line previously tracked as:

- Valid, to being tracked as Invalid.
- Unique, to being tracked as Shared.
- Dirty, to being tracked as Clean.

Note

Prior to Issue F, I and UCE were the only permitted initial cache line states for the ReadClean transaction. The Home that uses a Snoop Filter to track the cached states was permitted to set the state of the cached line based on the state in the response.

B12.4.1.3 Permitted responses and tag state

The data and state of the cache line received with the Allocation Tags must be appropriately handled to not break coherency.

When the request TagOp value is *Transfer*, the permitted response field values are:

- *Transfer*. Indicates the returned tags are Clean.
- *Update*. Indicates the returned tags are Dirty. Data response must pass Dirty [_PD].

When the request TagOp value is *Fetch*, the permitted response field values are:

- *Transfer*. Indicates the returned tags are Clean.
- *Update*. Indicates the returned tags are Dirty. Data response must pass Dirty [_PD].

When the request TagOp value is *Invalid*, the permitted response field values are:

- *Invalid*. Indicates the returned tags are Invalid.
- *Transfer*. Indicates the returned tags are Clean.

When the TagOp value in Read data is invalid, TU must be all zeros. Tag is inapplicable and can take any value.

When data and response are separately sent in a Read transaction, the TagOp field is only applicable in the Data-only message. TagOp is inapplicable in the Non-data response message and must be set to 0.

The cache state that the tags must be held in is consistent with the type of the Read request:

- For all Read requests with a TagOp value of *Invalid*, Invalid or Clean tags must be returned.
- For ReadNoSnp with a TagOp value of *Transfer* or *Fetch*, Clean tags must be returned.
- For ReadClean, ReadOnce, ReadOnceCleanInvalid, and ReadOnceMakeInvalid with a TagOp value of *Transfer*, Clean tags must be returned.
- For ReadNotSharedDirty with a TagOp value of *Transfer*, Clean or Dirty tags must be returned. Dirty tags are permitted to be returned only if the cache line state is Unique.
- For ReadShared with a TagOp value of *Transfer*, Clean or Dirty tags must be returned.

- For ReadUnique with a TagOp value of *Transfer* or *Fetch*, Clean or Dirty tags must be returned. The returned cache line state must be Unique.
- For MakeReadUnique with a TagOp value of *Invalid*, Invalid or Clean tags must be returned. Clean tags are only permitted in responses with data.
- For MakeReadUnique with a TagOp value of *Transfer*, if data is included in the response, Clean or Dirty tags must be returned.
  - Clean tags are permitted in responses with data.
  - Dirty tags are only permitted when the response is transferring the responsibility of updating Dirty data, that is, the response includes [UD_PD].
  - Dataless responses using Comp_UC or Comp_SC are permitted to signal Clean tags are being transferred, despite no Tag transfer occuring.
- In the case where Dirty tags are returned, the cache line returned must include pass Dirty [_PD].

When MTE is not supported for the targeted address, the response must use TagOp of *Invalid*.

For an Exclusive access sequence, the fetching of tags must avoid any form of request that Invalidates other copies of the cache line before the Exclusive Store transaction is performed. Typically, this is achieved by fetching tags at the same time the Exclusive Load transaction is performed.

B12.4.2 Permitted initial MTE tag states

Table B12.2 shows the permitted initial data state along with tag state for different Read transactions and permitted TagOp value in the corresponding request. The combination of tag and data states must obey the coherency rules described in B12.3 *Tag coherency*.

Table B12.2: Permitted initial Tag states and request TagOp values in Read transactions

Request	TagOp	Data state	Tag state
ReadNoSnp	*Invalid, Transfer, Fetch*	I	Invalid
ReadOnce ReadOnceMakeInvalid ReadOnceCleanInvalid	*Invalid, Transfer*	I	Invalid
ReadNotSharedDirty ReadShared	*Invalid, Transfer*	I, UCE	Invalid
ReadClean	*Invalid*	I, UCE	Invalid
	Transfer	I,UCE,UDP	Invalid
		SC, UC	Invalid, Clean
		SD, UD	Invalid, Clean, Dirty
ReadPreferUnique	*Invalid, Transfer*	I, UCE	Invalid
		SC	Invalid, Clean
		SD	Invalid, Clean, Dirty
ReadUnique	*Invalid, Transfer, Fetch*	I, UCE	Invalid
		SC, UC	Invalid, Clean

Continued on next page

Table B12.2 – *Continued from previous page*

Request	TagOp	Data state	Tag state
		SD, UD	Invalid, Clean, Dirty
MakeReadUnique	*Invalid*	SC	Invalid, Clean
		SD	Invalid, Clean, Dirty
	Transfer[a]	SC	Clean
		SD	Clean, Dirty

[a] To issue a MakeReadUnique with a TagOp value of Transfer, it is required that the Requester has copies of both the data and the tags.

B12.5 Write transactions

Different fields to support MTE are distributed between the Request and Data message of a Write transaction. The TagOp field, which indicates the operation to be performed on the tags in the WriteData message, is included in both the Request and WriteData message. The Request also includes the TagGroupID field to provide an identifier for the pass/fail response for a request that requires a Tag Match operation. When the TagOp field in the Request is *Match*, the Excl field must be 0.

Note

The use of the TagGroupID field is IMPLEMENTATION DEFINED. Typically, TagGroupID can be used to identify the Exception level and TTBR which a response relates to.

The TagOp value in the WriteData message is typically the same as the value in the Request message, except when either the write data is snooped out or the write is canceled. Whether or not to perform a Tag Match must be decided based on the TagOp value in the WriteData, when the TagOp values in the WriteData and Write request are different.

The WriteData message also includes a Tag Update (TU) bit per tag, which is applicable when TagOp is *Update*.

B12.5.1 Permitted TagOp values

This section describes the permitted WriteData TagOp values for each of the permitted TagOp values in the Write request message.

When the TagOp field in the Request is *Invalid*, the Memory Tagging fields in the WriteData must be set to 0 and ignored by the Completer.

When the TagOp field in the Request is *Transfer*, the TagOp field in the WriteData can be:

- *Transfer*: The Tags are Clean and must be handled appropriately by the Completer.
- *Invalid*: This is possible only if either the cached copy is invalidated, or the Write transaction is canceled.

In a WriteClean transaction where the Request TagOp is *Transfer*, the TagOp in the Write data is not permitted to be changed to *Update*.

When the TagOp field in the request is *Update*, the TagOp field in the WriteData can be:

- *Update*: The Dirty tags must be cached or written to memory.
- *Transfer*: The Tags are Clean. This is possible if the Dirty tags have been snooped out.
- *Invalid*: This is possible only if either the cached copy is invalidated or the Write transaction is canceled.

When the TagOp field in the request is *Match* the TagOp field in the WriteData can be:

- *Match*: The appropriate Tag Match must be performed at the Completer.
- *Invalid*: This is possible only if the Write transaction is canceled.

B12.5.2 TagOp, TU, and tags relationship

This section describes the relationship between TagOp, TU, and tags in different Write transactions:

- For all Write requests with TagOp *Invalid*, the Memory Tagging fields must be set to 0 and ignored by the Completer.
- For WriteBackFull and WriteCleanFull with TagOp:

  - *Transfer*: Clean tags must be returned. TU bits are inapplicable and must be set to 0.
  - *Update*: All TU bits must be asserted.
  - *Match*: Not permitted.
- For WriteBackPtl with TagOp:
  - *Transfer*: Not permitted.
  - *Update*: Not permitted.
  - *Match*: Not permitted.
- For WriteNoSnpFull with TagOp:
  - *Transfer*: TU bits are inapplicable and must be set to 0. Clean tag transfer is permitted from the Request Node to the Home Node as well as the Home Node to the Subordinate Node.
  - *Update*: All TU bits must be asserted.
  - *Match*: TU bits are inapplicable and must be set to 0.
- For WriteNoSnpDef with TagOp:
  - *Transfer*: Not permitted.
  - *Update*: Not permitted.
  - *Match*: Not permitted.
- For WriteUniqueFull and WriteUniqueFullStash with TagOp:
  - *Transfer*: Not permitted.
  - *Update*: All TU bits must be asserted.
  - *Match*: TU bits are inapplicable and must be set to 0.
- For WriteNoSnpPtl, WriteUniquePtl and WriteUniquePtlStash with TagOp:
  - *Transfer*: Not permitted.
  - *Update*: Any combination of TU and BE bits, including none or all, can be asserted.
  - *Match*: TU bits are inapplicable and must be set to 0. Tag Match must be performed for only those tags that have at least one corresponding BE bit asserted. A Tag Match must not be performed when all BE bits are set to 0.
- For WriteEvictFull, WriteEvictOrEvict and with TagOp:
  - *Transfer*: Clean tags must be returned. TU bits are inapplicable and must be set to 0.
  - *Update*: Not permitted.
  - *Match*: Not permitted.
- For WriteNoSnpZero and WriteUniqueZero only TagOp *Invalid* is permitted.

For a Write request with a TagOp of *Match* the tags within the size wrap boundary can take any value, and the tags outside of size are inapplicable and can also take any value.

In the WriteDataCancel Write data response, irrespective of the TagOp value in the Write request, the MTE fields are inapplicable and must be set to 0.

In Write data, with TagOp *Invalid*, all the TU bits and all the Tag bits must be set to 0.

B12.6 Dataless transactions

MakeUnique is the only Dataless transaction that supports the use of the TagOp field. In all other Dataless transactions, the TagOp field is inapplicable and must be set to all zeros.

The TagOp value in MakeUnique request can be Invalid or Update only. A Request TagOp value of *Update* is an indication that the Requester updates the tags along with data. Home, in response to MakeUnique, is permitted to use SnpMakeInvalid only when either the Request TagOp value is *Update* or the Home knows that the Snoopee does not have Dirty tags.

The only permitted TagOp value in response to MakeUnique is *Invalid*.

Cache Maintenance Operations must operate on both the data and the corresponding memory tags. A MakeInvalid that permits dropping of Dirty data without writing to memory must write Dirty tags to memory.

B12.7 Atomic transactions

TagOp is applicable in Atomic transactions. The permitted values for the field are *Invalid* and *Match*.

The Physical Tags to be matched are provided in the write data and correspond to the valid data bytes in AtomicLoad, AtomicStore, and AtomicSwap. Only one set of tag bits are applicable in these Atomic transactions because the maximum data size is 8-byte. The remaining tag bits in the set are not applicable and must be set to 0.

In AtomicCompare with a data size of up to 16 bytes, the valid data still corresponds to only one set of tag bits.

In AtomicCompare with data size of 32-bytes the individual compare and swap data is only 16-bytes. Only one set of Physical Tag bits is required to be matched when TagOp is set to *Match*. Physical Tags must be duplicated to cover both Compare and Swap data. The Completer is permitted to use either set of Physical Tags to perform Tag Match.

The permitted TagOp values in the CompData response to Non-store Atomic transactions are *Invalid* and *Transfer*.

The Write data for Atomic transactions where TagOp is *Invalid* must have all TU bits and all the Tag bits set to 0.

B12.8 Stash transactions

In StashOnce and StashOnceSep transactions, TagOp values of *Invalid* and *Transfer* are permitted.

For Stash snoop interaction with Memory Tagging see B12.9.3 *Stash snoops*.

B12.9 Snoop requests

This section describes the permitted use by Home of the following snoop types:

- B12.9.1 *Non-forwarding snoops*
- B12.9.2 *Forwarding snoops*
- B12.9.3 *Stash snoops*

B12.9.1 Non-forwarding snoops

Home can use any applicable Non-forwarding snoop when tags are needed by the request. If the Snoop response returns data to Home but not tags, the Home is responsible for obtaining tags before returning the Data response to the Requester.

A Home in response to WriteUniqueFull, WriteUniqueFullStash, MakeUnique, and MakeInvalid must not use the SnpMakeInvalid snoop request unless either:

- The transaction is WriteUniqueFull, WriteUniqueFullStash, or MakeUnique with a TagOp value of *Update*.
- The Home can determine that the Snoopee does not hold Dirty tags.

Note

By sending a ReadUnique with TagOp *Fetch*, a Requester is indicating its willingness to update the full cache line but not the tags. Home must not use SnpMakeInvalid in response to such a request, to avoid losing modified tags at the Snoopee.

B12.9.2 Forwarding snoops

The use by Home of the Forwarding snoop is permitted only if the request does not need to fetch tags. A Forwarding snoop is permitted to be sent even when the Snoopee has Valid tags. If the Snoopee has Dirty tags when responding to a Forwarding snoop, Dirty tags must not be forwarded to the Requester nor split dirtiness or uniqueness of tags and data.

A Snoopee is always permitted to forward Clean tags to the Requester.

When tags are Clean, the Snoopee must follow the same data transfer rules as in the Non-MTE case.

When tags are Dirty, the Snoopee must follow the rules:

- For the invalidating Forwarding snoop SnpUniqueFwd and SnpPreferUniqueFwd handled as an invalidating snoop:
  - Must not forward data to the Requester.
  - Must return data and tags to the Home.
- For the Non-invalidating Forwarding snoops SnpCleanFwd, SnpSharedFwd, SnpNotSharedDirtyFwd, SnpPreferUniqueFwd handled as a non-invalidating snoop:
  - Must treat the snoop as SnpCleanFwd.

    Note

    Same as SnpNotSharedDirtyFwd.

- For SnpOnceFwd:

- Permitted to return Dirty tags to the Home.
- Dirty tags must be kept at the Snoopee if the data transfer to Home is not performed.
- Dirty tags must be written back to Home if the cache line is being invalidated or Dirty data is being written back to Home.

A Snoopee is permitted, but not required, to convert any Forwarding snoop to its corresponding Non-forwarding snoop.

Note

This property is similar to that in the Non-MTE case.

See B12.9.4 *Permitted TagOp values in Snoop responses* for more details.

B12.9.3 Stash snoops

Because the SNP channel does not include a TagOp field, the Home cannot forward the TagOp intentions of the Requester to the Stash target.

The permitted TagOp values in Snoop responses are:

- *Invalid* in response to SnpStash*.
- *Invalid*, *Transfer*, and *Update* in response to SnpUniqueStash.
- *Invalid* in response to SnpMakeInvalidStash.

See B12.9.4 *Permitted TagOp values in Snoop responses* for more details.

The requirements for determining the TagOp value in the Read request implied from the Data Pull request in the Snoop response are as follows:

For a DataPull request in response to SnpStash*:

- If the TagOp value in the original request is *Transfer*, it is recommended that Clean tags are returned in response to the Data Pull request.
- If the TagOp value in the original request is *Invalid*, it is recommended that Clean tags are returned in response to the Data Pull request if tags are available.

For a DataPull request in response to SnpUniqueStash:

- If tags are available when data is returned, irrespective of the presence or absence of data in the Snoop response and for any cache state, it is recommended that the Clean tags are returned in response to a Data Pull request.

B12.9.4 Permitted TagOp values in Snoop responses

The permitted tag field values in Snoop responses are:

- For SnpResp, the TagOp field is inapplicable and must be set to 0.
- For SnpRespDataPtl, the permitted TagOp value is Invalid. Both the TU and the Tag field must be set to 0 and ignored by the receiver.
- For SnpRespData:
  - *Invalid*. All Tag fields must be set to 0.
  - *Transfer*. Clean tags must be returned. TU bits are inapplicable and must be set to 0.

- *Update*. All TU bits must be asserted. The data state must include Pass Dirty.
- *Match*. Not permitted.

B12.10 Home to Subordinate transactions

For a Read request to the Subordinate:

- The permitted TagOp values are *Invalid*, *Transfer*, and *Fetch*.

For a Write request to the Subordinate:

- The permitted TagOp values are *Invalid*, *Transfer*, *Update*, and *Match*.

For an Atomic request to the Subordinate:

- The permitted TagOp values are *Invalid*, and *Match*.

A ReadNoSnp or ReadNoSnpSep with TagOp *Transfer* or *Fetch* can be used to fetch tags from the Subordinate. Tags can be returned from a Subordinate to the Home directly or sent to the Requester using DMT. When TagOp *Fetch* is used, the Subordinate is not required to return valid data. TagOp *Fetch* is permitted only with a data size of 64 bytes.

When memory needs to be updated with tags, WriteNoSnp with TagOp *Update* must be used. When only tags need to be updated the data BE bits can be set to all 0.

When TagOp is *Match*, the TagGroupID in WriteNoSnp and Atomic transactions to the Subordinate are applicable, and these values must be returned in the TagMatch response.

When Atomic operations are performed at the Subordinate Node, the Home is permitted to include TagOp *Match* in Atomic requests to the Subordinate. Where the tag match is performed at the Subordinate, in both non-Store Atomic and Store Atomic transactions, the TagMatch response TgtID must be obtained from the ReturnNID value in the request. To support this feature, it is required that the ReturnNID in the Atomic Store from a Home Node to a Subordinate Node is applicable and must be set to the same value as the SrcID in the request.

Note

Applicability of the ReturnNID in non-Store Atomic transactions is already a requirement in the previous version of this specification.

B12.11 Error response

The following sub-sections describe Error response handling for:

- B12.11.1 *Tag Match*
- B12.11.2 *Non-Tag Match errors*
- B12.11.3 *MTE not supported*

B12.11.1 Tag Match

Write and Atomic transactions that have TagOp value of *Match* in the Request must return the results of the Tag Match operation. The results are returned using the TagMatch message. The transaction must proceed as normal irrespective of the Tag Match result. The TagMatch response must be sent even if the WriteData is canceled or a Tag Match is not performed.

Note

Using a separate TagMatch message adds complexity and extra messages to the Write and Atomic transactions. A separate TagMatch message also has the advantage to provide a sufficiently accurate response mechanism. Using a separate response does not delay the sending of the Comp response.

The TagMatch message characteristics are:

- The Home Node or Subordinate Node performing the Tag Match operation sends the TagMatch response.
- The TgtID value in the message is copied from:
  - SrcID in the request if the Completer is a Home Node.
  - ReturnNID in the request if the Completer is a Subordinate Node. The ReturnNID can point to the Requester or the Home. When the Subordinate returns the TagMatch response to the Home, if required by the transaction, the Home is responsible for forwarding the response to the Requester.
- The response must return the TagGroupID value from the request.
- The TraceTag field is inapplicable and can take any value.
- The Resp field value in the response indicates the Tag Match state as either pass or fail. See B13.10.48 *Response status, Resp*.
- The TagMatch response can be sent as soon as the Completer can determine the result. TagMatch is permitted to be sent without waiting for data. This can occur when the Completer does not support MTE.
- The Resp value in the TagMatch response must be:
  - A Fail when MTE is not supported.
  - A Pass when MTE is supported but the Tag Match is not performed. For example, when a write targets an MTE capable location, but fails an access permissions check.
  - Accurate, if the match is performed.

B12.11.2 Non-Tag Match errors

Permitted Data and Non-data Error responses to requests are independent of the presence or absence of Memory Tagging.

Permitted RespErr field values in the TagMatch response are OK, DERR, and NDERR. See B9.1.4 *Error response use by transaction type*.

When the RespErr field in a response is NDERR:

- The TagOp in the response to the Requester is inapplicable and can take any value.
- The Snoop response must be Invalid.

Poison on tags is not supported. See B9.2.1 *Poison*.

B12.11.3 MTE not supported

When a Completer does not support MTE for the address in the request, the Completer must send a TagOp value of *Invalid* in response to a Read transaction with a TagOp value of *Transfer* or *Fetch*. The returned tags must be set to 0. The returned tags can be cached as Clean by the Requester.

A Completer in response to a Read request to a Non-cacheable or Device memory location must set the TagOp value in the response to *Invalid*. A Completer is expected to give an OK response to an MTE operation, unless a different response can be given to an equivalent non-MTE operation.

When a Write request with TagOp *Match* is received and a Completer does not support MTE for the address in the request, the Completer is still required to send a TagMatch response. The Resp field value must indicate that the Tag Match failed. The Completer is permitted, but not required, to wait for the write data before sending the TagMatch response.

B12.12 Requests and permitted tag operations

Table B12.3 shows a summary of the permitted TagOp field values in the different requests. The following key is used:

Y Yes, permitted

- Not permitted

Table B12.3: Permitted TagOp values for each request type

Requests	**Tag operation**				
	Invalid	*Transfer*	*Update*	*Match*	*Fetch*
ReadOnce	Y	Y	-	-	-
ReadClean					
ReadShared					
ReadNotSharedDirty					
ReadPreferUnique					
ReadOnceMakeInvalid					
ReadOnceCleanInvalid					
ReadUnique	Y	Y	-	-	Y
ReadNoSnp	Y	Y	-	-	Y
ReadNoSnpSep					
MakeReadUnique	Y	Y	-	-	-
CleanShared	Y	-	-	-	-
CleanSharedPersist					
CleanSharedPersistSep					
CleanUnique	Y	-	-	-	-
CleanInvalid					
CleanInvalidPoPA					
MakeInvalid					
MakeUnique	Y	-	Y	-	-
Evict	Y	-	-	-	-
StashOnceUnique	Y	Y	-	-	-
StashOnceSepUnique					
StashOnceShared					
StashOnceSepShared					
WriteNoSnpFull	Y	Y	Y	Y	-
WriteNoSnpDef	Y	-	-	-	-

Continued on next page

Table B12.3 – *Continued from previous page*

Requests	**Tag operation**				
	Invalid	*Transfer*	*Update*	*Match*	*Fetch*
WriteUniqueFull WriteUniqueFullStash	Y	-	Y	Y	-
WriteNoSnpPtl WriteUniquePtl WriteUniquePtlStash	Y	-	Y	Y	-
WriteBackFull WriteCleanFull	Y	Y	Y	-	-
WriteBackPtl	Y	-	-	-	-
WriteEvictFull WriteEvictOrEvict	Y	Y	-	-	-
WriteNoSnpFull + (P)CMO	Y	Y	Y	-	-
WriteNoSnpPtl + (P)CMO	Y	-	Y	-	-
WriteUniqueFull + (P)CMO	Y	-	-	-	-
WriteUniquePtl + (P)CMO	Y	-	-	-	-
WriteBackFull + (P)CMO	Y	Y	Y	-	-
WriteCleanFull + (P)CMO	Y	Y	Y	-	-
WriteNoSnpZero	Y	-	-	-	-
WriteUniqueZero	Y	-	-	-	-
Atomic*	Y	-	-	Y	-
PrefetchTgt	Y	Y	-	-	-
PCrdReturn DVMOp	Y	-	-	-	-

The TagOp field in ReqLCrdReturn, DatLCrdReturn, and RspLCrdReturn is inapplicable and can take any value.

The Tag and TU fields in DatLCrdReturn are inapplicable and can take any value.

B12.13 TagOp field use summary

The following sections give a summary of the use of the TagOp field in messages in different channels:

REQ channel message:

- Read* and MakeReadUnique transaction:
  - TagOp field can be *Invalid*, *Transfer*, or *Fetch*.
  - TagOp *Fetch* is permitted only in ReadUnique, ReadNoSnp, and ReadNoSnpSep transactions.
  - TagOp field *Transfer* indicates whether Allocation Tags must be returned alongside read data.
  - TagOp field *Fetch* indicates only valid tags are required to be returned. Returned data is not required to be valid.
  - For all other REQ channel messages, the TagOp field is inapplicable and must be 0.
  - The TagOp field *Match* cannot be used in an Exclusive transaction.
- Write transaction:
  - TagOp field can be *Invalid*, *Transfer*, *Match* or *Update*.
  - TagOp field *Transfer* indicates Clean tags are being transferred and tags can be cached alongside the data.
  - TagOp field *Match* indicates a Match check is required between the Physical Tags in the message and the Allocation Tags at the memory location.
  - TagOp field *Update* indicates Dirty tags are being passed, which must update the Allocation Tag values.
- MakeUnique transaction:
  - TagOp field can be *Invalid* or *Update*.
  - TagOp field *Update* indicates all tags are written to.
- Atomic transaction:
  - TagOp field can be *Invalid* or *Match*.
  - TagOp field *Match* indicates whether a Tag Match is required.
- StashOnce transaction:
  - TagOp field can be: *Invalid* or *Transfer*.
  - TagOp field *Transfer* indicates whether Allocation Tags should be stashed alongside Stash data.
- PrefetchTgt transaction:
  - TagOp value can be *Invalid* or *Transfer*.
- For all the other REQ channel messages, the TagOp field is inapplicable and must be 0.

DAT channel message:

- For Read data, the TagOp field indicates whether the Allocation Tags sent along with the data are Invalid, Clean, or Dirty.
- For Snoop data, the TagOp field indicates whether the Allocation Tags sent in the Snoop response are Invalid, Clean, or Dirty.
- For write data, the TagOp field indicates whether the Allocation Tags sent in the write data are Invalid, Clean, Dirty, or a Match check is required. The TagOp value must be the same as in the Request message except when a snoop has downgraded the state of the tags or the write has been canceled.

RSP channel message:

- For a Comp response, the TagOp field is only used in response to a MakeReadUnique transaction and is used to indicate if responsibility for Dirty tags is being passed to the Requester or not.
- For all other RSP channel messages, the TagOp field is inapplicable and must be 0.

SNP message:

- The TagOp field is not present in the SNP channel.

Chapter B13
Link Layer

This chapter describes the Link layer that provides a streamlined mechanism for packet-based communication between nodes and the interconnect across links. It contains the following sections:

B13.1 Introduction

The Link layer provides a streamlined mechanism for packet-based communication between nodes and the interconnect.

The Link layer defines packet and flit formats, and flow control across a link.

Figure B13.1 shows a typical system using link-based communication.

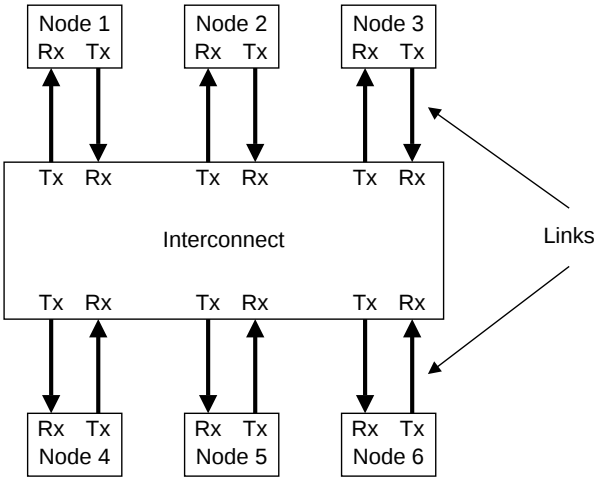

Figure B13.1: System using link-based communication

Interface parity signals, which are discussed in B9.3 *Use of interface parity* are not included in this chapter.

B13.2 Link

Flit communication occurs between a Transmitter and a Receiver pair.

The connection between a Transmitter and a Receiver is referred to as a link.

Two-way communication between a node and the interconnect requires a pair of links.

Figure B13.2 shows the link requirements.

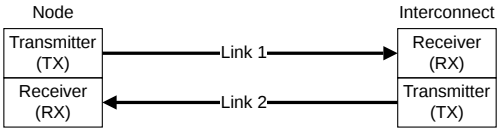

Figure B13.2: Two-way link communication

B13.2.1 Outbound and inbound links

The link used by a Transmitter to send packets is defined as the outbound link.

The link used by a Receiver to receive packets is defined as the inbound link.

Figure B13.3 shows the outbound and inbound links at a node. The interface at the interconnect has a complementary pair of links.

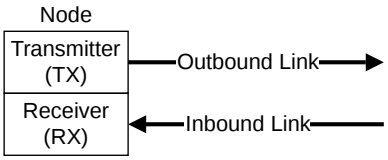

Figure B13.3: Outbound and inbound links

B13.3 Flit

A flit is the basic unit of transfer in the Link layer.

Packets are formatted into flits and transmitted across a link. There are two types of flits:

Protocol flit A Protocol flit carries a protocol packet in its payload. Every protocol packet is mapped into exactly one protocol flit.

Link flit A Link flit carries messages associated with link maintenance. For example, a Transmitter uses a Link flit to return a Link layer Credit, also referred to as an L-Credit, to the Receiver during a link deactivation sequence. Link flits originate at a link Transmitter and terminate at the link Receiver connected at the other side of the link.

B13.4 Channel

The Link layer provides a set of channels for flit communication.

Each channel has a defined flit format that has multiple fields and some of the field widths have multiple possible values. In some cases, the defined flit format can be used on both an inbound and an outbound channel.

Table B13.1 shows the channels, and the mapping onto the Request Node and Subordinate Node component channels.

Table B13.1: Channels' mapping onto the Request Node and Subordinate Node component channels

Channel	Description	Usage	Request Node channel	Subordinate Node channel
REQ Request	The request channel transfers flits associated with request messages such as Read requests and Write requests. See B13.8.1 *Request, REQ, channel*.	All Requests	TXREQ	RXREQ
RSP Response	The response channel transfers flits associated with response messages that do not have a data payload such as write completion messages. See B13.8.2 *Response, RSP, channel*.	Responses from the Completer	RXRSP	TXRSP
		Snoop Response and Completion Acknowledge	TSRSP	-
SNP Snoop	The snoop channel transfers flits associated with Snoop and SnpDVMOp Request messages. See B13.8.3 *Snoop, SNP, channel*.	All Snoop requests	RSRSP	-
DAT Data	The data channel transfers flits associated with protocol messages that have a data payload such as read completion and WriteData messages. See B13.8.4 *Data, DAT, channel*.	WriteData, and Snoop response data from a Request Node	TXDAT	RXDAT
		Read data	RXDAT	TXDAT

B13.4.1 Channel dependencies

The following dependencies are permitted between the channels in the protocol.

For a Request Node:

- Must make forward progress on the inbound SNP channel without requiring forward progress on outbound REQ channel.
- Permitted, but not required, to wait for forward progress on the outbound RSP channel before making forward progress on the inbound SNP channel.
- Permitted, but not required, to wait for forward progress on the outbound DAT channel before making forward progress on the inbound SNP channel.
- Must make forward progress on the inbound RSP channel without requiring forward progress on any other channel.
- Must make forward progress on the inbound DAT channel without requiring forward progress on any other channel.

Note

The requirement that a Request Node must make forward progress on the inbound RSP and DAT channel, without requiring forward progress on any other channel, means that a Request Node must be able to accept all Comp and CompData responses for outstanding transactions without sending any CompAck responses.

For a Subordinate Node:

- Permitted, but not required, to wait for forward progress on the outbound RSP channel before making forward progress on the inbound REQ channel.
- Must make forward progress on the inbound REQ channel without requiring forward progress on the outbound DAT channel.
- Must make forward progress on the inbound DAT channel without requiring forward progress on any other channel.

B13.5 Port

A Port is defined as the set of all links at the interface of a node.

Figure B13.4 shows the relationship between links, channels, and port. See B13.6 *Node interface defintions* for the specific node requirements. See B13.8 *Channel interface signals*, and Chapter B14 *Link Handshake* for signal details.

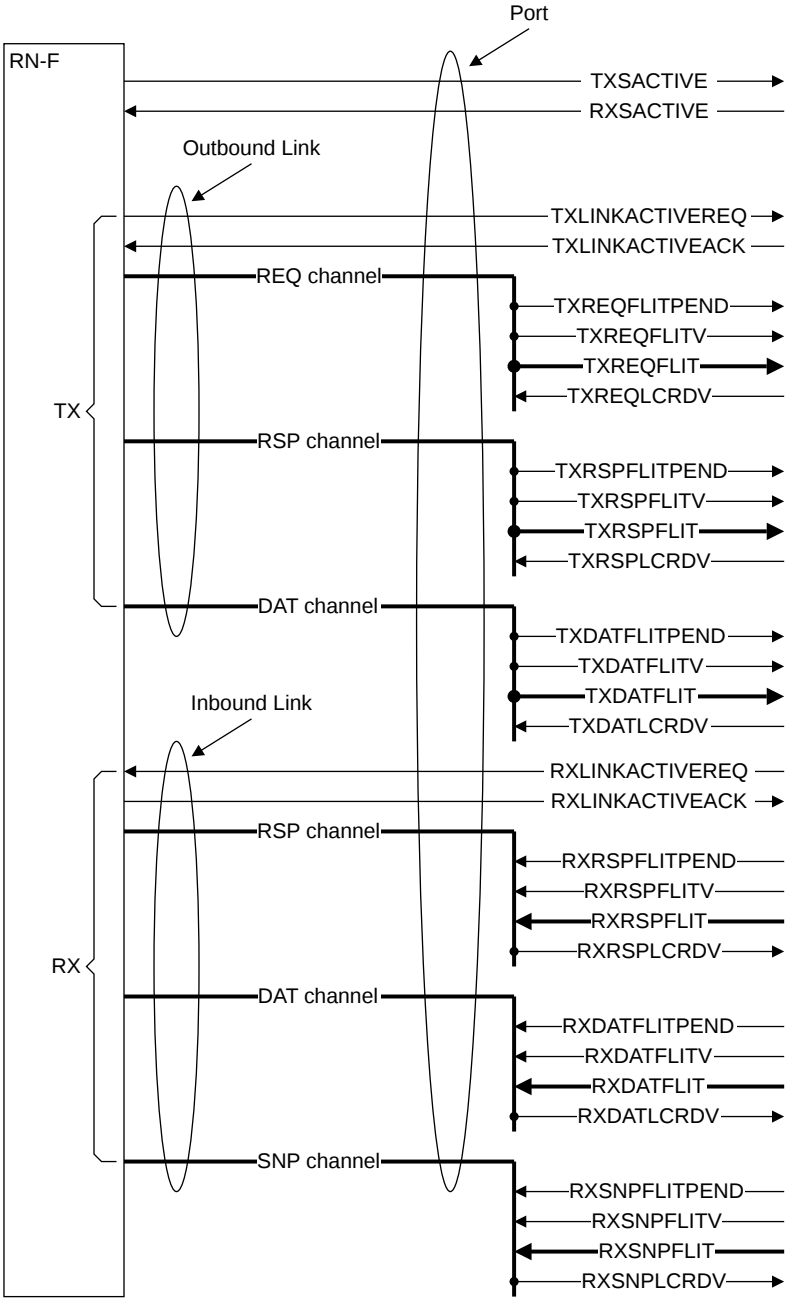

Figure B13.4: Relationship between links, channels, and ports

B13.6 Node interface defintions

The nodes exchange messages by sending flits across the node interface. This section describes two node types of node interfaces supported by the CHI protocol:

- B13.6.1 *Request Nodes*
- B13.6.2 *Subordinate Nodes*

Note

The LINKACTIVE interface pins and signals used by each node for link management are described in Chapter B14 *Link Handshake*.

B13.6.1 Request Nodes

This section describes the Request Node interfaces:

- B13.6.1.1 *RN-F*
- B13.6.1.2 *RN-D*
- B13.6.1.3 *RN-I*

B13.6.1.1 RN-F

The RN-F interface uses all channels and is used by a fully coherent Requester such as a core or cluster.

Figure B13.5 shows the RN-F interface.

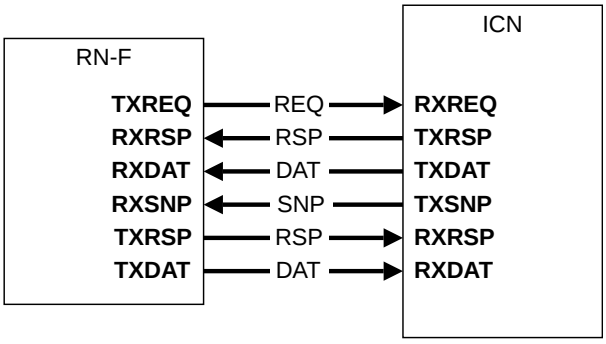

Figure B13.5: RN-F interface

B13.6.1.2 RN-D

The RN-D interface uses all channels and is used by an IO coherent node that processes DVM messages. Use of the SNP channel is limited to DVM transactions. See B8.2 *DVM transaction flow* for details.

Figure B13.6 shows the RN-D interface.

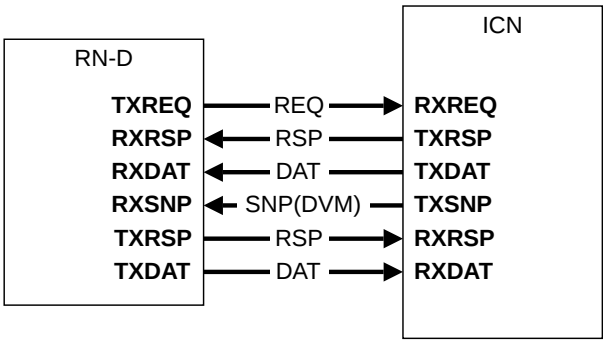

Figure B13.6: RN-D interface

B13.6.1.3 RN-I

The RN-I interface uses all channels, with the exception of the SNP channel, and is used by an IO coherent Request Node such as a GPU or IO bridge. A SNP channel is not required because an RN-I node does not include a hardware-coherent cache or TLB.

Figure B13.7 shows the RN-I interface.

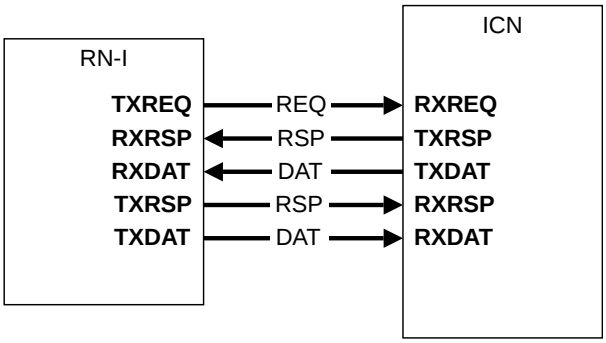

Figure B13.7: RN-I interface

B13.6.2 Subordinate Nodes

This section describes the Subordinate Node interfaces:

- B13.6.2.1 *SN-F and SN-I*

B13.6.2.1 SN-F and SN-I

The SN-F and SN-I interfaces are identical and use a RX request channel, a TX response channel, a TX data channel, and an RX data channel. The SN-F and SN-I receive request messages from the interconnect, and return response messages to the interconnect. However, the SN-F and SN-I receive different types of transactions.

Figure B13.8 shows the SN-F and SN-I interface.

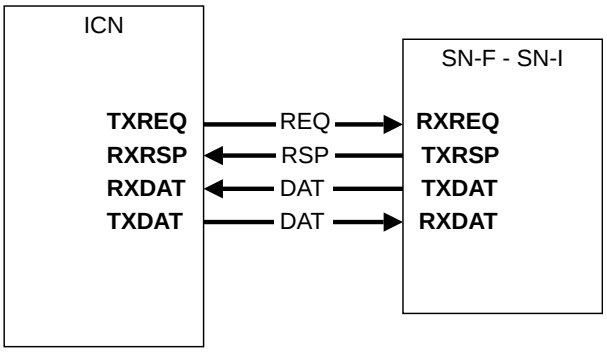

Figure B13.8: SN-F and SN-I interface

B13.7 Increasing inter-port bandwidth

The available bandwidth at a node interface can be increased in several ways. Two architectural methods permitted are detailed in the following sections:

- B13.7.1 *Multiple interfaces*
- B13.7.2 *Replicated channels on a single interface*

B13.7.1 Multiple interfaces

The simplest method for a component to increase the available bandwidth is to have multiple interfaces. A complete interface can be duplicated. The number of times an interface on a node is duplicated is IMPLEMENTATION DEFINED.

Figure B13.9 shows an example of duplicated interfaces.

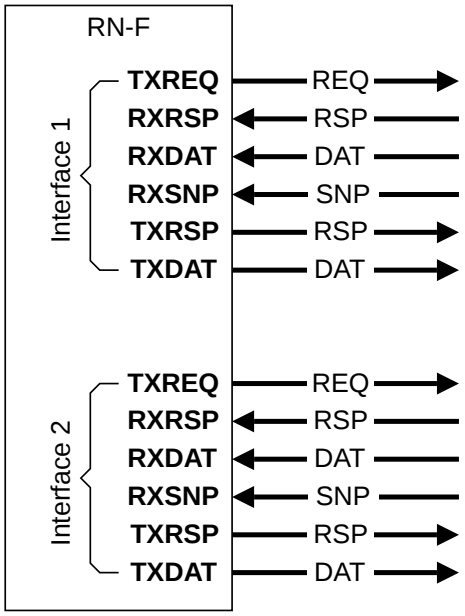

Figure B13.9: Multiple interface example

The main features of this method of increasing bandwidth across two interfaces are:

- Each interface has its own:
  - NodeID
  - TxnID pool
  - Set of **SACTIVE** signals
  - Set of **LINKACTIVE** signals
  - Set of **SYSCOREQ/SYSCOACK** signals
  - Set of optional broadcast control pins
- Each duplicate interface must be treated as an independent interface:
  - If one interface allocates the cache line, another duplicate interface cannot deallocate that cache line.

  - When responding to a request, the Completer must send the response on the same interface as the one used by the request.
  - A snoop must be sent to the same interface that was used for the transaction that caused the allocation of a cache line.
  - A transaction from one interface must make forward progress without requiring forward progress of a transaction on a duplicate interface.
- All channels must be replicated even when only a subset of the channels needs increased bandwidth.

B13.7.1.1 Address striping

An optional optimization is permitted where a Request Node can specify the address striping that is used to select between multiple interfaces. This can be specified for any number of interfaces.

A Request Node is permitted to use address striping to guide its requests to the appropriate interface without declaring the striping algorithm being used.

A Home Node typically can filter snoops based on a snoop filter. If the snoop filter is precise, the node ID of the Requester that cached the cache line is tracked and a snoop is sent for a subsequent request to the same cache line on a single interface. A snoop filter that does not track precisely, or is sized to the number of components in the system instead of the number of Request Node interfaces, is not able to isolate the single interface needed to send the snoop on, unless the snoop filter knows and uses the same address striping algorithm as the Requester.

When a Request Node does not declare its striping algorithm, the snoop filters either need to be larger or the Home must send redundant snoops. It is recommended that a Request Node that uses address striping advertises the striping algorithm in order to be used by the Home Node.

The address striping used by a Request Node can be specified by a hash function.

B13.7.1.2 Hash function example

This section describes a suggested hash function to be used to distribute requests across multiple REQ interfaces. The same hash function can be used to distribute snoops across multiple available SNP interfaces. The steps in generating the interface number are:

1. The cache line aligned input address is first filtered through a predefined Hash Mask. The most significant address bits that are not used must be set to all 0 before filtering the address. In this example, the result of the filtering is Mask_Result.
2. The individual bits of Mask_Result are XORed to obtain the target interface.

Where the number of interfaces is a power-of-two:

- For 2 interfaces:

Interfaces[0] = Mask_Result[n-1] ^ Mask_Result[n-2] . . . Mask_Result[7] ^ Mask_Result[6]

- For 4 interfaces:

Interfaces[1:0] = Mask_Result[n-1:n-2] ^ Mask_Result[n-3:n-4] . . . Mask_Result[9:8] ^ Mask_Result[7:6]

- For 8 interfaces:

Interfaces[2:0] = Mask_Result[n-1:n-3] ^ Mask_Result[n-4:n-6] . . . Mask_Result[11:9] ^ Mask_Result[8:6]

This example does not cover the situation where the number of interfaces is not a power-of-two.

B13.7.2 Replicated channels on a single interface

Rather than replicating a complete interface through more complex methods, a more efficient method to increase the available interface bandwidth is to selectively replicate the channels that require greater bandwidth.

Figure B13.10 shows an example of replicated channels.

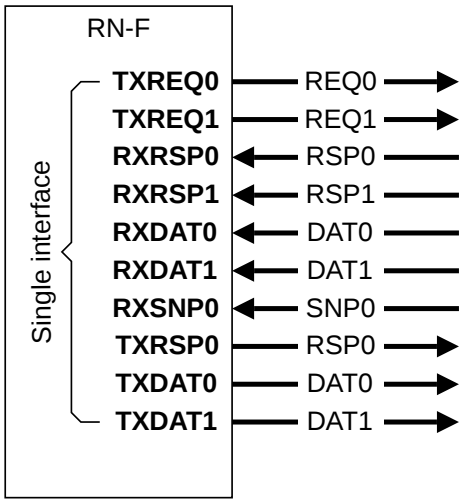

Figure B13.10: Replicated channels example

B13.7.2.1 Features

The main features of this method of increasing available bandwidth are described in this section.

Each channel can be selectively replicated. There are no restrictions on which channels are replicated. Typically, replication of a channel is based on the expected bandwidth required on that channel. For example, in Figure B13.10, TXREQ is duplicated as TXREQ0, TXREQ1, whereas RXSNP is not duplicated and has only RXSNP0. The characteristics of the replicated channel interface are:

- All replicated DAT subchannels corresponding to a single DAT channel must be of the same width.
- The complete interface must use:
  - The same NodeID
  - A single TxnID pool
- Messages within a transaction can use any subchannel:
  - A response message does not need to use the same subchannel as the request. For example, Request on TXREQ0 can give a response on either RXRSP0 or RXRSP1.
  - Multiple response messages for a single request can come on any subchannel. For example, DBIDResp for a write transaction is received on RXRSP0 whereas the corresponding Comp can be received on RXRSP1.
- Like non-replicated channels, replicated channels do not provide any in-channel ordering guarantees.
- All link crediting is done on a subchannel basis.
  - Cannot use the credit for TXREQ0 to send the flit on TXREQ1.
  - Credits are required to be provided by the Receiver on all subchannels.

- Protocol credits are for the combined TXREQ channel.
- There is no support for powering down a subchannel individually.
- The two parts of a DVM snoop can come on either subchannel. Each part can be on a different subchannels.
- The number of subchannels on two connected interfaces must match.
- There must be only one set of **SACTIVE**, **LINKACTIVE** and **SYSCOREQ/SYSCOACK** signals, and optional broadcast control pins.
- Interface property CCF_Wrap_Order is not permitted to be set to True when the interface includes replicated DAT channels.

Note

When a channel has been replicated on an interface, it is IMPLEMENTATION DEFINED which of the subchannels is used for a transfer.

B13.8 Channel interface signals

This section describes the channel interface signals. It contains the following sections:

B13.8.1 Request, REQ, channel

Figure B13.11 shows the REQ channel interface pins, where R is the width of **REQFLIT**.

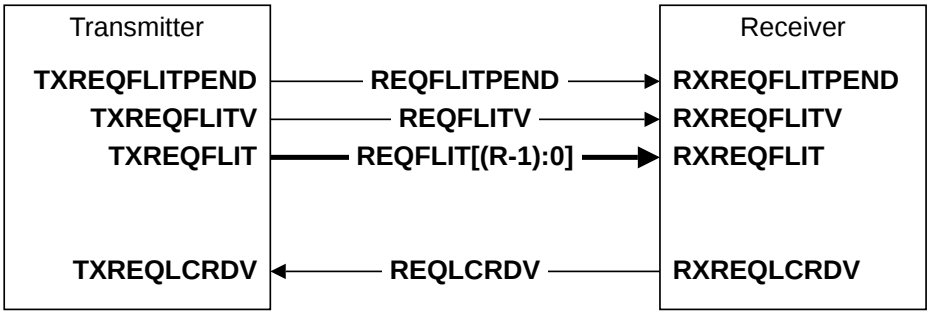

Figure B13.11: REQ channel interface pins

Table B13.2 shows the REQ channel interface signals.

Table B13.2: REQ channel interface signals

Signal	Description
REQFLITPEND	Request Flit Pending. Early indication that a request flit could be transmitted in the following cycle. See B14.4 *Flit level clock gating*.
REQFLITV	Request Flit Valid. The transmitter sets this signal HIGH to indicate when **REQFLIT[(R-1):0]** is valid.
REQFLIT[(R-1):0]	Request Flit. See B13.9.1 *Request flit* for a description of the request flit format.
REQLCRDV	Request L-Credit Valid. The receiver sets this signal HIGH to return a request channel L-Credit to a transmitter. See B14.2.1 *L-Credit flow control*.

B13.8.2 Response, RSP, channel

Figure B13.12 shows the RSP channel interface pins, where T is the width of **RSPFLIT**. The same interface is used for both inbound and outbound RSP channels.

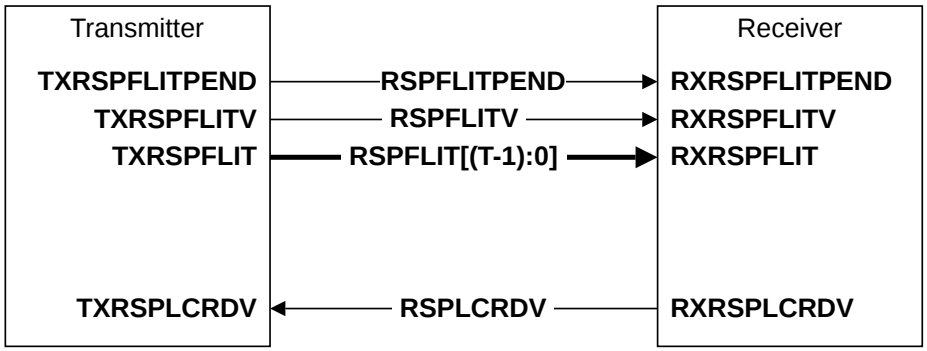

Figure B13.12: RSP channel interface pins

Table B13.3 shows the RSP channel interface signals.

Table B13.3: RSP channel interface signals

Signal	Description
RSPFLITPEND	Response Flit Pending. Early indication that a response flit could be transmitted in the following cycle. See B14.4 *Flit level clock gating*.
RSPFLITV	Response Flit Valid. The transmitter sets this signal HIGH to indicate when **RSPFLIT[(T-1):0]** is valid.
RSPFLIT[(T-1):0]	Response Flit. See B13.9.2 *Response flit* for a description of the response flit format.
RSPLCRDV	Response L-Credit Valid. The receiver sets this signal HIGH to return a response channel L-Credit to a transmitter. See B14.2.1 *L-Credit flow control*.

B13.8.3 Snoop, SNP, channel

Figure B13.13 shows the SNP channel interface pins, where S is the width of **SNPFLIT**.

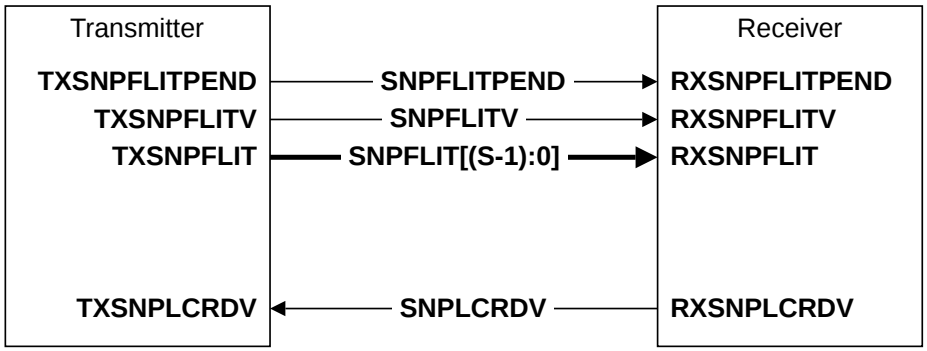

Figure B13.13: SNP channel interface pins

Table B13.4 shows the SNP channel interface signals.

Table B13.4: SNP channel interface signals

Signal	Description
SNPFLITPEND	Snoop Flit Pending. Early indication that a snoop flit could be transmitted in the following cycle. See B14.4 *Flit level clock gating*.
SNPFLITV	Snoop Flit Valid. The transmitter sets this signal HIGH to indicate when **SNPFLIT[(S-1):0]** is valid.
SNPFLIT[(S-1):0]	Snoop Flit. See B13.9.3 *Snoop flit* for a description of the snoop flit format.
SNPLCRDV	Snoop L-Credit Valid. The receiver sets this signal HIGH to return a snoop channel L-Credit to a transmitter. See B14.2.1 *L-Credit flow control*.

B13.8.4 Data, DAT, channel

Figure B13.14 shows the DAT channel interface pins, where D is the width of **DATFLIT**. The same interface is used for both inbound and outbound DAT channels.

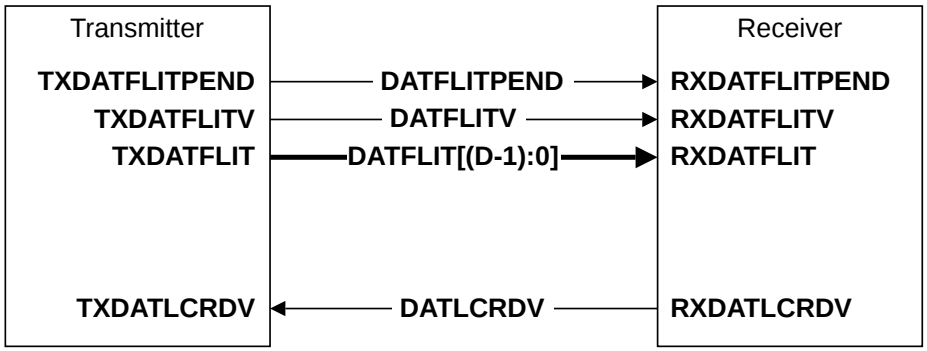

Figure B13.14: DAT channel interface pins

Table B13.5 shows the DAT channel interface signals.

Table B13.5: DAT channel interface signals

Signal	Description
DATFLITPEND	Data Flit Pending. Early indication that a data flit could be transmitted in the following cycle. See B14.4 *Flit level clock gating*.
DATFLITV	Data Flit Valid. The transmitter sets this signal HIGH to indicate when **DATFLIT[(D-1):0]** is valid.
DATFLIT[(D-1):0]	Data Flit. See B13.9.4 *Data flit* for a description of the data flit format.
DATLCRDV	Data L-Credit Valid. The receiver sets this signal HIGH to return a data channel L-Credit to a transmitter. See B14.2.1 *L-Credit flow control*.

B13.9 Flit packet definitions

This section defines the flit format. See:

- B13.9.1 *Request flit*
- B13.9.2 *Response flit*
- B13.9.3 *Snoop flit*
- B13.9.4 *Data flit*

B13.9.1 Request flit

Table B13.6 shows the Request flit format in a REQ channel packet starting at bit 0.

The following key is used:

MBZ Must Be 0

Table B13.6: Request flit format

REQFLIT[(R-1):0] format		
Field	**Field width**	**Comments**
QoS	4	-
TgtID	7 to 11	Width determined by NodeID_Width
SrcID	7 to 11	Width determined by NodeID_Width
TxnID	12	-
ReturnNID	7 to 11	Used for DMT
StashNID		Used in Stash transactions
{(NodeID_Width - 7)'b0,		MBZ
SLCRepHint[6:0]}		Used in cache line replacement algorithms
StashNIDValid	1	Used in Stash transactions
Endian		Used in Atomic transactions
Deep		Used in CleanSharedPersist* transactions
ReturnTxnID[11:0]	12	Used for DMT
{6'b0,		MBZ
StashLPIDValid,		Used in Stash transactions
StashLPID[4:0]}		Used in Stash transactions
Opcode	7	-
Size	3	-
Addr	RAW = 44 to 52	Width determined by Req_Addr_Width (RAW)
NS	1	-
NSE	1	-
LikelyShared	1	-

Continued on next page

Table B13.6 – *Continued from previous page*

REQFLIT[(R-1):0] format		
Field	**Field width**	**Comments**
AllowRetry	1	-
Order	2	-
PCrdType	4	-
MemAttr	4	-
SnpAttr	1	-
DoDWT		Used for DWT
PGroupID[7:0]	8	Used in PCMO transactions
StashGroupID[7:0]		Used in the StashOnceSep transaction
TagGroupID[7:0]		Used for Memory Tagging
{3'b0,		MBZ
LPID[4:0]}		-
Excl	1	Used in Exclusive transactions
SnoopMe		Used in Atomic transactions
CAH		Used in CopyBack Write transactions
ExpCompAck	1	-
TagOp	2	-
TraceTag	1	-
MPAM	M = 0	No MPAM bus
	M = 12	-
PBHA	PB = 0	No PBHA bus
	PB = 4	-
RSVDC	Y = 0	No RSVDC bus
	Y = 4, 8, 12, 16, 24, 32	-
Total	R = (88 + RAW + Y + M + PB) to (100 + RAW + Y + M + PB)	

B13.9.2 Response flit

Table B13.7 shows the Response flit format in a RSP channel packet starting at bit 0.

The following key is used:

MBZ Must Be Zero

Table B13.7: Response flit format

RSPFLIT[(T-1):0] format		
Field	**Field width**	**Comments**
QoS	4	-
TgtID	7 to 11	Width determined by NodeID_Width
SrcID	7 to 11	Width determined by NodeID_Width
TxnID	12	-
Opcode	5	-
RespErr	2	-
Resp	3	-
FwdState[2:0]	3	Used for DCT
DataPull[2:0]		Used in Stash transactions
CBusy	3	-
DBID[11:0]	12	-
{4'b0,		MBZ
PGroupID[7:0]}		Used in Persistent CMO transactions
{4'b0,		MBZ
StashGroupID[7:0]}		Used in Stash transactions
{4'b0,		MBZ
TagGroupID[7:0]}		Used for Memory Tagging
PCrdType	4	-
TagOp	2	-
TraceTag	1	-
Total	T = 65 to 73	

B13.9.3 Snoop flit

Table B13.8 shows the Snoop flit format in a SNP channel packet starting at bit 0.

The following key is used:

MBZ Must Be Zero

Table B13.8: Snoop flit format

SNPFLIT[(S-1):0] format		
Field	**Field width**	**Comments**
QoS	4	-

Continued on next page

Table B13.8 – *Continued from previous page*

SNPFLIT[(S-1):0] format		
Field	**Field width**	**Comments**
SrcID	7 to 11	Width determined by NodeID_Width
TxnID	12	-
FwdNID	7 to 11	Width determined by NodeID_Width
{(NodeID_Width - 4)'0,		
PBHA[3:0]}		
FwdTxnID[11:0]	12	Used for DCT
{6'b0,		MBZ
StashLPIDValid,		Used in Stash transactions
StashLPID[4:0]}		Used in Stash transactions
{4'b0,		MBZ
VMIDExt[7:0]}		Used in DVM transactions
Opcode	5	-
Addr	SAW = 41 to 49	Req_Addr_Width - 3
NS	1	-
NSE	1	-
DoNotGoToSD	1	-
RetToSrc	1	-
TraceTag	1	-
MPAM	M = 0	No MPAM bus
	M = 12	-
Total	S = (52 + SAW + M) to (60 + SAW + M)	

B13.9.4 Data flit

Table B13.9 shows the Data flit format in a DAT channel packet starting at bit 0.

The number of data flits required is dependent on the number of data bytes, and the data bus width. See B2.8.4 *Data packetization*.

The data channel interface supports a 128-bit, 256-bit, and 512-bit data bus width. There are three data flit formats defined, one for each of the three data bus widths supported at the data channel interface.

DataCheck field width is either 0 or equal to the width of the Data field divided by 8.

Poison field width is either 0 or equal to the width of the Data field divided by 64.

The following key is used:

MBZ Must Be Zero

Table B13.9: Data flit fields

DATFLIT[(D-1):0] format		
Field	**Field width**	**Comments**
QoS	4	-
TgtID	7 to 11	Width determined by NodeID_Width
SrcID	7 to 11	Width determined by NodeID_Width
TxnID	12	-
HomeNID {(NodeID_Width - 4)'b0, PBHA[3:0]}	7 to 11	Width determined by NodeID_Width
Opcode	4	-
RespErr	2	-
Resp	3	-
DataSource[4:0] {2'b0, FwdState[2:0]} {2'bX, DataPull[2:0]}	5	Indicates Data source in a response MBZ Used for DCT Can take any value Used in Stash transactions
CBusy	3	-
DBID[11:0]	12	-
CCID	2	-
DataID	2	-
TagOp	2	-
Tag	DW/32 = 4, 8, 16	-
TU	DW/128 = 1, 2, 4	-
TraceTag	1	-
CAH	1	-
RSVDC	Y = 0 Y = 4, 8, 12, 16, 24, 32	No RSVDC bus -
BE	DW/8 = 16, 32, 64	-
Data	DW = 128, 256, 512	DW = B16.1.13 *Data_Width*
DataCheck	0 or DW/8 = 16, 32, 64	DC = DataCheck

Continued on next page

Table B13.9 – *Continued from previous page*

DATFLIT[(D-1):0] format		
Field	**Field width**	**Comments**
Poison	0 or DW/64 = 2, 4, 8	P = Poison
Total	D = (223 to 235) + Y + DC + P	DW = 128-bit Data
	D = (372 to 384) + Y + DC + P	DW = 256-bit Data
	D = (670 to 682) + Y + DC + P	DW = 512-bit Data

B13.10 Protocol flit fields

A Protocol flit is identified by a non-0 value in the opcode field. All the flit fields defined in this section are applicable for a Protocol flit. The following sections describe the encoding of the Protocol flit fields:

B13.10.1 Quality of Service, QoS

This field is used to assign a Quality of Service (QoS) value to a transaction. Ascending values of QoS indicate higher priority levels. See B11.1 *Quality of Service (QoS) mechanism*.

B13.10.2 Target Identifier, TgtID

This field is the node ID of the component to which the message is targeted. This is used by the interconnect to determine the port to which the message is sent. See B2.4.1 *Target Identifier, TgtID, and Source Identifier, SrcID*.

B13.10.3 Source Identifier, SrcID

This field is the node ID of the component from which the message is sent. This is used by the interconnect to determine the port from which the message is sent. See B2.4.1 *Target Identifier, TgtID, and Source Identifier, SrcID*.

B13.10.4 Home Node Identifier, HomeNID

This field is associated with the original request. The Requester uses the value in this field to determine the TgtID of the CompAck to be sent in response to CompData. See B2.4.11 *Home Node Identifier, HomeNID*.

Applicable in CompData and DataSepResp.

Inapplicable and must be 0 in all other Data messages.

B13.10.5 Return Node Identifier, ReturnNID

This field identifies the node to which the Subordinate sends a CompData, DataSepResp, or a Persist response. This field also indicates the node to which the Subordinate sends DBIDResp when DoDWT = 1. The value can be either the node ID of Home or the Requester that originated the transaction. See B2.4.10 *Return Node Identifier, ReturnNID*.

Applicable from Home to Subordinate in ReadNoSnp, ReadNoSnpSep, CleanSharedPersistSep, WriteNoSnp, WriteNoSnpDef, Combined Write, and Atomic requests.

Inapplicable and must be 0 for all other requests. For Stash requests, the same bits in the packet are used for StashNID.

B13.10.6 Forward Node Identifier, FwdNID

This field identifies the Requester to which the CompData response can be forwarded. The value must be the NodeID of the Requester that initiated the transaction. See B2.4.12 *Forward Node Identifier, FwdNID*.

Applicable in Forward-type snoops.

Inapplicable and must be 0 in all other Snoop requests, except range-based TLBI DVM operations.

In range-based TLBI DVM operations, the bits in the field are used for DVM payload.

B13.10.7 Logical Processor Identifier, LPID

This field is used when a single Requester contains more than one logically separate processing agent. The SrcID is used with LPID to uniquely identify the LP that generated the request.

The LPID must be set to the correct value for the following transactions:

- For any Non-snoopable Non-cacheable or Device access:
  - ReadNoSnp
  - WriteNoSnp
  - WriteNoSnpDef
- For Exclusive accesses, that can be one of the following transaction types:
  - ReadClean
  - ReadShared
  - ReadNotSharedDirty
  - ReadPreferUnique
  - MakeReadUnique
  - CleanUnique
  - ReadNoSnp
  - WriteNoSnp

See Chapter B6 *Exclusive accesses* for further details.

In requests, when applicable, the same bits in the packet are used for TagGroupID, PGroupID, and StashGroupID.

For other transactions, the LPID value is permitted, but not required, to indicate the original LP that caused a transaction to be issued.

B13.10.8 Persistence Group Identifier, PGroupID

This field is used by a Requester to process different sets of CleanSharedPersistSep transactions by grouping them together and identifying each using PGroupID.

Applicable in the CleanSharedPersistSep and Write*CleanShPerSep requests and the Persist and CompPersist responses.

Inapplicable and must be set to 0 in all other requests and responses.

In requests, when applicable, the same bits in the packet are used for LPID, TagGroupID, and StashGroupID.

In responses, when applicable, the same bits in the packet are used for DBID, TagGroupID, and StashGroupID.

B13.10.9 Stash Node Identifier, StashNID

This field identifies the target of the Stash request. Provides a valid stash target value when the corresponding StashNIDValid bit is asserted. See B7.4 *Stash target identifiers*.

Applicable in Stash requests.

Inapplicable and must be 0 for all other requests. For ReadNoSnp and ReadNoSnpSep requests, the same bits in the packet are used for ReturnNID.

B13.10.10 Stash Node Identifier Valid, StashNIDValid

This field indicates if StashNIDValid field has a valid value. See B7.4 *Stash target identifiers*.

Applicable in Stash requests, inapplicable and must be set to 0 in all other requests.

Table B13.10 shows the StashNIDValid value encodings.

Table B13.10: StashNIDValid value encodings

StashNIDValid	Description
0	StashNID field value is inapplicable and must be set to 0
1	StashNID field in the Request has a valid Stash target

For permitted combinations of StashNIDValid and StashLPIDValid, see Table B7.3.

B13.10.11 Stash Logical Processor Identifier, StashLPID

This field provides a valid LP target value within the Request Node specified by StashNID. See B7.4 *Stash target identifiers*.

Applicable in Stash requests and Stash type Snoop requests.

Inapplicable and must be 0 for all other requests. For ReadNoSnp requests, the same bits in the packet are used for ReturnTxnID.

Inapplicable and must be 0 for all other snoop requests. For Forwarding snoops, the same bits in the packet are used for FwdTxnID and for SnpDVMOp snoops the same bits in the packet are used for VMIDExt.

B13.10.12 Stash Logical Processor Identifier Valid, StashLPIDValid

This field indicates if the StashLPID field has a valid value. See B7.4 *Stash target identifiers*.

Applicable in Stash requests and Stash snoop requests, inapplicable and must be set to 0 in all other requests.

Table B13.11 shows the StashLPIDValid value encodings.

Table B13.11: StashLPIDValid value encodings

StashLPIDValid	Description
0	StashLPID field value is inapplicable and must be set to 0
1	StashLPID field in the Request has a valid Stash target

For permitted combinations of StashLPIDValid and StashNIDValid, see Table B7.3.

B13.10.13 Stash Group Identifier, StashGroupID

This field is used by a Requester to process different sets of StashOnceSep transactions by grouping them together and identifying each using StashGroupID.

Applicable only in the StashOnceSep request and StashDone response.

Inapplicable and must be set to 0 in all other requests and responses.

In requests, when applicable, the same bits in the packet are used for LPID, TagGroupID, and PGroupID.

In responses, when applicable, the same bits in the packet are used for DBID, TagGroupID, and PGroupID.

B13.10.14 Transaction Identifier, TxnID

This field provides the transaction identifier of the message. When there are multiple outstanding transactions from a given source node they will each use a unique transaction ID. See B2.4.2 *Transaction Identifier, TxnID*.

The TxnID in a link flit must be 0.

B13.10.15 Return Transaction Identifier, ReturnTxnID

This field identifies the value the Subordinate must use in the TxnID field of the CompData, and DataSepResp response. ReturnTxnID can be either the TxnID generated by Home for this transaction or the TxnID in the Request packet from the Requester that originated the transaction. See B2.4.4 *Return Transaction Identifier, ReturnTxnID*.

Applicable only in ReadNoSnp, ReadNoSnpSep, WriteNoSnp, WriteNoSnpDef, Combined Write, and Atomic requests from Home to Subordinate.

Inapplicable and must be set to 0 for all other requests. For Stash requests, the same bits in the packet are used for StashLPID.

B13.10.16 Forwarding Transaction Identifier, FwdTxnID

This field identifies the TxnID field of the original Request associated with the Snoop transaction. See B2.4.5 *Forward Transaction Identifier, FwdTxnID*.

Applicable in Forward-type snoops.

Inapplicable and must be set to 0 in all other Snoop requests. For Stash snoops, the same bits in the packet are used for StashLPID and for SnpDVMOp snoops the same bits in the packet are used for VMIDExt.

B13.10.17 Data Buffer Identifier, DBID

The value in this field in the response packet from a Completer is used in the TxnID field for CompAck or WriteData sent from the Requester.

In Snoop responses with Data Pull, the DBID value indicates the value to be used in the TxnID field of Data Pull response messages. See B2.4.3 *Data Buffer Identifier, DBID*.

In responses, when applicable, the same bits in the packet are used for PGroupID, StashGroupID, and TagGroupID.

B13.10.18 Channel opcodes, Opcode

This field specifies the operation to be carried out. The Opcode encodings are specific to each channel. See:

- B13.10.18.1 *REQ channel opcodes*
- B13.10.18.2 *RSP channel opcodes*
- B13.10.18.3 *SNP channel opcodes*
- B13.10.18.4 *DAT channel opcodes*

B13.10.18.1 REQ channel opcodes

Table B13.12 shows the opcodes for the Request channel.

Table B13.12: REQ channel opcodes

Opcode[5:0]	Request command	
	Opcode[6] = 0	Opcode[6] = 1
0x00	ReqLCrdReturn	Reserved
0x01	ReadShared	MakeReadUnique
0x02	ReadClean	WriteEvictOrEvict
0x03	ReadOnce	WriteUniqueZero
0x04	ReadNoSnp	WriteNoSnpZero
0x05	PCrdReturn	Reserved
0x06	Reserved	Reserved
0x07	ReadUnique	StashOnceSepShared
0x08	CleanShared	StashOnceSepUnique
0x09	CleanInvalid	Reserved
0x0A	MakeInvalid	Reserved
0x0B	CleanUnique	Reserved
0x0C	MakeUnique	ReadPreferUnique
0x0D	Evict	CleanInvalidPoPA[a]
0x0E	Reserved	WriteNoSnpDef[a]
0x0F	Reserved	Reserved
0x10	Reserved	WriteNoSnpFullCleanSh
0x11	ReadNoSnpSep	WriteNoSnpFullCleanInv
0x12	Reserved	WriteNoSnpFullCleanShPerSep
0x13	CleanSharedPersistSep	Reserved
0x14	DVMOp	WriteUniqueFullCleanSh
0x15	WriteEvictFull	Reserved
0x16	Reserved	WriteUniqueFullCleanShPerSep
0x17	WriteCleanFull	Reserved
0x18	WriteUniquePtl	WriteBackFullCleanSh
0x19	WriteUniqueFull	WriteBackFullCleanInv
0x1A	WriteBackPtl	WriteBackFullCleanShPerSep
0x1B	WriteBackFull	Reserved
0x1C	WriteNoSnpPtl	WriteCleanFullCleanSh

Continued on next page

Table B13.12 – *Continued from previous page*

Opcode[5:0]	Request command	
	Opcode[6] = 0	Opcode[6] = 1
0x1D	WriteNoSnpFull	Reserved
0x1E	Reserved	WriteCleanFullCleanShPerSep
0x1F	Reserved	Reserved
0x20	WriteUniqueFullStash	WriteNoSnpPtlCleanSh
0x21	WriteUniquePtlStash	WriteNoSnpPtlCleanInv
0x22	StashOnceShared	WriteNoSnpPtlCleanShPerSep
0x23	StashOnceUnique	Reserved
0x24	ReadOnceCleanInvalid	WriteUniquePtlCleanSh
0x25	ReadOnceMakeInvalid	Reserved
0x26	ReadNotSharedDirty	WriteUniquePtlCleanShPerSep
0x27	CleanSharedPersist	Reserved
0x28 – 0x2F	AtomicStore	Reserved
0x30	AtomicLoad	WriteNoSnpPtlCleanInvPoPA[a]
0x31		WriteNoSnpFullCleanInvPoPA[a]
0x32 – 0x37		Reserved
0x38	AtomicSwap	Reserved
0x39	AtomicCompare	WriteBackFullCleanInvPoPA[a]
0x3A	PrefetchTgt	Reserved
0x3B – 0x3F	Reserved	Reserved

[a] This request was not supported prior to CHI Issue F.

Table B13.13 shows the sub-opcodes for AtomicStore and AtomicLoad.

Table B13.13: Sub-opcodes for AtomicStore and AtomicLoad

Opcode[5:3]		Opcode[2:0]	Operation
AtomicStore	AtomicLoad		
101	110	000	ADD
		001	CLR
		010	EOR
		011	SET
		100	SMAX

Continued on next page

Table B13.13 – *Continued from previous page*

Opcode[5:3]		Opcode[2:0]	Operation
AtomicStore	**AtomicLoad**		
		101	SMIN
		110	UMAX
		111	UMIN

B13.10.18.2 RSP channel opcodes

Table B13.14 shows the opcodes for the Response channel.

Table B13.14: RSP channel opcodes

Opcode[4:0]	Response
0x0	RespLCrdReturn
0x1	SnpResp
0x2	CompAck
0x3	RetryAck
0x4	Comp
0x5	CompDBIDResp
0x6	DBIDResp
0x7	PCrdGrant
0x8	ReadReceipt
0x9	SnpRespFwded
0xA	TagMatch
0xB	RespSepData
0xC	Persist
0xD	CompPersist
0xE	DBIDRespOrd
0xF	Reserved
0x10	StashDone
0x11	CompStashDone
0x12 – 0x13	Reserved
0x14	CompCMO
0x15 – 0x1F	Reserved

B13.10.18.3 SNP channel opcodes

Table B13.15 shows the opcodes for the Snoop channel.

Table B13.15: SNP channel opcodes

Opcode[4:0]	Snoop command
0x00	SnpLCrdReturn
0x01	SnpShared
0x02	SnpClean
0x03	SnpOnce
0x04	SnpNotSharedDirty
0x05	SnpUniqueStash
0x06	SnpMakeInvalidStash
0x07	SnpUnique
0x08	SnpCleanShared
0x09	SnpCleanInvalid
0x0A	SnpMakeInvalid
0x0B	SnpStashUnique
0x0C	SnpStashShared
0x0D	SnpDVMOp
0x0E – 0x0F	Reserved
0x10	SnpQuery
0x11	SnpSharedFwd
0x12	SnpCleanFwd
0x13	SnpOnceFwd
0x14	SnpNotSharedDirtyFwd
0x15	SnpPreferUnique
0x16	SnpPreferUniqueFwd
0x17	SnpUniqueFwd
0x18 – 0x1F	Reserved

B13.10.18.4 DAT channel opcodes

Table B13.16 shows the opcodes for the Data channel.

Table B13.16: DAT channel opcodes

Opcode[3:0]	Data command
0x0	DataLCrdReturn
0x1	SnpRespData
0x2	CopyBackWriteData
0x3	NonCopyBackWriteData
0x4	CompData
0x5	SnpRespDataPtl
0x6	SnpRespDataFwded
0x7	WriteDataCancel
0x8 – 0xA	Reserved
0xB	DataSepResp
0xC	NCBWrDataCompAck
0xD – 0xF	Reserved

B13.10.19 Deep persistence, Deep

This field is used by the Requester to indicate that writes must have been written to the final destination before the Completer can provide certain responses.

Applicable in the CleanSharedPersist* request and Combined Write request with CleanSharedPersistSep.

Inapplicable and must be set to 0 in all other requests.

When Deep is deasserted:

- All earlier writes must have reached the PoP before thc Completer can send:
  - Comp for a CleanSharedPersist transaction
  - Persist or CompPersist for a CleanSharedPersistSep transaction
- The PoP is the point at which it is guaranteed that sufficient time is available to make the data persistent after loss of power.

When Deep is asserted:

- All earlier writes must have been written to the final destination, not just the PoP, before the Completer can send:
  - Comp for a CleanSharedPersist transaction
  - Persist or CompPersist for a CleanSharedPersistSep transaction
- The final destination is the point at which no time is required to make the data persistent after the loss of power, thus preserving data even when battery failure occurs.

See Table B2.7 for more information on response ordering guarantees.

B13.10.20 Address, Addr

This field specifies the address associated with the message.

The CHI protocol supports a PA of 44 to 52 bits. The address is carried in Addr field in the REQ and SNP channel.

The width of the field is defined by the Req_Addr_Width parameter. The field width is the same value of the parameter in REQ channel, Addr[(43-51):0], and is three less than the parameter value in the SNP channel, Addr[(43-51):3].

The Addr field is utilized in the REQ and SNP channels in the following manner.

- For Read, PrefetchTgt, Dataless, Write, and Atomic transactions, the Addr field includes the address of the memory location being accessed. This starts with Addr[0] mapped to bit 0 of Addr.
- For a snoop request, except SnpDVMOp, the Addr field includes the address of the location being snooped. This starts with Addr[3] mapped to bit 0 of Addr.
  - Addr[(43-51):6] is the cache line address. It is sufficient to uniquely identify the cache line to be accessed by the snoop.
  - Addr[5:4] identifies the critical chunk being accessed by the transaction. See B2.8.6 *Critical Chunk Identifier*. It is recommended that the snooped cache returns the data in wrap order with the critical chunk returned first.

  Note

  Addr[3] in the REQ channel, that is Addr[0] in SNP channel, is supplied but is not used by a snoop request.
- For a DVMOp and SnpDVMOp request, the Addr field is used to carry information related to a DVM operation. See Chapter B8 *DVM Operations*.
- The Addr value is not used for PCrdReturn transaction and must be set to 0.

B13.10.21 Non-secure, NS

This field is combined with NSE to establish the PAS of an access. See B2.7.2 *Physical Address Space, PAS*.

B13.10.22 Non-secure extension, NSE

This field is combined with NS to establish the PAS of an access. See B2.7.2 *Physical Address Space, PAS*.

B13.10.23 Size of transaction data, Size

This field specifies the size of the data associated with the transaction. See B2.8.1 *Data size*.

Table B13.17 shows the Size field value encodings.

Table B13.17: Size field value encodings

Size[2:0]	Bytes
0b000	1
0b001	2
0b010	4
0b011	8

Continued on next page

Table B13.17 – *Continued from previous page*

Size[2:0]	Bytes
0b100	16
0b101	32
0b110	64
0b111	Reserved

B13.10.24 Memory Attribute, MemAttr

This field is associated with the transaction.

Table B13.18 shows the MemAttr value encodings.

Table B13.18: MemAttr value encodings

MemAttr[3:0]	Description	0	1
[3]	Allocate hint bit. Indicates whether or not the cache receiving the transaction is recommended to allocate the transaction	Recommended that the cache line is not allocated	Recommended that the cache line is allocated
[2]	Cacheable bit. Indicates a Cacheable transaction for which the cache, when present, must be looked up in servicing the transaction	Non-cacheable. Looking up a cache is not required	Cacheable. Looking up a cache is required
[1]	Device bit. Indicates if the memory type associated with the transaction is Device or Normal	Normal memory type	Device memory type
[0]	Early Write Acknowledge (EWA) bit. Specifies the EWA status for the transaction	EWA not permitted	EWA permitted

See B2.7.3 *Memory Attributes*.

B13.10.25 Snoop Attribute, SnpAttr

This field specifies the snoop attribute associated with the transaction.

Table B13.19 shows the SnpAttr value encodings.

Table B13.19: SnpAttr value encodings

SnpAttr	Description
0	Non-snoopable An HN-F receiving this request is not expected, but is permitted, to snoop any RN-F nodes.

Continued on next page

Table B13.19 – *Continued from previous page*

SnpAttr	Description
1	Snoopable An HN-F receiving this request might need to send appropriate snoops to any RN-F nodes that could have the cache line. This could implicate all, certain, or no RN-F nodes, if the HN-F does not have a snoop filter. Snoops are not required as a result of CopyBack transactions. The response given by an HN-I receiving this request is detailed in B16.1.17 *Nonshareable_Cache_Maint*.

See B2.7.6 *Snoop attribute*.

B13.10.26 Do Direct Write Transfer, DoDWT

The characteristics of this field are:

- Only applicable in WriteNoSnpPtl, WriteNoSnpFull, WriteNoSnpDef, and Combined Write requests from Home to Subordinate.
- Inapplicable and must be set to 0 in all other requests.
- The bit shares the same field as SnpAttr.

Table B13.20 shows the DoDWT value encodings.

Table B13.20: DoDWT value encodings

DoDWT	DBIDResp.TgtID value	DBIDResp.TxnID value
0	Set to the SrcID value of the request	Set to TxnID value in the request
1	Set to the ReturnNID value in the request	Set to ReturnTxnID value in the request

See B2.3.2.1 *Immediate Write* and B2.3.2.4 *Combined Immediate Write and CMO*.

B13.10.27 Likely Shared, LikelyShared

This field indicates whether the requested data is likely to be shared with another Request Node. See B2.7.5 *Likely Shared*.

Table B13.21 shows the LikelyShared field value encodings.

Table B13.21: LikelyShared value encodings

LikelyShared	Description
0	Not likely to be shared by another Request Node

Continued on next page

Table B13.21 – *Continued from previous page*

LikelyShared	Description
1	Likely to be shared by another Request Node

B13.10.28 Ordering requirements, Order

This field specifies the ordering requirements for a transaction. See B2.6 *Ordering* for more information on the ordering requirements.

Table B2.9 shows the Order field value encodings.

Table B13.22: Order value encodings

Order[1:0]	Description	Permitted between
0b00	No ordering required	All
0b01	Request accepted	HN-F to SN-F, and HN-I to SN-I
	Reserved	RN to HN
0b10	Request Order/OWO	RN to HN[a]
	Request Order	HN-I to SN-I
	Reserved	HN-F to SN-F
0b11	Endpoint Order	RN to HN, and HN-I to SN-I
	Reserved	HN-F to SN-F

[a] Request Order when ExpCompAck = 0. OWO when ExpCompAck = 1.

B13.10.29 Exclusive, Excl

This field indicates that the corresponding transaction is an Exclusive-type transaction. The Exclusive bit must only be used with the following transactions:

- ReadNotSharedDirty
- ReadShared
- ReadClean
- ReadPreferUnique
- CleanUnique
- MakeReadUnique
- ReadNoSnp
- WriteNoSnp

Table B13.23 shows the Excl value encodings.

Table B13.23: Excl value encodings

Excl	Description
0	Normal transaction
1	Exclusive transaction

See B6.3 *Exclusive transactions*.

B13.10.30 CopyAtHome, CAH

This field indicates:

- In responses from a Home or a Snoopee: whether the Home has a copy of the cache line that is provided to the Requester.
- In CopyBack requests to Home: the Requester has not modified the line or MTE tags since Home indicated keeping a copy of the line.

The CAH attribute does not affect the normal operation of the Requester regarding Clean or Dirty lines. CAH is only used to influence whether a data transfer is required when executing a CopyBack transaction.

Applicable and can take any value in:

- All CopyBack and Combined CopyBack Write requests, except WriteBackPtl
- CompData and DataSepResp responses to the Requester
- SnpRespData and SnpRespDataFwded

Applicable and must be set to 0 in:

- WriteBackPtl
- SnpRespDataPtl

Inapplicable and must be set to 0 in all other messages.

See B2.3.2.3 *CopyBack Write* and B2.7.8 *CopyAtHome attribute* for more information.

B13.10.31 Page-based Hardware Attribute, PBHA

This field carries 4 bits from the translation tables that can be used for IMPLEMENTATION DEFINED hardware control. See B11.5 *Page-based Hardware Attributes*.

Support for PBHA on an interface is defined by PBHA_Support property. See B16.1.18 *PBHA_Support*.

When supported, the PBHA field is present on REQ, DAT, and SNP channels:

- On the REQ channel, PBHA is applicable in all requests, except for DVMOp and PCrdReturn where it is inapplicable and must be set to 0.
- On the DAT channel, PBHA is only applicable in SnpRespData, SnpRespDataPtl, and SnpRespDataFwded. The PBHA field is inapplicable and must be set to 0 in all other DAT messages.
- On the SNP channel, PBHA is only applicable in Stash snoops. The PBHA field is inapplicable and must be set to 0 in all Non-stash snoops.

The PBHA field is not present in RSP channel.

B13.10.32 Endian

This field indicates the endianness of Data in an Atomic transaction. See B2.8.5.3 *Endianness*.

Applicable in Atomic requests, inapplicable in all other requests and must be set to 0.

Table B13.24 shows the Endian value encodings.

Table B13.24: Endian value encodings

Endian	Description
0	Little Endian
1	Big Endian

B13.10.33 Allow Retry, AllowRetry

This field specifies that the request is being sent without a P-Credit and that the target can determine if a retry response is given. See B2.9.2 *Transaction Retry mechanism*.

Table B13.25 shows the AllowRetry value encodings.

Table B13.25: AllowRetry value encodings

AllowRetry	Description
0	RetryAck response not permitted
1	RetryAck response permitted

B13.10.34 Expect Completion Acknowledge, ExpCompAck

This field indicates that the transaction includes a CompAck response.

Table B13.26 shows the ExpCompAck value encodings.

Table B13.26: ExpCompAck value encodings

ExpCom-pAck	Description
0	Transaction does not include a CompAck response
1	Transaction includes a CompAck response

For CopyBack Write transactions, when ExpCompAck cannot be used in isolation to determine if a CompAck is included in the transaction. The transaction flow selected by Home determines if a CompAck is required. See B2.3.2.3 *CopyBack Write* for more information.

B13.10.35 SnoopMe

This field indicates that the Home must determine whether to send a snoop to the Requester. See B2.3.3 *Atomic transactions*.

Only applicable in Atomic requests.

Table B13.27 shows the SnoopMe value encodings.

Table B13.27: SnoopMe value encodings

SnoopMe	Description
0	Home is permitted, but not required, to send a snoop to the Requester.
1	Home must send a Snoop to the Requester if the cache line could be present at the Requester.

B13.10.36 Return to Source, RetToSrc

This field requests the Snoopee to return a copy of the cache line to the Home.

RetToSrc is inapplicable and must be set to 0 in:

- SnpCleanShared, SnpCleanInvalid, and SnpMakeInvalid
- SnpOnceFwd and SnpUniqueFwd
- SnpUniqueStash, SnpMakeInvalidStash, SnpStashUnique, and SnpStashShared
- SnpQuery

RetToSrc is applicable and can take any value in all other snoops except SnpDVMOp.

RetToSrc is inapplicable and must be set to 0 in SnpDVMOp.

For RetToSrc bit semantics, see B4.9 *Returning Data with Snoop response*.

B13.10.37 Data Pull, DataPull

This field indicates the inclusion of a Read request, also referred to as a Data Pull, in the Snoop response. See B7.1.1 *Snoop requests and Data Pull*.

Applicable in SnpResp, SnpRespData, and SnpRespDataPtl response to a Stash request, not applicable in all other Snoop responses.

When the DataPull bit is set in a SnpRespData message, it must be set in all packets of that response message.

Table B13.28 shows the DataPull field value encodings.

Table B13.28: DataPull value encodings

DataPull	Description	Comment
0b000	No read	Inclusion of Data Pull in the Snoop response
0b001	Read	
0b010 – 0b111	-	Reserved

B13.10.38 Do not transition to SD state, DoNotGoToSD

This field is an attribute in a Snoop request indicating if a Snoopee is required to not transition to SD state. See B4.10 *Do not transition to SD*.

Applicable, and can take any value in:

- SnpOnce, SnpOnceFwd
- SnpClean, SnpCleanFwd
- SnpNotSharedDirty, SnpNotSharedDirtyFwd
- SnpShared, SnpSharedFwd
- SnpPreferUnique, SnpPreferUniqueFwd

Applicable, and must be set to 1 in:

- SnpStashShared, SnpStashUnique
- SnpUnique, SnpUniqueFwd. SnpUniqueStash
- SnpCleanShared
- SnpCleanInvalid
- SnpMakeInvalid, SnpMakeInvalidStash

Inapplicable, and must be set to 0 in:

- SnpQuery
- SnpDVMOp

Table B13.29 shows the DoNotGoToSD value encodings.

Table B13.29: DoNotGoToSD value encodings

DoNotGoToSD	Description
0	Permitted to transition to SD state.
1	Transitioning to SD state is not permitted. If already in SD state, must exit SD state in response to the snoop, except for SnpStash*. The bit value can be ignored by the Snoopee if the Snoop request is SnpOnce or SnpOnceFwd.

B13.10.39 Protocol Credit Type, PCrdType

This field indicates the type of credit being granted or returned. See Table B13.30.

Table B13.30 shows the PCrdType value encodings.

Table B13.30: PCrdType value encodings

PCrdType	Description
`0x0 - 0xF`	P-Credit type 0 to 15 respectively

B13.10.40 Tag Operation, TagOp

This field indicates the operation to be performed on the tags present in the corresponding DAT channel.

Table B13.31 shows the TagOp value encodings.

Table B13.31: TagOp value encodings

TagOp[1:0]	Tag Operation	Description
0b00	*Invalid*	The tags are not valid
0b01	*Transfer*	– The tags are Clean. – Tag Match does not need to be performed. – All tags corresponding to data in the packet must be transferred. For Snoopable transactions, partial tag transfer is not supported. – TU field is not applicable and must be set to 0.
0b10	*Update*	– The Allocation Tag values have been updated and are Dirty. – The Tags in memory should be updated. Only the Tags that have TU asserted must be updated.
0b11	*Match*	– The Physical Tags in the write must be checked against the Allocation Tag values obtained from memory. – The Match Tag operation must be enabled for only those tags that have BE asserted. – TU field is not applicable and must be set to 0.
	Fetch	– Tags must be fetched. – All tags must be fetched. – Permitted, but not required, to fetch valid data.

TagOp value in a Retry request must be the same as in the original request.

B13.10.41 Tag

Tag[4*n-1:0]. This field provides n sets of 4-bit tags, each associated with a 16-byte, aligned address location.

[Tag[((4*n)-1) : 4*(n-1)] corresponds to Data[((128*n)-1 : 128*(n-1)]

B13.10.42 Tag Update, TU

TU[n-1:0]. This field indicates which of the Allocation Tags must be updated. There is one TU bit for each tag. Only valid in Snoop responses and Write transactions which update the Allocation Tags. Must be set to 0 in all other transactions.

TU[n-1] corresponds to [Tag[(4*n)-1 : 4*(n-1)]

B13.10.43 Tag Group Identifier, TagGroupID

This field is used by a Requester to process different sets of TagMatch responses by grouping the corresponding requests together and identifying each group using TagGroupID. The precise contents of TagGroupID are IMPLEMENTATION DEFINED. Typically, TagGroupID is expected to contain an Exception Level, TTBR value, and CPU identifier.

TagGroupID is applicable in requests where TagOp is set to *Match* and in a TagMatch response.

In requests, when applicable, the same bits in the packet are used for LPID, PGroupID, or StashGroupID.

In responses, when applicable, the same bits in the packet are used for DBID, PGroupID, or StashGroupID.

B13.10.44 Trace Tag, TraceTag

This field is a bit in a packet used to tag the packets associated with a transaction for tracing purposes.

Table B13.32 shows the TraceTag field value encodings.

Table B13.32: TraceTag value encodings

TraceTag	Description
0	Packet is not tagged
1	Packet is tagged

See Chapter B11 *System Control, Debug, Trace, and Monitoring*.

B13.10.45 Memory System Resource Partitioning and Monitoring, MPAM

This field is used to efficiently utilize the memory resources among users and to monitor their use. See B11.4 *MPAM*.

B13.10.46 Virtual Machine Identifier Extension, VMIDExt

This field is used to extend VMID value from 8 bits to 16 bits. See B8.4.1 *DVM message payload*.

B13.10.47 Response Error, RespErr

This field indicates the error status of the response. See Chapter B9 *Error Handling*.

Table B13.33 shows the RespErr value encodings.

Table B13.33: RespErr value encodings

RespErr[1:0]	Description
0b00	Normal Okay. Indicates that either: – The Normal access was successful. For WriteNoSnpDef, RespErr must be used in combination with Resp to establish the exact response. See Table B13.35. – The Exclusive access failed.
0b01	Exclusive Okay. Indicates that either the read or write portion of an Exclusive access was successful.
0b10	Data Error, DERR
0b11	Non-data Error, NDERR

B13.10.48 Response status, Resp

This field must have the same value in all data flits of a multi-flit data transfer.

Table B13.34 shows the Resp value encodings.

Table B13.34: Resp value encodings

Resp[2:0]	Description
Resp[2]	PassDirty. Indicates that the data included in the response message is Dirty with respect to memory and that the responsibility of writing back the cache line is being passed to the recipient of the response message. **0** Returned data is not Dirty. **1** Returned data is Dirty and the responsibility of writing back the cache line is being passed on.
Resp[1:0]	For snoop responses, this field indicates the final state of the snooped RN-F. For completion responses, this field indicates the final state in the Request Node. For WriteData responses, this field indicates the state of the data in the Request Node when the data is sent. For TagMatch responses, Resp[0] alone indicates a Tag Match pass or fail.

Table B13.35 shows the valid Resp value encodings.

Table B13.35: Valid Resp value encodings for different message types

Response type	Resp[2:0]	State	Notes
Snoop responses	0b000	I	Final state of the snooped RN-F
	0b001	SC	
	0b010	UC, UD	

Continued on next page

Table B13.35 – *Continued from previous page*

Response type	Resp[2:0]	State	Notes
	0b011	SD	
	0b100	I_PD	Final state of the snooped RN-F. Responsibility for updating memory is passed to Home.
	0b101	SC_PD	
	0b110	UC_PD	
	0b111	-	Reserved
Comp responses (Not including WriteNoSnpDef transactions)	0b000	I	Final state of the requesting RN-F
	0b001	SC	
	0b010	UC	
	0b011	-	
	0b100	-	Reserved
	0b101	-	
	0b110	UD_PD	
	0b111	SD_PD	Final state of the requesting RN-F. Responsibility for updating memory is passed to the Requester.
Comp or CompDBIDResp responses for WriteNoSnpDef only	0b000	Successful	Only when RespErr[1:0] = 0b00
	0b001	Unsupported	Reserved and must be 0b000 when RespErr[1:0] != 0b00
	0b010	Defer	See Table B9.1 for more information
	0b011 – 0b111	-	Reserved
WriteData responses	0b000	I	Cache state in the response is imprecise and must be ignored
	0b001	SC	State of the cache line at the RN-F when data is sent
	0b010	UC	
	0b011	-	Reserved
	0b100	-	
	0b101	-	

Continued on next page

Table B13.35 – *Continued from previous page*

Response type	Resp[2:0]	State	Notes
	0b110	UD_PD	State of the cache line at the RN-F when data is sent. Responsibility for updating memory is passed to the Home.
	0b111	SD_PD	
CompAck responses for CopyBack transactions	0b000	I	Cache state in the response is imprecise and must be ignored
	0b001	SC	State of the cache line at the RN-F when the response is sent
	0b010	UC	
	0b011	-	Reserved
	0b100	-	
	0b101	-	
	0b110	UD_PD	State of the cache line at the RN-F when data is sent. Responsibility for updating memory is passed to the Home.
	0b111	SD_PD	
TagMatch responses	0b000	Fail	Part of TagMatch operation
	0b001	Pass	
	0b010 - 0b111	-	Reserved

B13.10.49 Forward State, FwdState

This field indicates the state in the CompData sent from the Snoopee to the Requester. Applicable in SnpRespFwded and SnpRespDataFwded, inapplicable in all other Snoop responses and must be set to 0.

Table B13.36 shows the FwdState value encodings.

Table B13.36: FwdState value encodings

FwdState	Description	
FwdState[2]	Pass Dirty.	
	0	Forwarded data is not Dirty.
	1	Forwarded data is Dirty and the responsibility of writing back the cache line is passed on to the Requester.

Continued on next page

Table B13.36 – *Continued from previous page*

FwdState	Description
FwdState[1:0]	Indicates the final state at the Requester. See Table B13.37

Table B13.37 enumerates the FwdState value encodings.

Table B13.37: Valid FwdState value encodings

FwdState[2:0]	State	Comment
0b000	I	Final state at the Requester
0b001	SC	
0b010	UC	
0b011	-	Reserved
0b100	-	
0b101	-	
0b110	UD_PD	Final state at the Requester.
0b111	SD_PD	Responsibility for updating memory is passed to the Requester

B13.10.50 Completer Busy, CBusy

This field is a mechanism for the Completer of a transaction to indicate its current level of activity. The CBusy value encodings are IMPLEMENTATION DEFINED. See B11.6 *Completer Busy*.

B13.10.51 Data payload, Data

This field is the data payload that is being transported in a Data packet.

The following data bus widths are supported:

- 128-bit
- 256-bit
- 512-bit

See B2.8.4 *Data packetization*.

B13.10.52 Critical Chunk Identifier, CCID

This field indicates the critical 128-bit chunk of the data that is being requested. See B2.8.6 *Critical Chunk Identifier*.

Table B13.38 shows the CCID value encodings.

Table B13.38: CCID value encodings

CCID[1:0]	Critical data chunk
0b00	Data[127:0]
0b01	Data [255:128]
0b10	Data [383:256]
0b11	Data [511:384]

B13.10.53 Data Identifier, DataID

This field indicates the relative position of the data chunk within the 512-bit cache line that is being transferred. See B2.8.4 *Data packetization*.

Table B13.39 shows the DataID value encodings.

Table B13.39: DataID value encodings

DataID	Data width		
	128-bit	256-bit	512-bit
0b00	Data[127:0]	Data[255:0]	Data[511:0]
0b01	Data [255:128]	Reserved	Reserved
0b10	Data [383:256]	Data[511:256]	Reserved
0b11	Data [511:384]	Reserved	Reserved

B13.10.54 Byte Enable, BE

This field indicates if the byte of data corresponding to this BE bit is valid. The BE field is defined for write data, DVM payload, and Snoop response data transfers. For Read response data transfers, this field is inapplicable and can take any value. The BE field consists of a bit for each data byte in the DAT flit. See B2.8.3 *Byte Enables*.

Table B13.40 shows the BE value encodings.

Table B13.40: BE value encodings

BE	Byte enable
0	Corresponding byte of data is not valid
1	Corresponding byte of data is valid

B13.10.55 Data check, DataCheck

This field is used to supply the DataCheck bit for the corresponding byte of Data. See B9.2.2 *Data Check*.

B13.10.56 Poison

This field indicates if the 64-bit chunk of data corresponding to a Posion bit is poisoned, that is, has an error, and must not be consumed. See B9.2.1 *Poison*.

Table B13.41 shows the Poison value encodings.

Table B13.41: Poison value encodings

Poison	Description
0	Corresponding 64-bit chunk is not poisoned
1	Corresponding 64-bit chunk is poisoned

B13.10.57 Data source, DataSource

This field identifies the Sender of the data response. See B11.2.1 *DataSource value assignment*.

Applicable in CompData and DataSepResp responses in Read and Atomic transactions, and SnpRespData and SnpRespDataPtl responses in Non-stash type Snoop transactions. DataSource is not applicable, and must be set to 0, in all other responses.

Table B13.42 shows the fixed values for DataSource when Data comes from memory. In Table B13.42, the following applies:

R = 0 The memory is local

R = 1 The memory is remote.

Table B13.42: Fixed values for DataSource

DataSource	Description	Comments
0bR0000	DataSource is not supported, or Completer is not sending useful information	Applicable only if the Completer is a non-memory component
0bR0001 – 0bR0101	IMPLEMENTATION DEFINED	See B11.2.2.1 *Suggested DataSource values*
0bR0110	PrefetchTgt memory prefetch was useful	Read data was obtained from Completer with lower latency as the PrefetchTgt request already read or initiated a read of data from memory

Continued on next page

Table B13.42 – *Continued from previous page*

DataSource	Description	Comments
0bR0111	PrefetchTgt memory prefetch was not useful	Read request went through a complete memory access. The precise reason for signaling that a prefetch was not useful is IMPLEMENTATION DEFINED. **Note** A non-exhaustive list of examples of what this could signal is: – There is no latency reduction due to the PrefetchTgt request sent earlier. – There was no PrefetchTgt request sent earlier. – The Completer does not support the reporting of the effectiveness of a PrefetchTgt request.
0bR1000 – 0bR1111	IMPLEMENTATION DEFINED	See B11.2.2.1 *Suggested DataSource values*

B13.10.58 System Level Caches Replacement Hint, SLCRepHint

This field forwards cache replacement hints from the Requesters to the SLC in the interconnect. See B11.3 *SLC Replacement Hint*.

B13.10.59 Reserved for Customer Use, RSVDC

This field in a Protocol flit can take any value. Propagation of this field through the interconnect is IMPLEMENTATION DEFINED. There is no defined relationship between RSDVC field values in different packets of a transaction.

This field is applicable in the REQ and DAT channels as follows:

- The presence of this field is optional.
- The permitted field widths are 4-bit, 8-bit, 12-bit, 16-bit, 24-bit, and 32-bit.
- The field widths:
  - Can be different between REQ and DAT channels.
  - Need not be the same across all REQ channels in the system.
  - Need not be the same across all DAT channels in the system.

When connecting Tx and Rx flit interfaces that have mismatched RSVDC widths:

- The corresponding lower-order bits of the RSVDC field must be connected at each side of the interface.
- The higher-order RSVDC bits at the RX interface that do not have corresponding bits at the TX interface must be tied LOW.

B13.11 Link flit

A link flit is used to return L-Credits to the receiver during a link deactivation sequence. Link flits originate at a link Transmitter and terminate at the link Receiver on the other side of the link.

A link flit is identified by a zero value in the Opcode field. The TxnID field of the link flit is required to be 0. The remaining fields are not used and can take any value. See B13.10.18 *Channel opcodes, Opcode* for the link flit type encoding.

Chapter B14
Link Handshake

This chapter describes the link handshake requirements. It contains the following sections:

B14.1 Clock and initialization

This section specifies the CHI requirement for global clock and reset signals.

B14.1.1 Clock

This specification does not define a specific clocking microarchitecture. It is expected that all devices, interconnects, and so on, includes one or more clocks that can be relied upon by other Link layer functions that require synchronous communication. A generic clock signal is referred to as CLK in the following sections, where applicable.

B14.1.2 Reset

This specification does not define a specific reset microarchitecture. It is expected that all devices, interconnects, and so on, include a specific reset deassertion event that can be relied upon by other Link layer functions. A generic reset signal is referred to as **RESETn** in the following sections, where applicable.

B14.1.3 Initialization

During reset the following interface signals must be deasserted by the component:

- **TX***LCRDV
- **TX***FLITV
- **TXLINKACTIVEREQ** and **RXLINKACTIVEACK**

The earliest point after reset that the component is permitted to begin driving these signals HIGH is at a rising CLK edge after RESETn is HIGH.

All other signals can take any value.

B14.2 Link layer Credit

This section describes the Link layer Credit (L-Credit) mechanism. Information is transferred across an interface channel by the use of L-Credits. To transfer one flit from the Transmitter to the Receiver, the Transmitter must have obtained an L-Credit.

B14.2.1 L-Credit flow control

An L-Credit is sent from the Receiver to the Transmitter by asserting the appropriate **LCRDV** signal for a single clock cycle. There is one **LCRDV** signal for each channel. See B13.8 *Channel interface signals* for the **LCRDV** signal naming for each channel.

Each transfer of a flit from the Transmitter to the Receiver consumes one L-Credit.

The minimum number of L-Credits that a Receiver can provide is 1.

The maximum number of L-Credits thata Receiver can provide is 15.

A Receiver must guarantee that it can accept all the flits for which it has issued L-Credits.

When the link is active, the Receiver must provide L-Credits in a timely manner without requiring any action on the part of the Transmitter.

Note

An L-Credit cannot be used in the same cycle as being received.

B14.3 Low power signaling

This section describes the signaling used to enhance the low power operation of the interface. There are several different levels of operation:

Flit Level Clock Gating

This technique is used to provide a cycle-by-cycle indication of the activity of each of the channels of the interface. For each channel, an additional signal is provided to indicate if a transfer could occur in the following cycle. This signaling permits local clock gating of certain registers associated with the interface.

Link Activation

Link activation and deactivation are supported to permit the interface to be taken to a safe state, so that both-sides of the interface can enter a low power state that permits them to be either clock-gated or power-gated.

Protocol Activity Indication

The Protocol layer activity indication is used by components to indicate if there are ongoing transactions in progress. The Protocol layer activity indication can be used to influence the decision to use other low power techniques.

B14.4 Flit level clock gating

The **FLITPEND** signal associated with a channel is used to indicate if a valid flit is going to be sent in the next clock cycle. There is one **FLITPEND** signal for each channel. See B13.8 *Channel interface signals* for the **FLITPEND** signal naming for each channel.

The requirements for the use of **FLITPEND** are:

- It is required that the signal is asserted exactly one cycle before a flit is sent from the Transmitter.
- When asserted, it is permitted, but not required, that the Transmitter sends a flit in the next cycle.
- When deasserted, it is required that the Transmitter does not send a flit in the next cycle.
- A Transmitter is permitted, but not required, to keep the signal permanently asserted. For example, if a Transmitter cannot determine in advance when a flit is to be sent.
- A Transmitter is permitted, but not required, to assert this signal without possessing an L-Credit.
- A Transmitter is permitted, but not required, to assert and subsequently deassert this signal without sending a flit.

Figure B14.1 shows an example of the use of the **FLITPEND** signal.

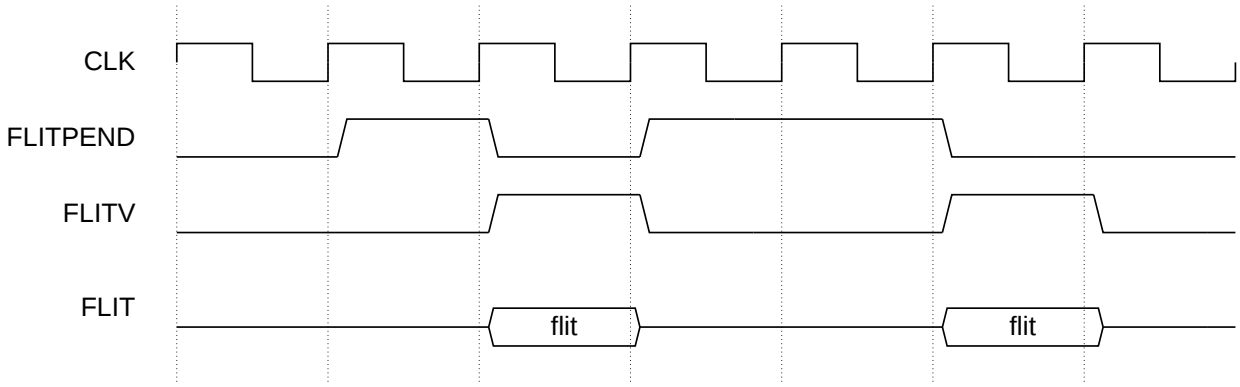

Figure B14.1: FLITPEND indicating a valid flit in next cycle

B14.5 Interface activation and deactivation

A mechanism is provided for an entire interface to move between a full running operational state and a low power state. It is important to exchange L-Credits when moving between operational states, including when exiting from reset. The exchange of Link flits is carefully controlled to avoid the loss of flits or credits.

On exit from reset, or when moving to a full running operational state, the interface starts in an idle state and the transfer of flits can only commence when L-Credits have been exchanged. L-Credits can only be exchanged when the Sender of the credits knows the Receiver is ready to receive them.

A two-signal, four-phase, handshake mechanism is used. This two signal interface is used for all channels traveling in the same direction, rather than being required for each individual channel. An entire interface uses a total of four signals, two signals are used for all the transmit channels and two signals are used for all the receive channels.

B14.5.1 Request and Acknowledge handshake

For the purposes of description, the two signal Request and Acknowledge signaling is described using the signal names **LINKACTIVEREQ** and **LINKACTIVEACK**.

This section describes the operation of the **LINKACTIVEREQ** and **LINKACTIVEACK** handshake pairs for all channels moving in one direction. B14.6 *Transmit and receive link interaction* describes the interaction between the handshake pairs for the transmit channels and those for the receive channels.

For a single channel, or group of channels traveling in the same direction, Figure B14.2 shows the relationship between the Payload, Credit, **LINKACTIVEREQ**, and **LINKACTIVEACK** signals.

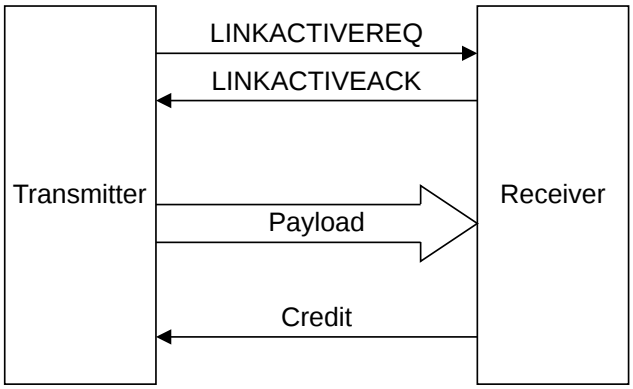

Figure B14.2: Relationship between Payload, Credit, and LINKACTIVE signals

As Figure B14.2 shows, the Transmitter, which sends the payload flits, requires a credit before a flit is sent. A credit is passed from the Receiver when resources are available to accept a flit:

- On exit from reset, credits are held by the Receiver and must be passed to the Transmitter before flit transfer can begin.
- During normal operation, there is an ongoing exchange of flits and credits between the two sides of the interface.
- Before entering a low power state, the sending of payload flits must be stopped and all credits must be returned to the Receiver. This effectively returns the interface to the same state previously immediately after reset.

Four states are defined for the interface operation:

RUN There is an ongoing exchange of flits and credits between the two components.

STOP The interface is in a low power state and is not operational. All credits are held by the Receiver and the Transmitter is not permitted to send any flits.

ACTIVATE This state is used when moving from the STOP state to the RUN state.

DEACTIVATE This state is used when moving from the RUN state to the STOP state.

RUN and STOP are stable states. When one of these states is entered, a channel can remain in this state for an indefinite period.

DEACTIVATE and ACTIVATE are transient states. It is expected that when one of these states is entered, a channel moves to the next stable state in a relatively short period.

Note

The specification does not define a maximum period in a transient state, but it is expected that for any given implementation it is deterministic.

The state is determined by the **LINKACTIVEREQ** and **LINKACTIVEACK** signals. Figure B14.3 shows the relationship between the four states.

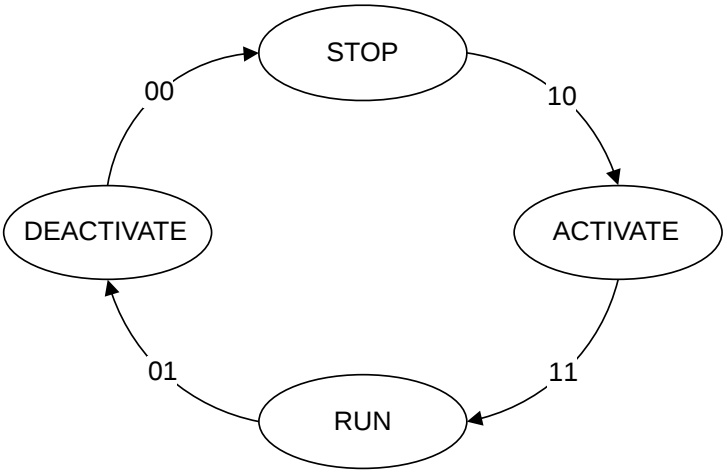

Figure B14.3: Request and Acknowledge handshake states

Table B14.1 shows the mapping of the states to the **LINKACTIVEREQ** and **LINKACTIVEACK** signals.

Table B14.1: Mapping of states to the LINKACTIVE signals

State	LINKACTIVEREQ	LINKACTIVEACK
STOP	0	0
ACTIVATE	1	0
RUN	1	1
DEACTIVATE	0	1

Table B14.2 describes the behavior of both the Transmitter and the Receiver of a single link for each of the four states.

Table B14.2: Behavior for each Request and Acknowledge state

State	Transmitter behavior	Receiver behavior
STOP	The Transmitter has no credits and must not send any flits.	The Receiver is guaranteed not to receive any flits.
	The Transmitter is guaranteed not to receive any credits.	The Receiver must not send any credits.
	The Transmitter must assert **LINKACTIVEREQ** to move to the ACTIVATE state if flits must be sent.	
ACTIVATE (ACT)	The Transmitter must not send any flits.	The Receiver is guaranteed not to receive any flits.
	The Transmitter must be prepared to receive credits in this state. However, the Transmitter must not use them until in the RUN state.	The Receiver must not send any credits.
	The Transmitter remains in the ACTIVATE state while waiting for the Receiver to acknowledge the move to the RUN state. **Note** The Transmitter only receives credits in the ACTIVATE state when there is a race between the Receiver sending credits and asserting **LINKACTIVEACK** to move to the RUN state.	The ACTIVATE state is a transient state and the Receiver controls the move to the RUN state by asserting **LINKACTIVEACK**. The Receiver must assert **LINKACTIVEACK** and move to the RUN state before sending credits. It is permitted, but not required, to assert **LINKACTIVEACK** and send a credit in the same cycle. **Note** A Receiver can appear to have sent credits in the ACTIVATE state if there is a race between the Receiver sending credits and asserting **LINKACTIVEACK** to move to the RUN state.
RUN	The Transmitter can receive credits.	The Receiver can receive flits corresponding to the credits previously sent.
	The Transmitter can send flits when credits are available.	The Receiver sends credits when resources are available to accept further flits.
	The Transmitter deasserts **LINKACTIVEREQ** to exit from this state if low power state is desired.	The Receiver must remain in the RUN state until observing the deassertion of **LINKACTIVEREQ**.
DEACTIVATE (DEACT)	The Transmitter must return credits using Protocol flits or L-Credit return flits.	During this state, the Receiver stops sending credits and collects all returned credits.

Continued on next page

Table B14.2 – *Continued from previous page*

State	Transmitter behavior	Receiver behavior
	It is recommended that the Transmitter enters the DEACTIVATE state only when no more Protocol flits can be sent. Therefore, it is expected that the Transmitter returns credits using only L-Credit return flits.	The Receiver must be prepared to receive flits, other than Link flits to return credits, in this state. This is not expected, but can occur.
	The Transmitter must be prepared to continue receiving credits. For each additional credit received, the Transmitter must send an L-Credit return flit to return the credit.	The Receiver is permitted, but not required, to send credits when first entering this state. However, the Receiver must have stopped sending credits and had all credits returned before exiting this state.
	The Transmitter remains in the DEACTIVATE state while waiting for the Receiver to acknowledge the move to the STOP state. At this point, the Receiver is guaranteed to receive no more credits.	The Receiver receives L-Credit return flits until all credits are returned. The Receiver must wait for all credits to be returned before deasserting **LINKACTIVEACK**. **Note** The Receiver only receives flits in the DEACTIVATE state when there is a race between the Transmitter sending the last remaining flits and deasserting **LINKACTIVEREQ** to move to the DEACTIVATE state.

Table B14.3 summarizes the required behavior described in detail in Table B14.2.

Table B14.3: Summary of behavior for each Request and Acknowledge state

State	Transmitter	Receiver
STOP	Must not send flits	Must not send credits
	Does not receive credits	Does not receive flits
ACT	Must not send flits	Must not send credits
	Must accept credits	Does not receive flits
RUN	Can send flits	Must accept flits
	Must accept credits	Can send credits
DEACT	Expected to send L-Credit return flits	Must accept flits
	Can send any flits	Must stop sending credits
	Must accept credits	
	Must return credits	

B14.5.1.1 Response to new state

When moving to a new state, where the state change has been initiated by the other-side of the interface, a component could be required to change its behavior.

If the state change requires a component to start sending flits or credits, there is no defined limit on the time taken for the component to start the new behavior. This new behavior only occurs in the new state.

If the state change requires a component to stop sending flits or credits, the component is permitted to take some time to respond. In this case, the behavior is observed when first entering a new state which is not expected within that state.

The state change from RUN to DEACTIVATE is the point at which flits and credits stop being sent.

Flits are sent by the Transmitter, which is also the component that determines the state change, and therefore the Transmitter can ensure flits are not sent after the state change.

Credits are sent by the Receiver, but that component does not determine the state change. The Receiver could take some time to react to the state change. Therefore, credits can be sent when first entering the DEACTIVATE state.

The CHI protocol requires that the Receiver has stopped sending credits and has had all credits returned before signaling the change from DEACTIVATE to STOP.

B14.5.1.2 Determining when to move to ACTIVATE or DEACTIVATE

For a given channel, or set of channels in the same direction, the Transmitter is always responsible for initiating the state change from RUN to STOP, or from STOP to RUN.

The Transmitter itself can determine that a state change is needed. This can happen through various mechanisms. The following examples are not exhaustive:

- The Transmitter has flits to send, so must move from STOP to RUN.
- The Transmitter can determine no activity can be performed for a significant period, so can move from RUN to STOP.
- The Transmitter can observe an independent sideband signal indicating a move either from RUN to STOP, or from STOP to RUN.
- The Transmitter can determine that a transaction is not fully complete and therefore the channels should remain in RUN state until all activity has completed.
- The Transmitter can observe a state change on the channel, or set of channels, that are used in the opposite direction. See B14.6 *Transmit and receive link interaction*.

B14.5.1.3 Multiple channels in the same direction

Figure B14.4 shows an example of a multiple channel interface, also referred to as a Link, that transfers payload flits in the same direction. A single pair of **LINKACTIVEREQ** and **LINKACTIVEACK** signals are used for all channels.

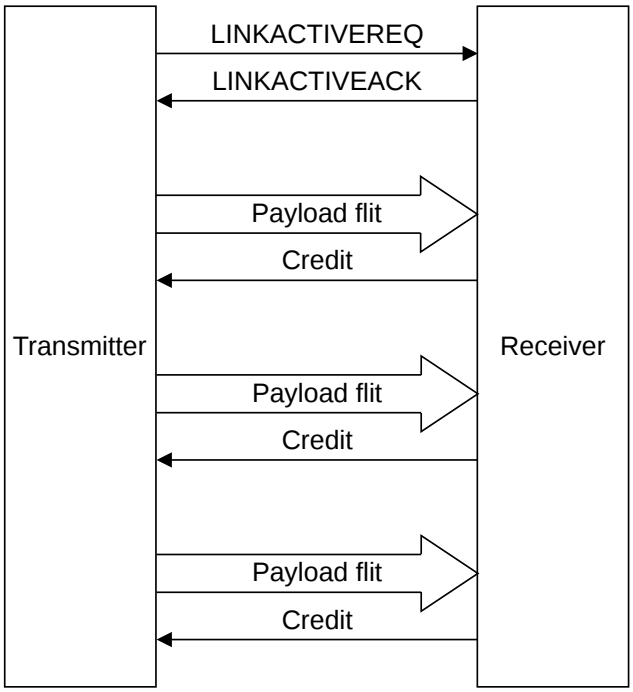

Figure B14.4: Example of a multiple channel unidirectional interface

The rules regarding the relationship between the **LINKACTIVEREQ** and **LINKACTIVEACK** signals must be applied appropriately across all channels:

- When a state change requires the Transmitter to be able to accept credits, the Transmitter must be able to accept credits on all channels.
- When a state change requires the Receiver to be able to accept flits, the Receiver must be able to accept credits on all channels.
- When the sending of flits must stop before a state change, the sending of flits must stop on all channels.
- When the sending of credits must stop before a state change, the sending of credits must stop on all channels.
- A credit can only be associated with a flit on the same channel.

B14.6 Transmit and receive link interaction

This section describes the interaction between a link Transmitter and Receiver. It contains the following subsections:

B14.6.1 Introduction

A single component has various different channels, some of which are inputs and some of which are outputs.

For a single component:

- All the channels where the Payload is an output are defined to be the Transmit Link (TXLINK).
- All the channels where the Payload is an input are defined to be the Receive Link (RXLINK).

It is required that the activation and deactivation of the TXLINK and RXLINK are coordinated.

When the TXLINK and RXLINK are both in the stable STOP state:

- If the RXLINK moves to the ACTIVATE state, which is controlled by the component on the other side of the interface, it is required that the TXLINK also moves to the ACTIVATE state, in a timely manner.
- If a component moves the TXLINK to the ACTIVATE state, which is controlled by the component, the RXLINK is expected to also move to the ACTIVATE state, in a timely manner.

When the TXLINK and RXLINK are both in the stable RUN state:

- If the RXLINK moves to the DEACTIVATE state, which is controlled by the component on the other side of the interface, it is required that the TXLINK also moves to the DEACTIVATE state, in a timely manner.
- If a component moves the TXLINK to the DEACTIVATE state, which is controlled by the component, the RXLINK is expected to also move to the DEACTIVATE state, in a timely manner.

When the TXLINK and RXLINK are changing states, the rules about the sending and receiving of credits and flits can be considered independently for each link.

B14.6.2 Tx and Rx state machines

Figure B14.5 shows the permitted relationships between the Tx and Rx state machines. Figure B14.5 is formatted so that the independent nature of the Tx and Rx state machines can be seen.

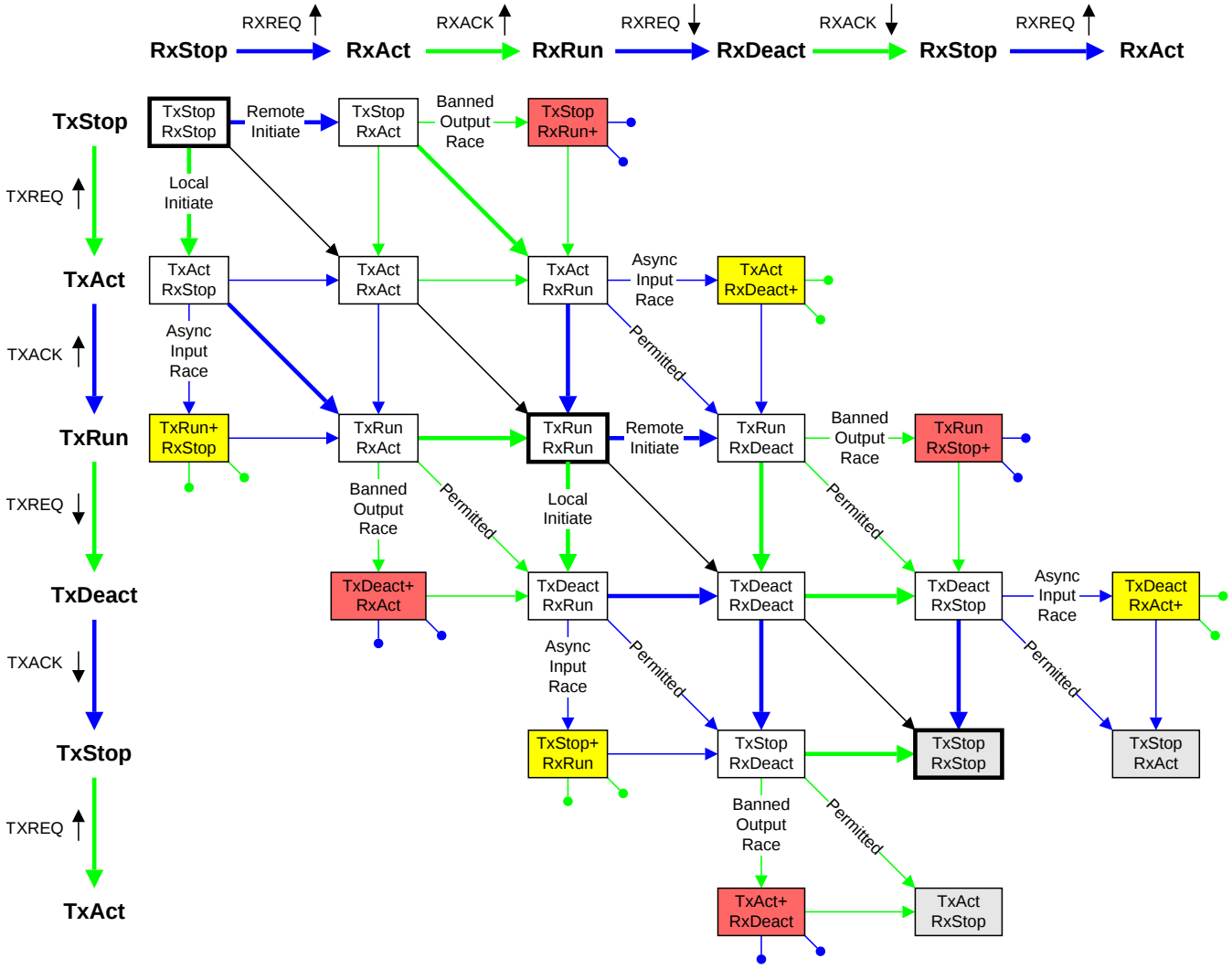

Figure B14.5: Combined Tx and Rx state machines

Figure B14.5 shows the combined Tx and Rx state machines for a single component:

- For clarity, shortened state names and signal names are used.
- A green arrow represents a transition that the local agent can control.
- A blue arrow represents a transition that is under the control of the remote agent on the other side of the interface.
- A black arrow represents a transition that is made when both the local and remote agents make a transition at the same time.
- Around the edge of Figure B14.5 is an indication of the individual Tx and Rx states. The green and blue arrows show which agent controls the transition. There is also an indication of the signal change that causes the state transition.
- A vertical or horizontal arrow is a state change caused by just one signal change, that is, only the Rx state machine or the Tx state machine changes state, not both.
- A diagonal arrow is a state change caused by two signals changing at the same time. If the diagonal arrow is green or blue, the same agent is changing both signals.

- There are a few cases where, by coincidence, a state change occurs due to two events, one on each side of the link, occurring at the same time. This is always a diagonal path and is shown by a black arrow.
- The stub-lines show dead-end paths where an exit from a state is not permitted. The color of a stub-line indicates which agent is responsible for ensuring that the path is not taken.
- The TxStop/RxStop and TxRun/RxRun states are expected to be stable states, and are typically the states where the state machines stay for long periods of time. These states are highlighted with a bold outline. All other states are considered transient states that are exited in a timely manner.
- The gray states, on the bottom right of Figure B14.5, are replications of those on the top left. They are shown to aid clarity and maintain the symmetry of the diagram.
- The yellow states can only be reached by observing a race between two input signals. The transition into these states is labeled with Async Input Race. See B14.6.3.4 *Asynchronous race condition*.
- The red states can only be reached by observing a race between two output signals. A race between two outputs is not permitted at the edge of a component and therefore the transition into these states is labeled with Banned Output Race. These states can only be observed at a midpoint between two components. See B14.6.3.4 *Asynchronous race condition*.
- The bold arrows are used to indicate the expected transitions around the state machine. These are described in more detail in B14.6.3 *Expected transitions*.
- The arrows labeled Permitted are state transactions that would not normally be expected, but they are permitted by the protocol.

B14.6.2.1 State naming

Figure B14.5 shows the full set of states, including those that can only be reached through race conditions. A more detailed discussion of race conditions can be found in B14.6.3.4 *Asynchronous race condition*.

There are two different TxStop/RxRun states and two different TxRun/RxStop states. These states differ in how they are reached and the permitted ways to exit from them. To differentiate between these states, a [+] suffix is used to indicate which state machine, Tx or Rx, is running ahead. For example:

- TxStop/RxRun+ indicates that the Tx state machine has remained in the previous STOP state, while the Rx state machine has advanced to the next RUN state.
- TxStop+/RxRun indicates that the Tx state machine has advanced to the next STOP state, while the Rx state machine remains in the previous RUN state.

B14.6.3 Expected transitions

Figure B14.6 shows the expected state transitions.

The annotations on the diagram arrows in Figure B14.6 are:

Local Initiate Indicates that the local agent has initiated the process of leaving one stable state towards the other stable state.

Remote Initiate Indicates that the remote agent on the other side of the interface has initiated the process of leaving one stable state towards the other stable state.

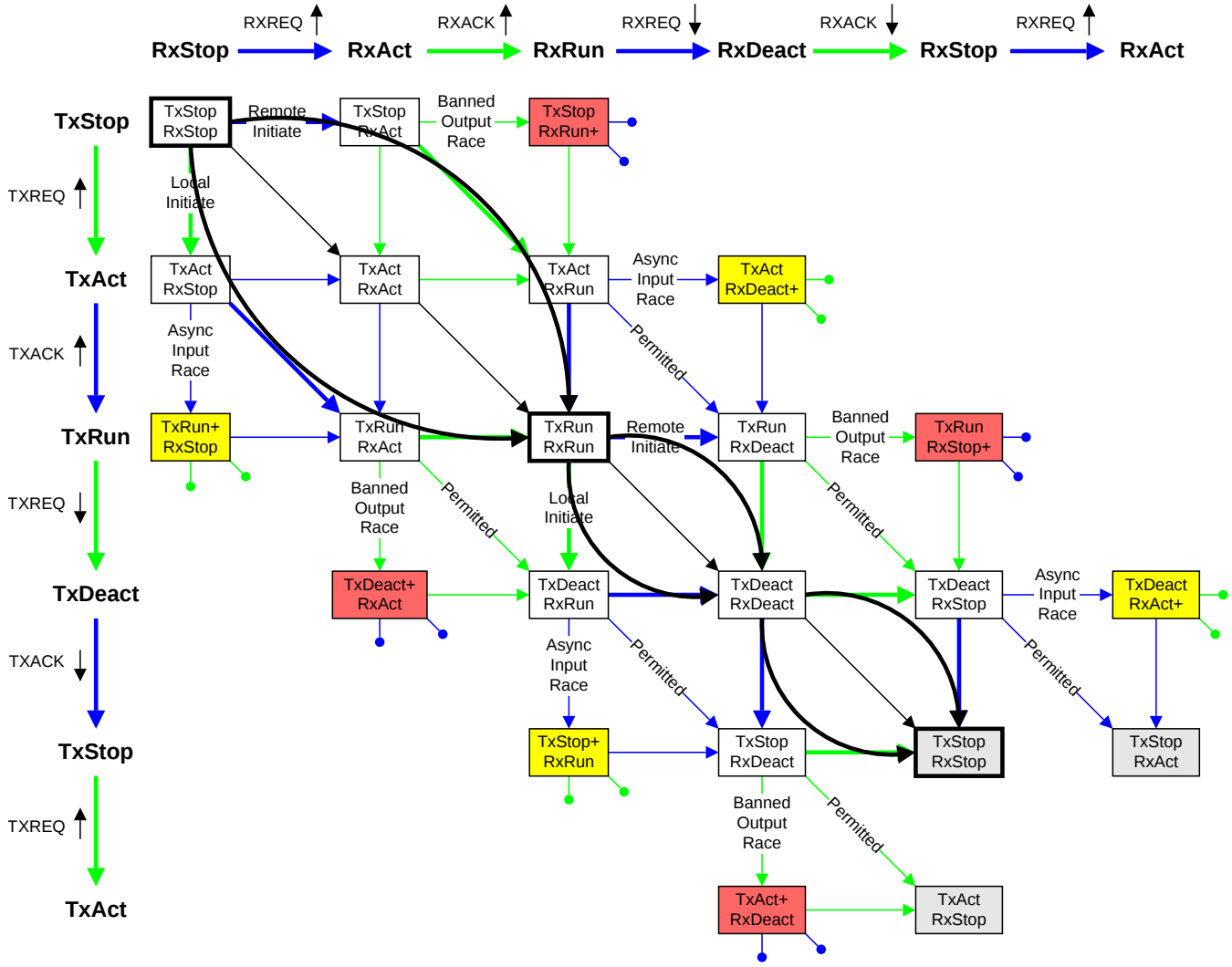

Figure B14.6: Expected Tx and Rx state machines transitions

Figure B14.6 shows, using bold arrows, the routes between the stable TxStop/RxStop and TxRun/RxRun states, and between the stable TxRun/RxRun and the TxStop/RxStop states.

The difference between the two routes moving from TxStop/RxStop to TxRun/RxRun states compared to moving from TxRun/RxRun to TxStop/RxStop states is due to the requirement to return L-Credits in the latter case. The differences are detailed in the following sections.

B14.6.3.1 Expected transitions from TxStop/RxStop to TxRun/RxRun

There are two expected routes from a stable Stop/Stop to Run/Run state.

Table B14.4 shows, in terms of the state transitions, the two expected paths.

Table B14.4: Stop/Stop to Run/Run state paths

Route	State 1	State 2	State 3	State 4
Path 1	TxStop/RxStop	TxStop/RxAct	TxAct/RxRun	TxRun/RxRun

Continued on next page

Table B14.4 – *Continued from previous page*

Route	State 1	State 2	State 3	State 4
Path 2	TxStop/RxStop	TxAct/RxStop	TxRun/RxAct	TxRun/RxRun

B14.6.3.2 Expected transitions from TxRun/RxRun to TxStop/RxStop

A transition from a Run/Run state to a Stop/Stop state requires that L-Credits are returned. A link must remain in the DEACTIVATE state until all L-Credits are returned.

There are four expected routes from a stable Run/Run to Stop/Stop state.

Table B14.5 shows, in terms of the state transitions, the four expected paths.

Table B14.5: State 5

Route	State 1	State 2	State 3	State 4	State 5
Path 1	TxRun/RxRun	TxDeact/RxRun	TxDeact/RxDeact	TxStop/RxDeact	TxStop/RxStop
Path 2	TxRun/RxRun	TxDeact/RxRun	TxDeact/RxDeact	TxDeact/RxStop	TxStop/RxStop
Path 3	TxRun/RxRun	TxRun/RxDeact	TxDeact/RxDeact	TxStop/RxDeact	TxStop/RxStop
Path 4	TxRun/RxRun	TxRun/RxDeact	TxDeact/RxDeact	TxDeact/RxStop	TxStop/RxStop

B14.6.3.3 Transitions around a stable state

It is permitted, but not expected, to transition around a stable TxRun/RxRun or TxStop/RxStop state.

In the majority of cases, moving to the stable Run/Run or Stop/Stop state would be expected.

The most likely use case for wanting to move quickly out of one of the stable states is when an interface has started to enter a low power state, but there is still some activity required. For example, the low power state could have been entered prematurely, or some new activity arose, by coincidence, while the low power state was being entered. In this use case, it is desirable to be able to move back to the Run/Run state as quickly as possible.

B14.6.3.4 Asynchronous race condition

There are situations where two output signals, X and Y, have a defined relationship such that:

- Output X must change after or at the same time as output Y, but it is not permitted to change before output Y.

This relationship applies specifically as follows:

- The assertion of RXACK must not occur before the assertion of TXREQ.
- The deassertion of RXACK must not occur before the deassertion of TXREQ.
- The assertion of TXREQ must not occur before the deassertion of RXACK.
- The deassertion of TXREQ must not occur before the assertion of RXACK.

In Figure B14.5, these transitions are labeled as Banned Output Race and the resultant state is shown in red.

It is possible to observe these states if monitoring the output signals at a point in the system where asynchronous race conditions can result in two signals, that are asserted within the same cycle, are observed in different clock cycles.

A component that is on the other side of the interface, and has the two signals as inputs, can see the state transition if an asynchronous input race occurs. These transitions are labeled on the diagram as Aysnc Input Race and the resultant state is shown in yellow.

For all input race conditions, a component that observes the input race is required to wait for both signals before changing any output signals. This is represented in Figure B14.5 by the fact that the only permitted output transition from a race state is caused by the arrival of the other signal associated with the race condition.

B14.6.3.5 Combined Tx and Rx state machines without race conditions

In Figure 14-7, all transitions and states that occur as a result of a race condition in the combined Tx and Rx state machines have been removed.

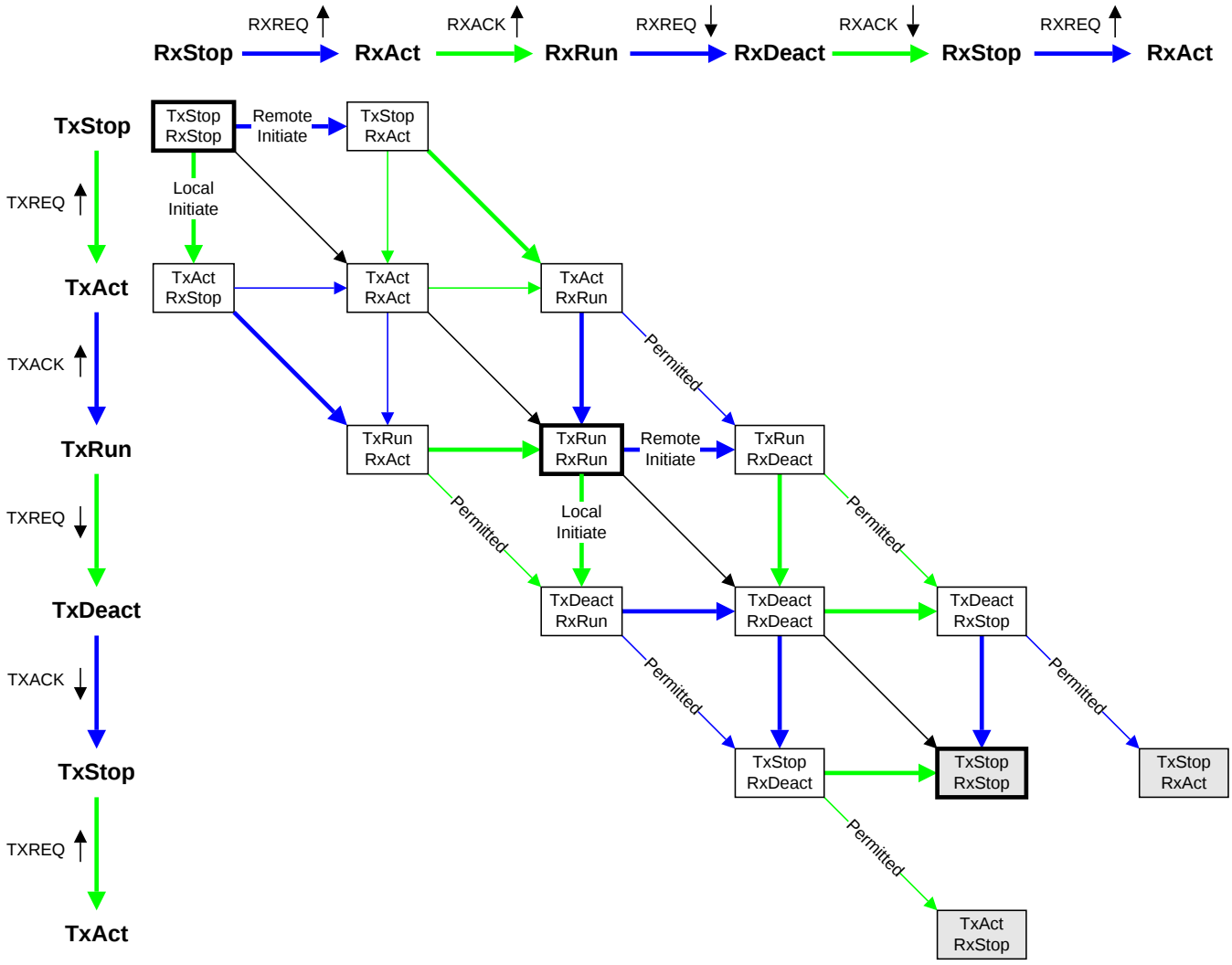

Figure B14.7: Combined Tx and Rx state machines without race conditions

B14.7 Protocol layer activity indication

This section describes the signals that indicate Protocol layer activity. It contains the following subsections:

B14.7.1 Introduction

SACTIVE signaling indicates that there are transactions in progress.

TXSACTIVE is an output signal that is asserted by an interface where there is a transaction either in progress or about to start:

- **TXSACTIVE** must be asserted before or in the same cycle in which the first flit relating to a transaction is sent.
- **TXSACTIVE** must remain asserted until after the last flit relating to all transactions is sent or received.

This means that the deassertion of **TXSACTIVE** on an interface implies that the component has completed all transactions in progress and does not need to send or receive any further flits.

A transaction that is given a RetryAck response is considered to be in progress, **TXSACTIVE** must remain asserted until the associated credit has been supplied and used or returned.

RXSACTIVE is an input signal which indicates that the other side of the interface has ongoing Protocol layer activity. When **RXSACTIVE** is asserted a component must respond to Protocol layer activity in a timely manner.

SACTIVE signals must be synchronous to **CLK** and therefore are not required to be synchronized. If they cross a clock domain, the clock domain crossing bridge is required to synchronize the signals.

B14.7.2 TXSACTIVE signal

The following rules apply to the **TXSACTIVE** signal:

- **TXSACTIVE** must be asserted when the transmitter has flits to send.
- A component that asserts **TXSACTIVE** must also, if required, initiate the link activation sequence. It is not permitted for a component to assert the **TXSACTIVE** signal and subsequently wait for the other side of the interface to initiate the link activation sequence.
- **TXSACTIVE** must remain asserted until after the last flit relating to all transactions is sent or received.
- It is permitted, but not required, for **TXSACTIVE** to be deasserted while transmitting link flits as part of the link deactivation sequence.

Note

To ensure an efficient power down sequence, it is recommended not to assert a deasserted **TXSACTIVE** signal during a link deactivation sequence.

It is permitted for the interface on an interconnect component to use the RXSACTIVE input signal to directly generate the TXSACTIVE output signal. This behavior is only permitted on the interconnect interface and it is not permitted on any attached component.

Except for an interconnect interface of a Link, no other interface of a Link is permitted to loop-back the incoming **RXSACTIVE** onto the outgoing **TXSACTIVE**.

Figure B14.8 shows the requirements for **TXSACTIVE** assertion during the life of a transaction.

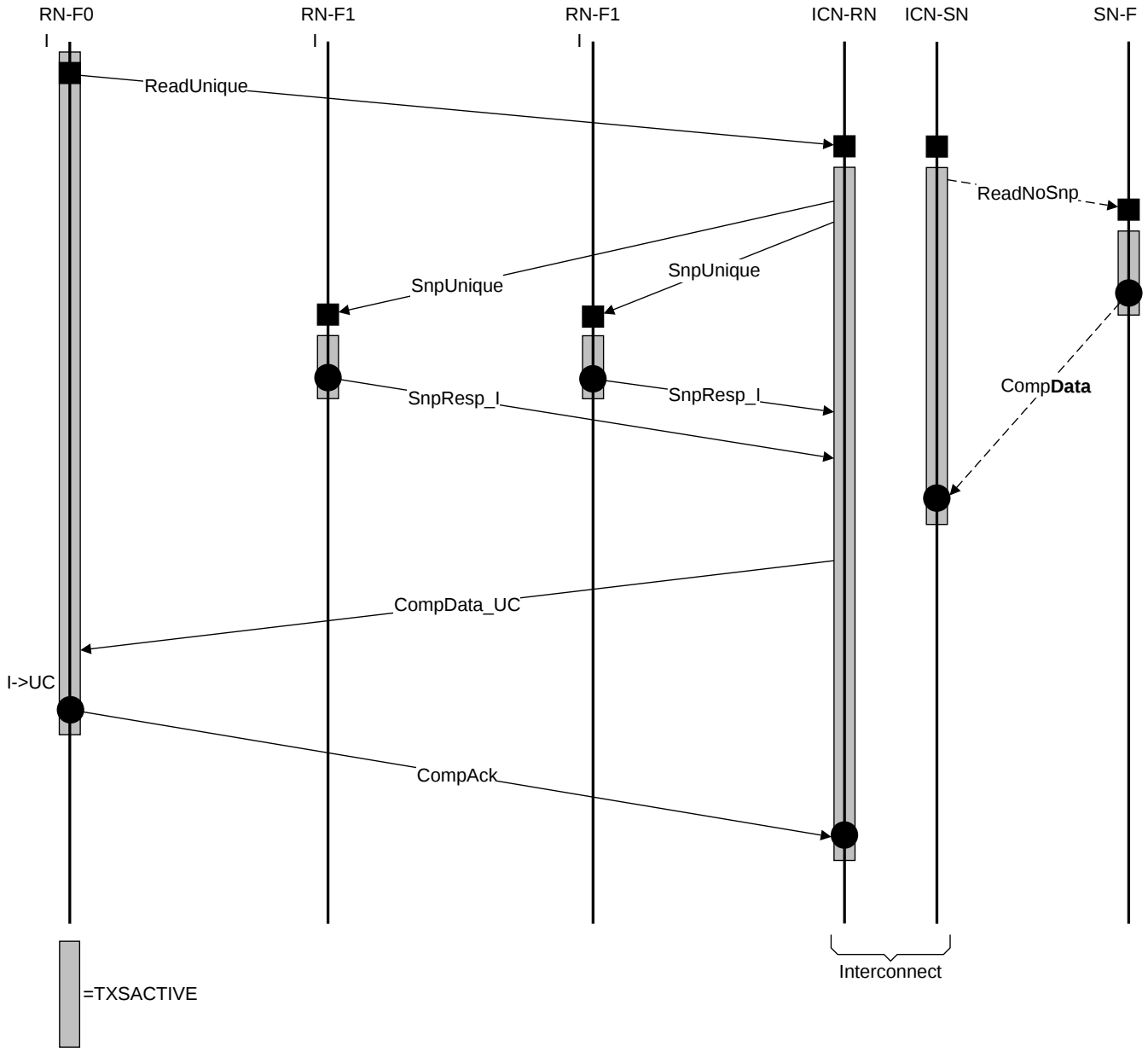

Figure B14.8: TXSACTIVE assertion during the life of a transaction

B14.7.2.1 TXSACTIVE signaling from a Request Node

When initiating new transactions, a Request Node must assert **TXSACTIVE** in the same cycle or before **TXREQFLITV** is asserted and must keep **TXSACTIVE** asserted until after the final completing flit of a transaction is sent or received.

The type of flit that completes a transaction initiated by a Request Node depends on both the transaction type and the manner in which the transaction progresses. For example, a ReadNoSnp transaction could typically complete with the receipt of the last CompData flit, but could equally complete with a ReadReceipt, if this is later than the last CompData flit.

An RN-F or RN-D component must also assert **TXSACTIVE** while a Snoop transaction is in progress.

TXSACTIVE must be asserted after receiving an initiating snoop or SnpDVMOp flit, and no later than when its first Response flit is sent. An RN-F or RN-D component must keep **TXSACTIVE** asserted until after the final completing flit is sent for all Snoop transactions. The lifetime of a PrefetchTgt is a single flit. From the perspective of **TXSACTIVE** for the PrefetchTgt, the initiating flit of the a PrefetchTgt transaction is also the completing flit and the transaction can be considered complete in the following cycle after the flit is sent.

For an RN-F or RN-D, the **TXSACTIVE** output is the logical OR of the requirements for the Request interface and the Snoop interface.

B14.7.2.2 TXSACTIVE signaling from a Subordinate Node

A Subordinate Node cannot initiate new transactions and is only required to assert **TXSACTIVE** while a transaction that is in progress is being processed.

A Subordinate Node must assert **TXSACTIVE** after receiving a transaction initiating flit and **TXSACTIVE** must be asserted before or in the same cycle in which its first Response flit is sent. The Subordinate Node must keep **TXSACTIVE** asserted until after the final completing flit is sent or received.

B14.7.2.3 TXSACTIVE signaling from an interconnect interface to a Request Node

The interconnect interface to a Request Node must assert **TXSACTIVE** in both the following conditions:

- On receiving a transaction initiating flit, a Request Node must be asserted before or in the same cycle in which its first Response flit is sent. The Request Node must keep **TXSACTIVE** asserted until after the final completing flit is sent or received.
- Before or in the same cycle in which its initiating Snoop or SnpDVMOp flit is sent. A Request Node must keep **TXSACTIVE** asserted until after the final completing flit is sent, which is either SnpResp or SnpRespData.

B14.7.2.4 TXSACTIVE signaling from an interconnect interface to a Subordinate Node

The interconnect interface to a Subordinate Node must assert **TXSACTIVE** before or in the same cycle in which its initiating Request flit is sent. The Subordinate Node must keep **TXSACTIVE** asserted until after the final completing flit is sent or received.

B14.7.3 RXSACTIVE signal

When **RXSACTIVE** is asserted, the receiver must respond to a link activation request in a timely manner. It is permitted for a Receiver to delay responding to a Link activation request when **RXSACTIVE** is deasserted.

Note

The deassertion of **RXSACTIVE** does not indicate that all Protocol layer activity has completed. A Receiver can receive a Protocol flit, which corresponds to a transaction that was in progress while **RXSACTIVE** was asserted, after **RXSACTIVE** is deasserted.

RXSACTIVE can be used in combination with a knowledge of the ongoing transactions, which is indicated by the components **TXSACTIVE** output, to indicate that no further transactions are required. This can be used to control entry to a low power state.

B14.7.4 Relationship between SACTIVE and LINKACTIVE

SACTIVE signaling is an indication of Protocol layer activity. A node can be considered inactive when both **TXSACTIVE** and **RXSACTIVE** are deasserted.

LINKACTIVE state is an indication of the Link layer activity. The Link layer at a node, or interconnect, can be considered inactive when its Transmitter is in TxStop state and its Receiver is in RxStop state.

SACTIVE signaling is orthogonal to the LINKACTIVE states with one constraint as specified in B14.7.3 *RXSACTIVE signal*.

A node, or interconnect, should only enable higher-level clock gating and low power optimizations when both its Protocol and Link layers are inactive.

Chapter B15
System Coherency Interface

This chapter describes the interface signals that support connecting and disconnecting an RN-F from both the Coherency and DVM domains and an RN-D from the DVM domain. It contains the following sections:

B15.1 Overview

The system coherency interface signals are:

SYSCOREQ Requester coherency request.

SYSCOACK Interconnect coherency acknowledge.

Both **SYSCOREQ** and **SYSCOACK** signals must be synchronous to CLK and therefore are not required to be synchronized. If they cross a clock domain, the clock domain crossing bridge is required to synchronize the signals.

Figure B15.1 shows the system coherency interface signals connections.

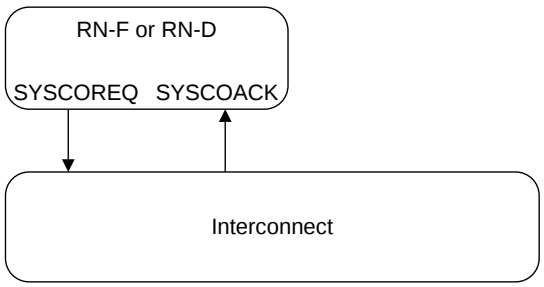

Figure B15.1: System coherency interface signals

Note

In this chapter:

- Coherency when stated, includes the DVM domain, unless explicitly stated otherwise.
- Snoop when stated, includes SnpDVMOp, unless explicitly stated otherwise.

B15.2 Handshake

A Request Node, an RN-F or an RN-D, requests connection to system coherency by setting **SYSCOREQ** HIGH. The interconnect indicates that coherency is enabled by setting **SYSCOACK** HIGH.

The Request Node requests disconnection from system coherency by setting **SYSCOREQ** LOW. The interconnect indicates that coherency is disabled by setting **SYSCOACK** LOW.

Requests to enter and exit coherency are always initiated by the Request Node.

Figure B15.2 shows the system coherency interface handshake timing.

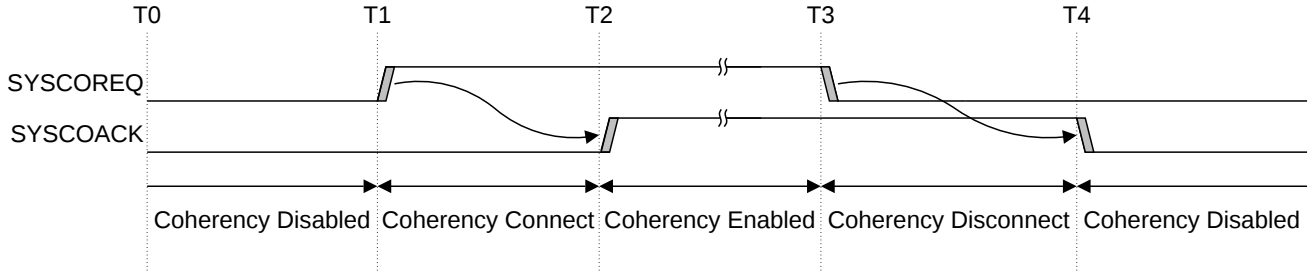

Figure B15.2: System coherency interface handshake timing

As Figure B15.2 shows, the interface signaling obeys four-phase handshake rules:

- **SYSCOREQ** can only change when **SYSCOACK** is at the same logic state.
- **SYSCOACK** can only change when **SYSCOREQ** is at the opposite logic state.

B15.2.1 Request Node rules

Referring to Figure B15.2, a Request Node must:

- Be able to service Snoop requests when **SYSCOREQ** is HIGH at T1.
- Not issue a transaction that permits a coherent location to be cached until **SYSCOACK** is HIGH at T2.
- Ensure all transactions that permit a coherent location to be cached are complete before **SYSCOREQ** is LOW at T3.
  - **SYSCOREQ** can only be deasserted on the cycle after all of the following:
    - All data packets are received for:
      - ReadUnique
      - ReadPreferUnique
      - ReadClean
      - ReadNotSharedDirty
      - ReadShared
      - MakeReadUnique that completes with a data transfer
    - Comp is received for:
      - CleanUnique
      - MakeUnique
      - MakeReadUnique that completes without a data transfer
    - All data packets are sent for a CopyBack transaction that completes with a data transfer.
    - CompAck is sent for a CopyBack transcation that completes without a data transfer.

    - All data packets are sent for snoops and Forwarding snoops.
- Keep servicing Snoop requests until **SYSCOACK** is sampled LOW at T4.

SACTIVE must be asserted during coherency state transition periods to guarantee the **SYSCOACK** transition occurs. See B14.7 *Protocol layer activity indication*.

B15.2.2 Interconnect rules

Referring to Figure B15.2:

The interconnect must do the following when **SYSCOREQ** is sampled HIGH:

- Set **SYSCOACK** HIGH without waiting for responses to any previous Snoop requests after **SYSCOREQ** goes HIGH.
- Be able to service coherent data accesses from the interface when **SYSCOACK** is HIGH at T2.

The interconnect must do the following when **SYSCOREQ** is sampled LOW:

- Stop issuing new Snoop requests in a timely manner.
- Complete all snoop accesses to the interface before it sets *SYSCOACK* LOW at T4.

B15.2.3 Protocol states

Table B15.1 shows the interface states and the rules that the Requester must follow in relation to the interface state.

Table B15.1: System coherency interface states

State name	SYSCOREQ	SYSCOACK	Request Node	Interconnect
Coherency Disabled	0	0	Caches must not contain coherent data. Must not issue transactions that permit the Request Node to cache a coherent location. Must not send DVM transactions. Not required to respond to Snoop requests.	Must not send Snoop requests
Coherency Connect	1	0	Caches must not contain coherent data. Must not issue transactions that permit the Request Node to cache a coherent location. Must not send DVM transactions. Must respond to Snoop requests.	Can send Snoop requests

Continued on next page

Table B15.1 – *Continued from previous page*

State name	SYSCOREQ	SYSCOACK	Request Node	Interconnect
Coherency Enabled	1	1	Caches can contain coherent data. Can issue transactions that permit the Request Node to cache a coherent location. Can send DVM transactions. Must respond to Snoop requests.	Can send Snoop requests
Coherency Disconnect	0	1	Caches must not contain coherent data. Must not issue transactions that permit the Request Node to cache a coherent location. Must not send DVM transactions. Must respond to Snoop requests.	Must stop issuing new Snoop requests in a timely manner. The interconnect must then complete all outstanding Snoop requests before **SYSCOACK** can be deasserted.

Chapter B16
Properties, Parameters, and Broadcast Signals

This chapter describes the properties, parameters, and optional broadcast signals that specify the behavior supported by an interface. It contains the following sections:

B16.1 Interface properties and parameters

A property is used to declare a capability.

The properties and parameters that specify the interface behavior are described in the following sections:

B16.1.1 Atomic_Transactions

An Atomic_Transactions property is used to indicate if a component supports Atomic transactions.

Table B16.1 shows the Atomic_Transactions property options.

Table B16.1: Atomic_Transactions property options

Atomic_Transactions value	Description
True	Atomic transactions are supported.
False or Not specified	Atomic transactions are not supported.

A component that supports Atomic transactions must support all Atomic transactions. However, it is not required that a component that supports Atomic transactions supports the targeting of all memory types. See B16.3 *Atomic transaction support*.

B16.1.2 Cache_Stash_Transactions

A Cache_Stash_Transactions property is used to indicate if a component supports Cache Stashing transactions.

Table B16.2 shows the Cache_Stash_Transactions propety options.

Table B16.2: Cache_Stash_Transaction property options

Cache_Stash_Transactions value	Description
True	Cache Stashing transactions are supported.
False or Not specified	Cache Stashing transactions are not supported.

B16.1.3 Direct_Memory_Transfer

A Direct_Memory_Transfer property is used to indicate if a component supports Direct Memory Transfer transactions.

Table B16.3 shows the Direct_Memory_Transfer property options.

Table B16.3: Direct_Memory_Transfer property options

Direct_Memory_Transfer value	Description
True	Direct Memory Transfer transactions are supported.
False or Not specified	Direct Memory Transfer transactions are not supported.

The Direct_Memory_Transfer property is defined at each Home Node for each Subordinate Node.

B16.1.4 Data_Poison

A Data_Poison property is used to indicate if a component supports Poison.

Table B16.4 shows the Data_Poison property options.

Table B16.4: Data_Poison property options

Data_Poison value	Description
True	Poison is supported and the Poison field is present in the DAT packet
False or Not specified	Poison is not supported and the Poison field is not present in the DAT packet

See B9.2.1 *Poison*.

B16.1.5 Direct_Cache_Transfer

A Direct_Cache_Transfer property is used to indicate if a component supports Direct Cache Transfer transactions.

Table B16.5 shows the Direct_Cache_Transfer property options.

Table B16.5: Direct_Cache_Transfer property options

Direct_Cache_Transfer value	Description
True	Direct Cache Transfer transactions are supported
False or Not specified	Direct Cache Transfer transactions are not supported

The HN-F is responsible to determine the correct snoop type to use.

B16.1.6 Data_Check

A Data_Check property is used to indicate the DataCheck field is present in the DAT packet.

Table B16.6 shows the Data_Check property options.

Table B16.6: Data_Check property options

Data_Check value	Description
Odd_Parity	Data Check is supported and the DataCheck field is present in the DAT packet. If Check_Type is defined and set to Odd_Parity_Byte_All, the DataCheck field is not present in the DAT packet.
False or Not specified	The DataCheck field is not in the DAT packet unless specified by the Check_Type property

See B9.2.2 *Data Check*.

B16.1.7 Check_Type

A Check_Type property is used to indicate the protection scheme employed on an interface.

Table B16.7 shows the Check_Type property options.

Table B16.7: Check_Type property options

Check_Type value	Description
Odd_Parity_Byte_All	Parity check signals are added to every channel. The signals added are detailed in B9.3 *Use of interface parity*.
Odd_Parity_Byte_Data	The DataCheck field is present in the DAT packet
False or Not specified	No checking signals are present on the interface, unless specified by the Data_Check property

B16.1.8 CleanSharedPersistSep_Request

A CleanSharedPersistSep_Request property is used to indicate if a component supports CleanSharedPersistSep.

Table B16.7 shows the CleanSharedPersistSep_Request property options.

Table B16.8: CleanSharedPersistSep_Request property options

CleanSharedPersistSep_Request value	Description
True	The component supports CleanSharedPersistSep
False or Not specified	The component does not support CleanSharedPersistSep and the component must not be sent a CleanSharedPersistSep request.

The CleanSharedPersistSep_Request property has the following conditions:

- A Home that receives a CleanSharedPersistSep request must support such a request.
- A Home can track if a connected Subordinate supports CleanSharedPersistSep.
- If a Subordinate does not support CleanSharedPersistSep:
  - The Home must send a CleanSharedPersist instead of a CleanSharedPersistSep request to that Subordinate.
  - Home must take the responsibility of sending the Persist response to the Requester. This Persist response must only be sent after receiving the Comp response from the Subordinate.
- A Requester that does not support CleanSharedPersistSep generates CleanSharedPersist instead.

B16.1.9 MPAM_Support

The MPAM_Support property is used to indicate whether an interface supports MPAM.

Table B16.9 shows the MPAM_Support property options.

Table B16.9: MPAM_Support property options

MPAM_Support value	Description
MPAM_9_1	The interface is enabled for partitioning and monitoring: – The MPAM field must be included on the REQ, DAT, and SNP channels. – The width of PartID is 9 bits, the width of PerfMonGroup is 1 bit, and the width of MPAMSP is 2 bits.
False or Not specified	MPAM is not supported: – The interface is not MPAM enabled. – No MPAM fields are present on the interface.

How the MPAM field values are used by a Receiver is IMPLEMENTATION DEFINED.

B16.1.10 CCF_Wrap_Order

See B2.8.7 *Critical Chunk First Wrap order*.

B16.1.11 Req_Addr_Width

The Req_Addr_Width parameter is used to indicate the maximum PA supported by a component.

Table B16.10 shows the Req_Addr_Width parameter options.

Table B16.10: Req_Addr_Width parameter options

Req_Addr_Width value	Description
44 to 52	Legal values
Not specified	Defaul value is 44

B16.1.12 NodeID_Width

The NodeID_Width parameter is used to indicate the width of NodeID fields supported by a component, which determines the maximum NodeID value in the system.

Table B16.11 shows the NodeID_Width parameter options. The width specified in Table B16.11 is uiformly applied to all NodeID related fields.

Table B16.11: NodeID_Width parameter options

NodeID_Width value	Description
7 to 11	Legal values
Not specified	Defaul value is 7

B16.1.13 Data_Width

The Data_Width parameter is used to indicate the data width in the DAT channel packet supported by a component.

Table B16.12 shows the Data_Width parameter options.

Table B16.12: Data_Width parameter options

Data_Width value	Description
128, 256, and 512	Legal values
Not specified	Defaul value is 128

B16.1.14 Enhanced_Features

The Enhanced_Features property describes the combined support for the following features:

- Data return from SC state

- IO deallocation transactions
- ReadNotSharedDirty transaction
- CleanSharedPersist transaction
- Receiving of forwarding snoops.

Table B16.13 shows the Enhanced_Features property options.

Table B16.13: Enhanced_Features property options

Enhanced_Features value	Description
True	The component support the features that are covered by the Enhanced_Features property.
False or Not specified	The component does not support the features that are covered by the Enhanced_Features property.

B16.1.15 Deferrable_Write

The Deferrable_Write property is used to indicate whether a component supports WriteNoSnpDef transaction.

Table B16.14 shows the Deferrable_Write property options.

Table B16.14: Deferrable_Write property options

Deferrable_Write value	Description
True	WriteNoSnpDef is supported
False or Not specified	WriteNoSnpDef is not supported

B16.1.16 RME_Support

An RME_Support property determines whether an interface supports RME transactions.

Table B16.15 shows the RME_Support property options.

Table B16.15: RME_Support property options

RME_Support value	Description
True	The interface supports RME transactions. RME_Support can only be True if Nonshareable_Cache_Maint property is True and DVM_Support property is DVM_v9.2.
False or Not specified	The interface does not support RME transactions

B16.1.17 Nonshareable_Cache_Maint

A Nonshareable_Cache_Maint property is used to indicate if an RN-F or interconnect supports the cache maintenance of a Non-snoopable cacheable location from a Requester that is different to the Requester that

originally accessed that location.

Table B16.16 shows the Nonshareable_Cache_Maint property options.

Table B16.16: Nonshareable_Cache_Maint property options

Nonshareable_Cache_Maint value	Description
True	An RN-F must update all accesses to locations that are marked in the page tables as Non-shareable Cacheable to be Shareable Cacheable. For more information, see B2.7.7 *Mismatched memory attributes*. An HN-I receiving a request with both SnpAttr and Cacheable asserted must respond with NDERR when completing the transaction, except for when the transaction is a ReadOnce*, WriteUnique, WriteUniqueZero, or an Atomic. An HN-I responding with NDERR upon receiving a ReadOnce*, WriteUnique, WriteUniqueZero, or an Atomic request with both SnpAttr and Cacheable asserted is IMPLEMENTATION DEFINED. When RME_Support property is True, Nonshareable_Cache_Maint property must be True. A system fully supports the cache maintenance of Non-snoopable Cacheable memory when all RN-Fs and the interconnect define the Nonshareable_Cache_Maint property as True.
False or Not specified	There are no additional requirements on an RN-F. An HN-I responding with NDERR upon receiving a request with both SnpAttr and Cacheable asserted is IMPLEMENTATION DEFINED.

The Nonshareable_Cache_Maint property is inapplicable for RN-Is.

B16.1.18 PBHA_Support

A PBHA_Support property determines whether an interface supports PBHA functionality.

Table B16.17 shows the PBHA_Support property options.

Table B16.17: PBHA_Support property options

PBHA_Support value	Description
True	PBHA feature is supported.

Continued on next page

Table B16.17 – *Continued from previous page*

PBHA_Support value	Description
	The 4-bit PBHA field is present on REQ, DAT, and SNP channels.
False or Not specified	PBHA feature is not supported. No PBHA signals are present on the interface.

B16.1.19 DVM_Support

A DVM_Support property is used to indicate Arm ARM specification supported by a given component.

Table B16.18 shows the DVM_Support property options.

Table B16.18: DVM_Support property options

DVM_Support value	Description
DVM_v8	The component supports DVM transactions required to support Armv8
DVM_v8.1	The component supports DVM transactions required to support Armv8.1
DVM_v8.4	The component supports DVM transactions required to support Armv8.4
DVM_v9.2	The component supports DVM transactions required to support Armv9.2
False or Not specified	The component does not support DVM transactions

In a system with heterogeneous components, the system configuration is responsible to determine the lowest common DVM specification supported in the system. The interconnect must have the knowledge of this lowest common denominator value by means of configuration, straps, parameterization, or some other IMPLEMENTATION DEFINED method.

To avoid deadlocks and denial of service, the interconnect must detect unsupported DVM operations. The interconnect must suppress unsupported DVM operation propagation and respond in a protocol-compliant manner. Error indication in such a response is optional.

For further information, see Chapter B8 *DVM Operations*.

B16.2 Optional interface broadcast signals

This specification includes optional pins to determine broadcasting of certain groups of transactions in the interconnect. These pins are optional at the Request Node to the interconnect, and the interconnect to the Subordinate Node interfaces. The optional broadcast pins are:

- B16.2.1 *BROADCASTINNER and BROADCASTOUTER*
- B16.2.2 *BROADCASTCACHEMAINT*
- B16.2.3 *BROADCASTPERSIST*
- B16.2.4 *BROADCASTATOMIC*
- B16.2.5 *BROADCASTICINVAL*
- B16.2.6 *BROADCASTMTE*
- B16.2.7 *BROADCASTERTLBIINNER and BROADCASTTLBIOUTER*
- B16.2.8 *BROADCASTCMOPOPA*

An implementation that includes these signals at the interface must ensure that the signal values are stable when Reset is deasserted.

B16.2.1 BROADCASTINNER and BROADCASTOUTER

The **BROADCASTINNER** and **BROADCASTOUTER** signals are used to control the issuing of Snoopable transactions from an interface. It is required that these two pins must be set to the same value.

When the signals are present and deasserted, all transactions are converted to Non-snoopable equivalents before they are sent. The transaction conversion from Snoopable to Non-snoopable, using the **BROADCASTINNER** and **BROADCASTOUTER** signals, the following conversions apply:

- All Read transactions must be converted to ReadNoSnp.
- All Snoopable CMO transactions must be converted to Non-snoopable CMO.
- The following Dataless transactions must be dropped:
  - CleanUnique
  - MakeUnique
  - Evict
  - StashOnce
  - StashOnce*Unique
  - StashOnce*Shared
- All Combined Writes must be converted to Combined WriteNoSnp.
- All Write transactions, except WriteEvictFull and WriteEvictOrEvict, must be converted to WriteNoSnp.
- WriteEvictFull and WriteEvictOrEvict transactions must be dropped.
- All Snoopable Atomic transactions must be converted to Non-snoopable.
- PrefetchTgt transaction conversion is not required.

B16.2.2 BROADCASTCACHEMAINT

The **BROADCASTCACHEMAINT** signal is used to control the issuing of Cache Maintenance Operations when there are software-managed caches downstream of the interface.

When the **BROADCASTCACHEMAINT**, **BROADCASTINNER**, and **BROADCASTOUTER** signals are all present and deasserted:

- CleanShared, CleanInvalid, and MakeInvalid transactions are not issued.

- Combined Write transactions that combine the following Cache Maintenance Operations with Write transactions must be converted to standalone Write transactions:
  - CleanShared
  - CleanInvalid
  - MakeInvalid

B16.2.3 BROADCASTPERSIST

The **BROADCASTPERSIST** signal is used to control the issuing of CleanSharedPersist and CleanSharedPersistSep Cache Maintenance Operations.

When the **BROADCASTPERSIST** signal is present and deasserted, CleanSharedPersist and CleanSharedPersistSep transactions are converted to CleanShared.The conversion applies to both standalone CMO and Combined Write transactions.

Note

The issuing of the CleanShared is controlled by the **BROADCASTINNER**, **BROADCASTOUTER**, and **BROADCASTCACHEMAINT** signals.

B16.2.4 BROADCASTATOMIC

The **BROADCASTATOMIC** signal is used to control the generation of Atomic transactions:

- When asserted, the interface is permitted, but not required, to generate Atomic transactions.
- When deasserted, the interface must not generate Atomic transactions.

A Request Node is not required to make use of Atomic transactions. A Request Node that does not make use of Atomic transactions itself, needs no added functionality to be compatible with an interconnect that supports Atomic transactions.

A Request Node that supports atomic operations but does not include support for the execution of atomic operations must be able to send Atomic transactions.

See B16.3 *Atomic transaction support*.

B16.2.5 BROADCASTICINVAL

The **BROADCASTICINVAL** signal at each Request Node is used to inform the Request Node that broadcasting of *Instruction Cache* (ICache) invalidations using the DVM mechanism is required:

- When asserted, DVMOp for ICache Invalidations must be sent to the interconnect.
- When deasserted, DVMOp for ICache invalidations are not required to be sent to the interconnect.

In a system where all Instruction Caches are fully coherent the hardware coherency mechanism automatically invalidates all ICache copies on a cache line update, In such systems, it is not necessary to broadcast ICache invalidation operations.

If a system contains one or more Instruction Caches that are not updated by the hardware coherency mechanism, it is necessary for ICache invalidation operations to be broadcast using DVM transactions.

B16.2.6 BROADCASTMTE

The **BROADCASTMTE** signal is used to control the issuing of requests with MTE beyond the interface:

- When asserted, requests with MTE can be sent beyond the interface.
- When deasserted, requests with MTE must not be sent beyond the interface.

The **BROADCASTMTE** signal is typically deasserted when the interface does not support MTE functionality.

When the **BROADCASTMTE** signal is deasserted, all other MTE-related interface pins must be tied to 0. The interconnect is permitted, but not required, to remove the related MTE transport wires.

The interface fields that can be fixed to a value of 0 are:

- On DAT channels:
  - TagOp, Tag, and TU
- On REQ and RSP channels:
  - TagOp

B16.2.7 BROADCASTERTLBIINNER and BROADCASTTLBIOUTER

The **BROADCASTTLBIINNER** (BTI) and **BROADCASTTLBIOUTER** (BTO) signals are used to control the issuing of TLBI operations in the interconnect.

Table B16.19 shows the permitted BTI and BTO signal encodings.

Table B16.19: BTI and BTO signal encodings

BTI	BTO	Permitted
0	0	Yes
0	1	Yes
1	0	Reserved
1	1	Yes

Note

The decision to broadcast a DVM(Sync) does not only depend on the values of BTI and BTO but must also consider if any DVM operations other than TLBI must be pushed to their completion by the DVM(Sync).

B16.2.8 BROADCASTCMOPOPA

The **BROADCASTCMOPOPA** signal is used to control the issuing of CleanInvalidPoPA Cache Maintenance Operations. The **BROADCASTCMOPOPA** signal is optional.

When the **BROADCASTCMOPOPA** signal is present and deasserted, a CleanInvalidPoPA transaction is converted to CleanInvalid. This conversion applies to both standalone PoPA Cache Maintenance Operations and Combined Write transactions.

Note

The issuing of CleanInvalid transactions is further controlled by **BROADCASTINNER**, **BROADCASTOUTER**, and **BROADCASTCACHEMAINT** signals.

B16.3 Atomic transaction support

The CHI component support requirements for Atomic transactions are described in the following sections:

- B16.3.1 *Request Node support*
- B16.3.2 *Interconnect support*
- B16.3.3 *Subordinate Node support*

B16.3.1 Request Node support

A Requester component is required to support a mechanism to suppress the generation of Atomic transactions to ensure compatibility in systems where Atomic transactions are not supported. A Requester can use the optional interface pin **BROADCASTATOMIC** to determine whether Atomic transactions are transmitted.

A Request Node is not required to make use of Atomic transactions. A Request Node that does not use Atomic transactions itself needs no added functionality to be compatible with an interconnect that supports Atomic transactions.

A Request Node that supports atomic operations but does not include support for the execution of atomic operations must be able to send Atomic transactions.

For a Request Node that supports both the execution of atomic operations as well as the sending of Atomic transactions the following applies:

For cacheable locations, both Snoopable and Non-snoopable, a Request Node is able to perform an atomic operation locally without generating an Atomic transaction at its interface. To achieve this, the Requester obtains a copy of the location in its local cache, in the same manner that the Requester would for a store operation, and subsequently performs the atomic operation within its local cache. For cacheable locations that are Snoopable, if the contents of the cache line are updated and the cache line was not previously Dirty, the cache line must be marked as Dirty.

B16.3.2 Interconnect support

Interconnect support for Atomic transactions is optional.

The Atomic_Transactions property is used to indicate that an interconnect supports Atomic transactions.

If Atomic transactions are not supported by the interconnect, all attached Request Nodes must be configured to not generate Atomic transactions. The **BROADCASTATOMIC** pin can be used for this purpose, when implemented. See B16.3.1 *Request Node support*.

For interconnects that support Atomic transactions, atomic operation execution can be supported at any point within an interconnect, including passing an Atomic transaction downstream to a Subordinate Node.

Atomic transactions are not required to be supported for every address location.

If Atomic transactions are supported for a given Snoopable address location, they must be supported for the complete Snoopable address range.

If Atomic transactions are not supported for a given address location, an appropriate Error response can be given for the Atomic transaction. See B9.1.4.4 *Atomic transactions*.

For transactions to a Device, the Atomic transaction must be passed to the appropriate endpoint Subordinate. If the Subordinate is configured to not support Atomic transactions, the interconnect must return an Error response for the transaction.

For Non-snoopable transactions, the Atomic transaction must be performed either:

- At a point, or past a point, where the transaction is visible to all other agents.

- At the endpoint.

For Snoopable transactions, the interconnect can either:

- Perform the atomic operation required by an Atomic transaction within the interconnect. This requires that the interconnect performs the appropriate Read, Write, and Snoop transactions to complete the Atomic transaction.
- If the appropriate endpoint Subordinate is configured to indicate support of Atomic transactions, the interconnect can pass the Atomic transaction to the Subordinate. The interconnect is still required to perform the appropriate Snoop and Write transactions before issuing the Atomic transaction to the Subordinate.

B16.3.3 Subordinate Node support

Subordinate Node support for Atomic transactions is optional.

The Atomic_Transactions property is used to indicate that a Subordinate Node supports Atomic transactions.

If a Subordinate Node supports Atomic transactions for particular memory types, or for particular address regions, when receiving an unsupported Atomic transaction, the Subordinate Node must give an appropriate Error response.

Part C
Appendicies

Chapter C1
Message Field Mappings

This appendix shows the field mappings for the request, response, data, and snoop request messages. It contains the following sections:

Table C1.1 shows the conventions used in the field mapping tables.

Table C1.1: Key to field mapping table conventions

Symbol	Description
CF	Common Field. Two or more protocol message fields share the same set of bits in this packet field.
X	Inapplicable. Field value can take any value.
1	Applicable. Field value is used, must be set to 1.
0	Applicable. Field value is used, must be set to 0.
0^a	Inapplicable. Field value must be set to 0.
Y	Applicable. Field value is used. See specification for permitted values and usage.
D	Inapplicable. Field value must be default setting for MPAM fields.
8B	Size field must be set to 8-byte encoding.
64B	Size field must be set to 64-byte encoding.
M	Field position reused for DVM operations. See (DVM Operations)
–	Assigned to another protocol message field that shares the same set of bits in this packet field. If the other field is not present, then it is considered inapplicable and the field value must be set to 0.

C1.1 Request message field mappings

C1.1.1 Read, Dataless and Miscellaneous

Table C1.2 and Table C1.3 show the Read, Dataless and Miscellaneous Request message field mappings. See Table C1.1 for the conventions used in the field mappings. For further information on field use see B13.10 *Protocol flit fields*.

Table C1.2: Read, Dataless and Miscellaneous Request message field mappings Part 1

Request message	QoS	TgtID	SrcID	TxnID	Opcode	AllowRetry	PCrdType	RSVDC	TagOp	TraceTag	MPAM	PBHA	Addr	NSE	NS	Size	Order	LikelyShared	ExpCompAck
ReqLCrdReturn	X	X	X	0	Y	X	X	X	X	X	X	X	X	X	X	X	X	X	X
PCrdReturn	Y	Y	Y	0^a	Y	0^a	Y	Y	0^a	Y	0^a	0^a	0^a	0^a	0^a	0^a	0^a	0^a	0
DVMOp	Y	Y	Y	Y	Y	Y	Y	Y	0^a	Y	0^a	0^a	M	0^a	0^a	8B	0^a	0^a	0
PrefetchTgt	Y	Y	Y	X	Y	0	X	Y	Y	Y	Y	Y	Y	Y	Y	X	X	X	0^a
ReadNoSnp	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	Y
ReadNoSnpSep	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	0
ReadOnce	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	Y
ReadOnceCleanInvalid	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	Y
ReadOnceMakeInvalid	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	Y
ReadClean	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	Y	1
ReadNotSharedDirty	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	Y	1
ReadShared	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	Y	1
ReadUnique	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	Y	1
ReadPreferUnique	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	Y	1
MakeReadUnique	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	Y	1
CleanShared	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	64B	0^a	0	0
CleanSharedPersist	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	64B	0^a	0	0
CleanSharedPersistSep	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	64B	0^a	0	0
CleanInvalid	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	64B	0^a	0	0
CleanInvalidPoPA	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	64B	0^a	0	0
MakeInvalid	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	64B	0^a	0	0
CleanUnique	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	64B	0	0	1
MakeUnique	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	0	1
Evict	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	64B	0	0	0

Table C1.3: Read, Dataless and Miscellaneous Request message field mappings Part 2

	MemAttr				CF		CF			CF				CF			CF			CF		
Request message	Allocate	Cacheable	Device	EWA	SnpAttr	DoDWT	Excl	SnoopMe	CAH	LPID	TagGroupID	StashGroupID	PGroupID	ReturnNID	StashNID	SLCRepHint	StashNIDValid	Endian	Deep	ReturnTxnID	StashLPIDValid	StashLPID
ReqLCrdReturn	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
PCrdReturn	0[a]	0[a]	0[a]	0[a]	0[a]	–	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
DVMOp	0[a]	0[a]	0[a]	0[a]	M	–	0[a]	0[a]	0[a]	Y	–	–	–	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
PrefetchTgt	X	X	X	X	X	X	X	–	–	Y	–	–	–	0[a]	0[a]	0[a]	X	X	X	0[a]	0[a]	0[a]
ReadNoSnp	Y	Y	Y	Y	0	–	Y	–	–	Y	–	–	–	Y	–	Y	0[a]	0[a]	0[a]	Y	–	–
ReadNoSnpSep	Y	Y	Y	Y	0	–	0	–	–	Y	–	–	–	Y	–	–	0[a]	0[a]	0[a]	Y	–	–
ReadOnce	Y	1	0	1	1	–	0	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
ReadOnceCleanInvalid	Y	1	0	1	1	–	0	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
ReadOnceMakeInvalid	0	1	0	1	1	–	0	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
ReadClean	Y	1	0	1	1	–	Y	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
ReadNotSharedDirty	Y	1	0	1	1	–	Y	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
ReadShared	Y	1	0	1	1	–	Y	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
ReadUnique	Y	1	0	1	1	–	0	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
ReadPreferUnique	Y	1	0	1	1	–	Y	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
MakeReadUnique	Y	1	0	1	1	–	Y	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
CleanShared	Y	Y	Y	Y	Y	–	0	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
CleanSharedPersist	Y	Y	Y	Y	Y	–	0	–	–	0[a]	0[a]	0[a]	0[a]	–	–	Y	–	–	Y	0[a]	0[a]	0[a]
CleanSharedPersistSep	Y	Y	Y	Y	Y	–	0	–	–	–	–	–	Y	Y	–	Y	–	–	Y	0[a]	0[a]	0[a]
CleanInvalid	Y	Y	Y	Y	Y	–	0	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
CleanInvalidPoPA	Y	Y	Y	Y	Y	–	0	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
MakeInvalid	Y	Y	Y	Y	Y	–	0	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
CleanUnique	Y	1	0	1	1	–	Y	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
MakeUnique	Y	1	0	1	1	–	0	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
Evict	0	1	0	1	1	–	0	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]

C1.1.2 Write and Combined Write

Table C1.4 and Table C1.5 show the Write and Combined Write Request message field mappings. See Table C1.1 for the conventions used in the field mappings. For further information on field use see B13.10 *Protocol flit fields*.

Table C1.4: Write and Combined Write Request message field mappings Part 1

Request message	QoS	TgtID	SrcID	TxnID	Opcode	AllowRetry	PCrdType	RSVDC	TagOp	TraceTag	MPAM	PBHA	Addr	NSE	NS	Size	Order	LikelyShared	ExpCompAck
WriteNoSnpPtl	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	Y
WriteNoSnpPtlCleanInv	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	Y
WriteNoSnpPtlCleanInvPoPA	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	Y
WriteNoSnpPtlCleanSh	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	Y
WriteNoSnpPtlCleanShPerSep	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	Y
WriteNoSnpFull	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	Y	0	Y
WriteNoSnpFullCleanInv	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	Y	0	Y
WriteNoSnpFullCleanInvPoPA	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	Y	0	Y
WriteNoSnpFullCleanSh	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	Y	0	Y
WriteNoSnpFullCleanShPerSep	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	Y	0	Y
WriteNoSnpDef	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	64B	Y	0	0
WriteNoSnpZero	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	64B	Y	0	0
WriteUniquePtlStash	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
WriteUniqueFullStash	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	Y	Y	Y
WriteUniquePtl	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
WriteUniquePtlCleanSh	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	Y	Y	0	Y
WriteUniquePtlCleanShPerSep	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	Y	Y	0	Y
WriteUniqueFull	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	Y	Y	Y
WriteUniqueFullCleanSh	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	64B	Y	0	Y
WriteUniqueFullCleanShPerSep	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	64B	Y	0	Y
WriteUniqueZero	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	64B	Y	Y	0
WriteBackPtl	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	64B	0	0	0
WriteBackFull	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	Y	0
WriteBackFullCleanInv	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	0	0
WriteBackFullCleanInvPoPA	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	0	0
WriteBackFullCleanSh	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	0	0
WriteBackFullCleanShPerSep	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	0	0
WriteCleanFull	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	Y	0
WriteCleanFullCleanSh	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	0	0
WriteCleanFullCleanShPerSep	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	0	0
WriteEvictFull	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	Y	0
WriteEvictOrEvict	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	Y	1

Table C1.5: Write and Combined Write Request message field mappings Part 2

	MemAttr				CF		CF			CF				CF			CF			CF		
Request message	Allocate	Cacheable	Device	EWA	SnpAttr	DoDWT	Excl	SnoopMe	CAH	LPID	TagGroupID	StashGroupID	PGroupID	ReturnNID	StashNID	SLCRepHint	StashNIDValid	Endian	Deep	ReturnTxnID	StashLPIDValid	StashLPID
WriteNoSnpPtl	Y	Y	Y	Y	0	Y	Y	–	–	Y	Y	–	–	Y	–	Y	0[a]	0[a]	0[a]	Y	–	–
WriteNoSnpPtlCleanInv	Y	Y	Y	Y	0	Y	0	–	–	Y	–	–	–	Y	–	Y	0[a]	0[a]	0[a]	Y	–	–
WriteNoSnpPtlCleanInvPoPA	Y	Y	Y	Y	0	Y	0	–	–	Y	–	–	–	Y	–	Y	0[a]	0[a]	0[a]	Y	–	–
WriteNoSnpPtlCleanSh	Y	Y	Y	Y	0	Y	0	–	–	Y	–	–	–	Y	–	Y	0[a]	0[a]	0[a]	Y	–	–
WriteNoSnpPtlCleanShPerSep	Y	Y	Y	Y	0	Y	0	–	–	–	–	–	Y	Y	–	Y	–	–	Y	Y	–	–
WriteNoSnpFull	Y	Y	Y	Y	0	Y	Y	–	–	Y	Y	–	–	Y	–	Y	0[a]	0[a]	0[a]	Y	–	–
WriteNoSnpFullCleanInv	Y	Y	Y	Y	0	Y	0	–	–	Y	–	–	–	Y	–	Y	0[a]	0[a]	0[a]	Y	–	–
WriteNoSnpFullCleanInvPoPA	Y	Y	Y	Y	0	Y	0	–	–	Y	–	–	–	Y	–	Y	0[a]	0[a]	0[a]	Y	–	–
WriteNoSnpFullCleanSh	Y	Y	Y	Y	0	Y	0	–	–	Y	–	–	–	Y	–	Y	0[a]	0[a]	0[a]	Y	–	–
WriteNoSnpFullCleanShPerSep	Y	Y	Y	Y	0	Y	0	–	–	–	–	–	Y	Y	–	Y	–	–	Y	Y	–	–
WriteNoSnpDef	0	0	Y	Y	0	Y	0	–	–	Y	–	–	–	Y	–	Y	0[a]	0[a]	0[a]	Y	–	–
WriteNoSnpZero	Y	Y	Y	Y	0	0[a]	0	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
WriteUniquePtlStash	Y	1	0	1	1	–	0	–	–	Y	Y	–	–	–	Y	Y	Y	–	–	–	Y	Y
WriteUniqueFullStash	Y	1	0	1	1	–	0	–	–	Y	Y	–	–	–	Y	Y	Y	–	–	–	Y	Y
WriteUniquePtl	Y	1	0	1	1	–	0	–	–	Y	Y	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
WriteUniquePtlCleanSh	Y	1	0	1	1	–	0	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
WriteUniquePtlCleanShPerSep	Y	1	0	1	1	–	0	–	–	–	–	–	Y	–	–	Y	–	–	Y	0[a]	0[a]	0[a]
WriteUniqueFull	Y	1	0	1	1	–	0	–	–	Y	Y	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
WriteUniqueFullCleanSh	Y	1	0	1	1	–	0	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
WriteUniqueFullCleanShPerSep	Y	1	0	1	1	–	0	–	–	–	–	–	Y	–	–	Y	–	–	Y	0[a]	0[a]	0[a]
WriteUniqueZero	Y	1	0	1	1	–	0	–	–	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
WriteBackPtl	Y	1	0	1	1	–	–	–	0	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
WriteBackFull	Y	1	0	1	1	–	–	–	Y	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
WriteBackFullCleanInv	Y	1	0	1	1	–	–	–	Y	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
WriteBackFullCleanInvPoPA	Y	1	0	1	1	–	–	–	Y	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
WriteBackFullCleanSh	Y	1	0	1	1	–	–	–	Y	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
WriteBackFullCleanShPerSep	Y	1	0	1	1	–	–	–	Y	–	–	–	Y	–	–	Y	–	–	Y	0[a]	0[a]	0[a]
WriteCleanFull	Y	1	0	1	1	–	–	–	Y	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
WriteCleanFullCleanSh	Y	1	0	1	1	–	–	–	Y	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
WriteCleanFullCleanShPerSep	Y	1	0	1	1	–	–	–	Y	–	–	–	Y	–	–	Y	–	–	Y	0[a]	0[a]	0[a]
WriteEvictFull	1	1	0	1	1	–	–	–	Y	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
WriteEvictOrEvict	1	1	0	1	1	–	–	–	Y	Y	–	–	–	–	–	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]

C1.1.3 Stash and Atomic

Table C1.6 and Table C1.7 show the Stash and Atomic Request message field mappings. See Table C1.1 for the conventions used in the field mappings. For further information on field use see B13.10 *Protocol flit fields*.

Table C1.6: Stash and Atomic Request message field mappings part 1

Request message	QoS	TgtID	SrcID	TxnID	Opcode	AllowRetry	PCrdType	RSVDC	TagOp	TraceTag	MPAM	PBHA	Addr	NSE	NS	Size	Order	LikelyShared	ExpCompAck
StashOnceUnique	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	Y	0
StashOnceSepUnique	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	Y	0
StashOnceShared	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	Y	0
StashOnceSepShared	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	64B	0	Y	0
AtomicLoad	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	0
AtomicStore	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	0
AtomicCompare	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	0
AtomicSwap	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	0

Table C1.7: Stash and Atomic Request message field mappings part 2

	MemAttr				CF		CF			CF				CF			CF			CF		
Request message	Allocate	Cacheable	Device	EWA	SnpAttr	DoDWT	Excl	SnoopMe	CAH	LPID	TagGroupID	StashGroupID	PGroupID	ReturnNID	StashNID	SLCRepHint	StashNIDValid	Endian	Deep	ReturnTxnID	StashLPIDValid	StashLPID
StashOnceUnique	Y	1	0	1	1	–	0	–	–	Y	–	–	–	–	Y	Y	Y	–	–	–	Y	Y
StashOnceSepUnique	Y	1	0	1	1	–	0	–	–	–	–	Y	–	–	Y	Y	Y	–	–	–	Y	Y
StashOnceShared	Y	1	0	1	1	–	0	–	–	Y	–	–	–	–	Y	Y	Y	–	–	–	Y	Y
StashOnceSepShared	Y	1	0	1	1	–	0	–	–	–	–	Y	–	–	Y	Y	Y	–	–	–	Y	Y
AtomicLoad	Y	Y	Y	Y	Y	–	–	Y	–	Y	Y	–	–	Y	–	–	–	Y	–	Y	–	–
AtomicStore	Y	Y	Y	Y	Y	–	–	Y	–	Y	Y	–	–	Y	–	–	–	Y	–	X	–	–
AtomicCompare	Y	Y	Y	Y	Y	–	–	Y	–	Y	Y	–	–	Y	–	–	–	Y	–	Y	–	–
AtomicSwap	Y	Y	Y	Y	Y	–	–	Y	–	Y	Y	–	–	Y	–	–	–	Y	–	Y	–	–

C1.2 Response message field mappings

Table C1.8 shows the Response message field mappings. See Table C1.1 for the conventions used in the field mappings. For further information on field use see B13.10 *Protocol flit fields*.

Table C1.8: Response message field mappings

Response message	QoS	TgtID	SrcID	TxnID	Opcode	RespErr	Resp	CBusy	TagOp	TraceTag	PCrdType	CF				CF	
												DBID	TagGroupID	StashGroupID	PGroupID	FwdState	DataPull
RspLCrdReturn	X	X	X	0	Y	X	X	X	X	X	X	X	X	X	X	X	X
SnpResp	Y	Y	Y	Y	Y	Y	Y	Y	0[a]	Y	0[a]	Y	–	–	–	–	Y
SnpRespFwded	Y	Y	Y	Y	Y	Y	Y	Y	0[a]	Y	0[a]	X	–	–	–	Y	–
CompAck	Y	Y	Y	Y	Y	0	Y	0[a]	0[a]	Y	0[a]	X	–	–	–	0[a]	0[a]
RetryAck	Y	Y	Y	Y	Y	0	0[a]	Y	0[a]	Y	Y	X	–	–	–	0[a]	0[a]
Comp	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0[a]	Y	–	–	–	0[a]	0[a]
CompCMO	Y	Y	Y	Y	Y	Y	Y	Y	0[a]	Y	0[a]	X	–	–	–	0[a]	0[a]
Persist	Y	Y	Y	0[a]	Y	Y	0[a]	Y	0[a]	X	0[a]	–	–	–	Y	0[a]	0[a]
CompPersist	Y	Y	Y	Y	Y	Y	Y	Y	0[a]	Y	0[a]	–	–	–	Y	0[a]	0[a]
StashDone	Y	Y	Y	0[a]	Y	Y	0[a]	Y	0[a]	X	0[a]	–	–	Y	–	0[a]	0[a]
CompStashDone	Y	Y	Y	Y	Y	Y	Y	Y	0[a]	Y	0[a]	–	–	Y	–	0[a]	0[a]
RespSepData	Y	Y	Y	Y	Y	Y	Y	Y	0[a]	Y	0[a]	Y	–	–	–	0[a]	0[a]
CompDBIDResp	Y	Y	Y	Y	Y	Y	Y	Y	0[a]	Y	0[a]	Y	–	–	–	0[a]	0[a]
DBIDResp	Y	Y	Y	Y	Y	0	0[a]	Y	0[a]	Y	0[a]	Y	–	–	–	0[a]	0[a]
DBIDRespOrd	Y	Y	Y	Y	Y	0	0[a]	Y	0[a]	Y	0[a]	Y	–	–	–	0[a]	0[a]
TagMatch	Y	Y	Y	0[a]	Y	Y	Y	Y	0[a]	X	0[a]	–	Y	–	–	0[a]	0[a]
PCrdGrant	Y	Y	Y	0[a]	Y	0	0[a]	Y	0[a]	Y	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
ReadReceipt	Y	Y	Y	Y	Y	0	0[a]	Y	0[a]	Y	0[a]	X	–	–	–	0[a]	0[a]

C1.3 Snoop Request message field mappings

Table C1.9shows the snoop request message field mappings. See Table C1.1 for the conventions used in the field mappings. For further information on field use see B13.10 *Protocol flit fields*.

Table C1.9: Snoop Request message field mappings

Snoop Request message	QoS	SrcID	TxnID	Opcode	Addr	NSE	NS	DoNotGoToSD	RetToSrc	TraceTag	MPAM	CF		CF			
												FwdNID	PBHA	FwdTxnID	StashLPIDValid	StashLPID	VMIDExt
SnpLCrdReturn	X	X	0	Y	X	X	X	X	X	X	X	X	X	X	X	X	X
SnpShared	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	D	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
SnpClean	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	D	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
SnpOnce	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	D	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
SnpNotSharedDirty	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	D	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
SnpUnique	Y	Y	Y	Y	Y	Y	Y	1	Y	Y	D	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
SnpPreferUnique	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	D	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
SnpCleanShared	Y	Y	Y	Y	Y	Y	Y	1	0[a]	Y	D	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
SnpCleanInvalid	Y	Y	Y	Y	Y	Y	Y	1	0[a]	Y	D	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
SnpMakeInvalid	Y	Y	Y	Y	Y	Y	Y	1	0[a]	Y	D	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
SnpSharedFwd	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	D	Y	–	Y	–	–	–
SnpCleanFwd	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	D	Y	–	Y	–	–	–
SnpOnceFwd	Y	Y	Y	Y	Y	Y	Y	Y	0[a]	Y	D	Y	–	Y	–	–	–
SnpNotSharedDirtyFwd	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	D	Y	–	Y	–	–	–
SnpUniqueFwd	Y	Y	Y	Y	Y	Y	Y	1	0[a]	Y	D	Y	–	Y	–	–	–
SnpPreferUniqueFwd	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	D	Y	–	Y	–	–	–
SnpUniqueStash	Y	Y	Y	Y	Y	Y	Y	1	0[a]	Y	Y	–	Y	–	Y	Y	–
SnpMakeInvalidStash	Y	Y	Y	Y	Y	Y	Y	1	0[a]	Y	Y	–	Y	–	Y	Y	–
SnpStashUnique	Y	Y	Y	Y	Y	Y	Y	1	0[a]	Y	Y	–	Y	–	Y	Y	–
SnpStashShared	Y	Y	Y	Y	Y	Y	Y	1	0[a]	Y	Y	–	Y	–	Y	Y	–
SnpQuery	Y	Y	Y	Y	Y	Y	Y	0[a]	0[a]	Y	D	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]
SnpDVMOp	Y	Y	Y	Y	M	0[a]	0[a]	0[a]	0[a]	Y	0[a]	M	M	–	–	–	Y

C1.4 Data message field mappings

Table C1.10 amd Table C1.11 show the Data message field mappings. See Table C1.1 for the conventions used in the field mappings. For further information on field use see B13.10 *Protocol flit fields*.

Table C1.10: Data message field mappings part 1

Data message	QoS	TgtID	SrcID	TxnID	Opcode	RespErr	Resp	CBusy	DBID	CCID	DataID	RSVDC	BE	Data	TraceTag	CAH	DataCheck	Poison	TagOp	Tag	TU
DatLCrdReturn	X	X	X	0	Y	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
SnpRespData	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SnpRespDataFwded	Y	Y	Y	Y	Y	Y	Y	Y	X	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
CopyBackWriteData	Y	Y	Y	Y	Y	Y	Y	0[a]	X	Y	Y	Y	Y	Y	Y	0[a]	Y	Y	Y	Y	Y
NonCopyBackWriteData	Y	Y	Y	Y	Y	Y	0	0[a]	X	Y	Y	Y	Y	Y	Y	0[a]	Y	Y	Y	Y	Y
NonCopyBackWriteDataCompAck	Y	Y	Y	Y	Y	Y	0	0[a]	X	Y	Y	Y	Y	Y	Y	0[a]	Y	Y	Y	Y	Y
CompData	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	X	Y	Y	Y	Y	Y	Y	Y	Y
DataSepResp	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	X	Y	Y	Y	Y	Y	Y	Y	Y
SnpRespDataPtl	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y
WriteDataCancel	Y	Y	Y	Y	Y	Y	0	0[a]	X	Y	Y	Y	0	0	Y	0[a]	Y	Y	Y	Y	Y

Table C1.11: Data message field mappings part 2

	CF		CF		
Data message	HomeNID	PBHA	FwdState	DataPull	DataSource
DatLCrdReturn	X	X	X	X	X
SnpRespData	–	Y	–	Y	Y
SnpRespDataFwded	–	Y	Y	–	–
CopyBackWriteData	0[a]	0[a]	0[a]	0[a]	0[a]
NonCopyBackWriteData	0[a]	0[a]	0[a]	0[a]	0[a]
NonCopyBackWriteDataCompAck	0[a]	0[a]	0[a]	0[a]	0[a]
CompData	Y	–	–	–	Y
DataSepResp	Y	–	–	–	Y
SnpRespDataPtl	–	Y	–	Y	Y
WriteDataCancel	0[a]	0[a]	0[a]	0[a]	0[a]

Chapter C2
Communicating Nodes

This appendix specifies, for each packet type, the nodes that communicate using that packet type. It contains the following sections:

C2.1 Request communicating nodes

Table C2.1 shows the Request communicating nodes. In Table C2.1, unless explicitly stated otherwise, a reference to a Write transaction includes both the individual Write transaction and the corresponding Combined Write transaction.

For some Requests, both an expected target and a permitted target are given. The use of the permitted target can occur in the case of a software based error. The permitted target must complete the transaction in a protocol compliant manner, this might require the use of an error response.

Table C2.1: Request communicating nodes

Request	From	To	
		Expected	Permitted
ReadNoSnp	RN-F, RN-D, RN-I	ICN(HN-F, HN-I)	-
WriteNoSnpPtl	ICN(HN-F)	SN-F	-
WriteNoSnpFull	ICN(HN-I)	SN-I	-
WriteNoSnpZero			
WriteNoSnpDef	RN-F	HN-I	HN-F
	RN-D, RN-I[a]		
	ICN(HN-I)	SN-I	-
ReadNoSnpSep	ICN(HN-F)	SN-F	-
	ICN(HN-I)	SN-I	-
ReadClean	RN-F	ICN(HN-F)	ICN(HN-I)
ReadShared			
ReadNotSharedDirty			
ReadUnique			
ReadPreferUnique			
MakeReadUnique			
CleanUnique			
MakeUnique			
Evict			
WriteBackPtl			
WriteBackFull			
WriteCleanFull			
WriteEvictFull			
WriteEvictOrEvict			
ReadOnce	RN-F, RN-D, RN-I	ICN(HN-F)	ICN(HN-I)
ReadOnceCleanInvalid			

Continued on next page

Table C2.1 – *Continued from previous page*

Request	From	To	
		Expected	Permitted
ReadOnceMakeInvalid			
StashOnceUnique			
StashOnceShared			
StashOnceSepUnique			
StashOnceSepShared			
WriteUniqueFull			
WriteUniqueFullStash			
WriteUniquePtl			
WriteUniquePtlStash			
WriteUniqueZero			
CleanShared	RN-F, RN-D, RN-I	ICN(HN-F, HN-I)	-
CleanSharedPersist	ICN(HN-F)	SN-F	-
CleanSharedPersistSep	ICN(HN-I)	SN-I	-
CleanInvalid			
CleanInvalidPoPA			
MakeInvalid			
WriteUniquePtlCleanSh	RN-F, RN-D, RN-I	ICN(HN-F)	ICN(HN-I)
WriteUniquePtlCleanShPerSep			
WriteUniqueFullCleanSh			
WriteUniqueFullCleanShPerSep			
WriteBackFullCleanInv	RN-F	ICN(HN-F)	ICN(HN-I)
WriteBackFullCleanInvPoPA			
WriteBackFullCleanSh			
WriteBackFullCleanShPerSep			
WriteCleanFullCleanSh			
WriteCleanFullCleanShPerSep			
WriteNoSnpFullCleanInv	RN-F, RN-D, RN-I	ICN(HN-F, HN-I)	-
WriteNoSnpFullCleanInvPoPA	ICN(HN-F)	SN-F	-
WriteNoSnpFullCleanSh	ICN(HN-I)	SN-I	-
WriteNoSnpFullCleanShPerSep			
WriteNoSnpPtlCleanInv			
WriteNoSnpPtlCleanInvPoPA			
WriteNoSnpPtlCleanSh			

Continued on next page

Table C2.1 – *Continued from previous page*

Request	From	To	
		Expected	**Permitted**
WriteNoSnpPtlCleanShPerSep			
DVMOp	RN-F	ICN(MN)	-
PCrdReturn	RN-F, RN-D, RN-I	ICN(HN-F, HN-I, MN)	-
	ICN(HN-F)	SN-F	-
	ICN(HN-I)	SN-I	-
AtomicStore	RN-F, RN-D, RN-I	ICN(HN-F, HN-I)	-
AtomicLoad	ICN(HN-F)	SN-F	-
AtomicSwap	ICN(HN-I)	SN-I	-
AtomicCompare			
PrefetchTgt	RN-F, RN-D, RN-I	SN-F	-

[a] The request is permitted, but not expected, from RN-D or RN-I.

C2.2 Snoop communicating nodes

Table C2.2 shows the Snoop communicating nodes.

Table C2.2: Snoop communicating nodes

Snoop	From	To
SnpShared	ICN(HN-F)	RN-F
SnpClean		
SnpOnce		
SnpNotSharedDirty		
SnpUnique		
SnpPreferUnique		
SnpCleanShared		
SnpCleanInvalid		
SnpMakeInvalid		
SnpSharedFwd		
SnpCleanFwd		
SnpOnceFwd		
SnpNotSharedDirtyFwd		
SnpUniqueFwd		
SnpPreferUniqueFwd		
SnpUniqueStash		
SnpMakeInvalidStash		
SnpStashUnique		
SnpStashShared		
SnpQuery		
SnpDVMOp	ICN(MN)	RN-F, RN-D

C2.3 Response communicating nodes

Table C2.3 shows the Response communicating nodes.

Table C2.3: Response communicating nodes

Response		From	To
Upstream	RetryAck	ICN(HN-F, HN-I, MN)	RN-F, RN-D, RN-I
	PCrdGrant	SN-F	ICN(HN-F)
	Comp CompDBIDResp	SN-I	ICN(HN-I)
	CompCMO	ICN(HN-F, HN-I)	RN-F, RN-D, RN-I
	ReadReceipt	SN-F	ICN(HN-F)
		SN-I	ICN(HN-I)
	RespSepData	ICN(HN-F, HN-I)	RN-F, RN-D, RN-I
	DBIDResp	ICN(HN-F, HN-I, MN)	RN-F, RN-D, RN-I
		SN-F	ICN(HN-F), RN-F, RN-D, RN-I
		SN-I	ICN(HN-I), RN-F, RN-D, RN-I
	DBIDRespOrd	ICN(HN-F, HN-I)	RN-F, RN-D, RN-I
	StashDone CompStashDone	ICN(HN-F)	RN-F, RN-D, RN-I
	TagMatch	ICN(HN-F)	RN-F, RN-D, RN-I
		SN-F	ICN(HN-F), RN-F, RN-D, RN-I
	Persist	ICN(HN-F, HN-I)	RN-F, RN-D, RN-I
		SN-F	ICN(HN-F), RN-F, RN-D, RN-I
		SN-I	ICN(HN-I), RN-F, RN-D, RN-I
	CompPersist	ICN(HN-F, HN-I)	RN-F, RN-D, RN-I
		SN-F	ICN(HN-F)
		SN-I	ICN(HN-I)
Downstream	CompAck	RN-F, RN-D, RN-I	ICN(HN-F, HN-I)
	SnpResp	RN-F	ICN(HN-F)
		RN-F, RN-D	ICN(MN)
	SnpRespFwded	RN-F	ICN(HN-F)

C2.4 Data communicating nodes

Table C2.4 shows the Data communicating nodes.

For some Data, both an expected target and a permitted target are given. The use of the permitted target can occur in the case of an incorrect address decode. The permitted target must complete the transaction in a protocol compliant manner. In Table C2.4, unless explicitly stated otherwise, a reference to a Write transaction includes both the individual Write transaction and the corresponding Combined Write transaction.

Table C2.4: Data communicating nodes

Data		From	To	
			Expected	Permitted
Upstream	CompData	ICN(HN-F, HN-I)	RN-F, RN-D, RN-I	-
		SN-F	RN-F, RN-D, RN-I, ICN(HN-F)	-
		SN-I	RN-F, RN-D, RN-I, ICN(HN-I)	-
	DataSepResp	ICN(HN-F, HN-I)	RN-F, RN-D, RN-I	-
		SN-F	RN-F, RN-D, RN-I	ICN(HN-F)
		SN-I	RN-F, RN-D, RN-I	ICN(HN-I)
Downstream	CopyBackWriteData	RN-F	ICN(HN-F)	ICN(HN-I)
	WriteDataCancel	RN-F, RN-D, RN-I	ICN(HN-F, HN-I), SN-F, SN-I	-
		ICN(HN-F)	SN-F	-
		ICN(HN-I)	SN-I	-
	NonCopyBackWriteData	RN-F, RN-D, RN-I	ICN(HN-F, HN-I), SN-F, SN-I	-
		RN-F, RN-D	ICN(MN)	-
		ICN(HN-F)	SN-F	-
		ICN(HN-I)	SN-I	-
	NCBWrDataCompAck	RN-F, RN-D, RN-I	ICN(HN-F, HN-I)	-
	SnpRespData SnpRespDataFwded SnpRespDataPtl	RN-F	ICN(HN-F)	
Peer-to-Peer	CompData	RN-F	RN-F, RN-D, RN-I	-

Chapter C3
Revisions

This appendix describes the technical changes between released issues of this specification.

C3.1 Changes between Issue A and Issue B

Table C3.1: Changes between Issue A and Issue B

Change	Location
No changes, first public release	-

C3.2 Changes between Issue B and Issue C

Table C3.2: Changes between Issue B and Issue C

Change	Location
New feature: Response after receiving first Data packet	B2.3.1.1 *Allocating Read* B2.3.1.2 *Non-allocating Read*
New feature: Separate Non-data and Data-only response	B2.3.1 *Read transactions* and multiple related locations
New feature: Combined CompAck with WriteData	B2.5.3.3 *WriteUnique transaction* and multiple related locations
Clarification: Regarding BE bits for Write transactions	B2.8.3 *Byte Enables*
Update: Concerning the list of fields that can change value from the original request in the retried request	B2.9 *Request Retry*
Clarification: Concerning the transaction responses permitted to be sent to the same address when a Snoop transaction response is pending	B4.11.1 *At the RN-F node*
Additional information: Concerning Legal RespErr field values for WriteDataCancel	Table B9.8
Clarification: Regarding TraceTag field value propagation	B11.7.1 *TraceTag usage and rules*
Corrections and additions: Concerning message field mappings	Table C1.3, Table C1.8, and Table C1.9

C3.3 Changes between Issue C and Issue D

Table C3.3: Changes between Issue C and Issue D

Change	Location
New feature: Persistent CMO with two part response	B2.3.5 *Dataless transactions* B2.5.2 *Dataless transactions*

Continued on next page

Table C3.3 – *Continued from previous page*

Change	Location
	B4.2.2.1 *Cache Maintenance transactions* B5.2.3 *Persistent CMO with snoop and separate Comp and Persist* B16.1 *Interface properties and parameters* B9.1.4.2 *Dataless transactions*
New feature: Deep Persistent cache maintenance	B4.2.2.1 *Cache Maintenance transactions* B13.10.19 *Deep persistence, Deep*
New feature: Interface parity	B9.3 *Use of interface parity* B16.1 *Interface properties and parameters*
New feature: *Memory System Resource Partitioning and Monitoring* (MPAM)	B11.4 *MPAM* B16.1 *Interface properties and parameters*
New feature: Completer Busy	B11.6 *Completer Busy*
New feature: ICache Invalidation broadcast signal	B16.2 *Optional interface broadcast signals*
Additional requirement: Concerning **SACTIVE** synchronization to **CLK**	B14.7 *Protocol layer activity indication*
Additional requirement: Concerning **SYSCOREQ** and **SYSCOACK** synchronization to **CLK**	Chapter B15 *System Coherency Interface*
Correction: Concerning the use of **RXSACTIVE** to directly generate the **TXSACTIVE** signal	B14.7.2 *TXSACTIVE signal*
Update: Concerning Ordered Write Observation flow enhancements	B2.6.5.3 *Streaming Ordered Write transactions*
Update: Concerning the relaxation of the order requirement between Cache Maintenance transactions and any other transaction to the same address	B4.2.2.1 *Cache Maintenance transactions*
Updare: Concerning UD_PD state is permitted on a DataSepResp response	B4.5.1.1 *Read and Atomic transaction completion*
Update: Concerning DVM early Comp	B8.2.1.1 *DVM early Comp for Non-sync DVMOps*
Update: Concerning increased TxnID width	B13.9 *Flit packet definitions*
Clarification: Regarding when a RespSepData response includes a NDERR	B9.1.4.1 *Read transactions*

Continued on next page

Table C3.3 – *Continued from previous page*

Change	Location
Clarification: Regarding when the DataPull bit is set in a SnpRespData message	B13.10.37 *Data Pull, DataPull*

C3.4 Changes between Issue D and Issue E.a

Table C3.4: Changes between Issue D and Issue E.a

Change	Location
New feature: Write with optional Data: WriteEvictOrEvict	B2.3.2.3 *CopyBack Write* B4.2.3.2 *CopyBack transactions* Chapter B9 *Error Handling* Chapter B12 *Memory Tagging*
New feature: Write Zero with no Data: WriteNoSnpZero WriteUniqueZero	B2.3.2.1 *Immediate Write* B2.6.5 *Transaction ordering* B4.2.3 *Write transactions* Chapter B9 *Error Handling* Chapter B12 *Memory Tagging*
New feature: SnpQuery Snoop request	B4.3 *Snoop request types* Chapter B9 *Error Handling*
New feature: DBIDRespOrd response	B2.3.2.1 *Immediate Write* B4.5.4 *Miscellaneous response*
New feature: New transactions to support Exclusive reads ReadPreferUnique SnpPreferUnique SnpPreferUniqueFwd MakeReadUnique MakeReadUnique(Excl)	B2.3.1 *Read transactions* B2.3.4 *Stash transactions* B4.2.1 *Read transactions* B4.3 *Snoop request types* B4.7.1.1 *MakeReadUnique transaction* B6.3 *Exclusive transactions* B6.3.1.1 *MakeReadUnique(Excl)* Chapter B9 *Error Handling* Chapter B12 *Memory Tagging*
New feature: Direct Write-data transfer	B2.3.2 *Write transactions* B13.10.26 *Do Direct Write Transfer, DoDWT*
New feature: Combined Write transaction	B1.4 *Transaction classification*

Continued on next page

Table C3.4 – *Continued from previous page*

Change	Location
CompCMO	B2.3.2.4 *Combined Immediate Write and CMO* B4.2.4 *Combined Write requests*
New feature: Two-part StashOnce transaction including: StashOnceSep requests StashDone response CompStashDone response	B2.3.4 *Stash transactions* B2.5.3.4 *StashOnce or StashOnceSep transaction* B4.2.2 *Dataless transactions* B7.3 *Independent Stash request* Chapter B9 *Error Handling* Chapter B12 *Memory Tagging*
New feature: Forward indication on Snoop forward treated as a hint	B4.4 *Request transactions and corresponding Snoop requests*
New feature: Increasing inter-port bandwidth Multiple interfaces Replicated channels	B13.7 *Increasing inter-port bandwidth*
New feature: Memory tagging	Chapter B12 *Memory Tagging*
New feature: Extending DVM operations: Range-based TLBI Level hint in TLBI operation DVM Domain	Chapter B8 *DVM Operations*
New feature: SLC Replacement Hint	B11.3 *SLC Replacement Hint*
Additional requirement: Concerning Transcation Ordering guarantees	B2.6.5 *Transaction ordering*
Additional requirement: Concerning change in WriteNoSnpFull behavior	B2.3.2 *Write transactions*
Correction: Concerning the Ordering guarantees provided by the Comp and CompData response	B2.6.2 *Completion response and ordering*
Update: Concerning removal of DoNotDataPull attribute on snoops	-
Update: Concerning extending the GroupID field width	See Table C3.5 for further information
Update: Concerning extending the TxnID field width	B13.9 *Flit packet definitions*
Update: Concerning ICache invalidation operations	B8.4.5.2 *Virtual Instruction Cache Invalidate*
Update: Concerning Secure EL2 TLBI operations	B8.4.3 *TLB Invalidate*

Continued on next page

Table C3.4 – *Continued from previous page*

Change	Location
Update: Concerning the Order requirements between transactins with different Order field values[a]	B2.6.5 *Transaction ordering*
Clarification: Regarding Comp and canceled Write	B2.3.2 *Write transactions*
Clarification: Regarding CopyBack Write transaction and RetryAck response	B2.9 *Request Retry*
Clarification: Regarding the SnpAttr and Cacheable field value in a standalone CMO transaction	B4.2.2.1 *Cache Maintenance transactions*
Clarification: Regarding the attributes of Exclusive accesses	B6.2 *Exclusive monitors*
Clarification: Regarding the receiving of WriteData and the sending of the Persist response	B4.2.2.1 *Cache Maintenance transactions*

[a] This Update applies retroactively to CHI Issue D.

C3.5 Changes between Issue E.a and Issue E.b

Table C3.5: Changes between Issue E.a and Issue E.b

Change	Location
Update: All offensive terminology has been replaced per Arm's commitment to progressive terminology	Throughout the specification
Update: Restructured and reformatted Transcation structures section	B2.3 *Transaction structure*
Clarification: SnpAttr bit reuse in DVMOp and inapplicability in PrefetchTgt	Table B2.13
Update: SLCRepHint value is permitted to be different in the resent request	B2.9 *Request Retry*
Clarification: Simultaneous pending of CMO and allocating requests to the same address	B4.2.2.1 *Cache Maintenance transactions*
Correction: Request Order permitted in WriteNoSnpZero from HN-I to SN-I	B4.2.3.1 *Immediate transactions*
Update: Re-write of transactions, grouping common characteristics	B4.2 *Request types*
Clarification: Permitted attribute values for requests, initial cache state at the Requester, and final cache state at the Requester outlined	B4.2 *Request types*
Correction: Order field value `0b00` and `0b01` permitted in ReadNoSnp[a]	B2.3.1 *Read transactions*
Correction: Security field `0b10` in PCI DVM operation expanded to include both Secure and Non-secure	Table B8.5
Clarification: Corrupt data must be marked with Poison, DERR, or NDERR	B9.1.4 *Error response use by transaction type*
Clarification: Poison on MTE is not supported	B9.2.1 *Poison* B12.11.2 *Non-Tag Match errors*
Correction: Legal RespErr field tables corrected	Table B9.7
Correction: Transaction error cannot be indicated in DBIDResp response	Table B9.7
Update: All four combinations of Request NS and MPAMNS field values are legal[b]	B11.4.1 *MPAMSP*
Clarification: MTE fields are inapplicable and must be set to 0 in WriteDataCancel write data response	B12.5.2 *TagOp, TU, and tags relationship*
Update: Redundant GroupIDExt field definition is removed from the specification	B13.10 *Protocol flit fields*
Clarification: Re-write of protocol flit fields	B13.10 *Protocol flit fields*

Continued on next page

Table C3.5 – *Continued from previous page*

Change	Location
Correction: Requirements for DoNotGoToSD in SnpQuery updated	B13.10.38 *Do not transition to SD state, DoNotGoToSD*
Correction: Link deactivation and sending of protocol flits updated. Race conditions section deleted	Table B14.3
Clarification: Restructured and reformatted Broadcast signals	B16.2 *Optional interface broadcast signals*
Correction: TagGroupID field is applicable in WriteNoSnpPtl, WriteNoSnpFull, WriteUniquePtlStash, WriteUniqueFullStash, WriteUniquePtl, and WriteUniqueFull	Table C3.5
Correction: PGroupID field applicable in WriteNoSnpPtl, WriteNoSnpFull, WriteUniquePtl, WriteUniqueFull, WriteCleanFull, and WriteBackFull	Table C3.5
Correction: Deep field in all CleanSharedPersist and WriteCleanShPerSep requests applicable	Table C3.5
Correction: SLCRepHint not applicable for Atomic transactions	Table C1.7
Clarification: UniqueDirtyPartial cache line can have none, some, or all bytes valid	B1.5.2 *Cache state model* and B4.1 *Cache line states*

[a] This correction is applied retrospectively to CHI Issue C, Issue D, and Issue E.a.

[b] This update is applied retrospectively to CHI Issue D and Issue E.a.

C3.6 Changes between Issue E.b and Issue E.c

Table C3.6: Changes between Issue E.b and Issue E.c

Change	Location
Editorial edits	Throughout the specification
Clarification: Cancelation of CopyBack requests following overlapping snoop	B4.11.1 *At the RN-F node* Table B4.28 Table B4.41
Defect: ReadReceipt not optional in certain transaction flows	Figure B2.1 Table B2.6
Clarification: DVM payload encoding for Instruction cache invalidations	Table B8.10
Clarification: Typographical error in example WriteUniqueStash with Data Pull transaction flow	Figure B5.23
Clarification: Requirements for ReadOnceMakeInvalid when invalidating a Dirty copy	B4.2.1 *Read transactions*
Clarification: Requirements for Completer read response when MTE is not supported	B12.11.3 *MTE not supported*
Clarification: Data_Check and Check_Type property descriptions	B9.2.2 *Data Check* B16.1.6 *Data_Check* B16.1.7 *Check_Type*
Clarification: Typographical error in Request Node to Home Node request attribute values table	B4.2.3.3 *Write request attribute values*
Clarification: DBID value in Comp and DBIDResp messages that originate from different sources in DWT flow have no relationship	B2.4.3 *Data Buffer Identifier, DBID*
Clarification: Field consistency requirements for data messages split into multiple packets	B2.8.4 *Data packetization*

C3.7 Changes between Issue E.c and Issue F

Table C3.7: Changes between Issue E.c and Issue F

Change	Location
Update: ReadOnceMakeInvalid return states enhanced to allow UD_PD data repsonses, allowing for SnpUniqueFwd DCT flows	Chapter B4 *Coherence Protocol*
Update: Permitted TagOp values updated for ReadOnceMakeInvalid, ReadOnceCleanInvalid, and MakeReadUnique	Chapter B12 *Memory Tagging*
Update: DataSource field extension to 5 bits	Chapter B11 *System Control, Debug, Trace, and Monitoring* B13.10.57 *Data source, DataSource*
Update: Readonce updated to enable partial cache line read	B4.2.1 *Read transactions*
New feature: Deferrable write	B2.3.2.1 *Immediate Write* B4.5.1.3.1 *WriteNoSnpDef* B2.3.2 *Write transactions*
New feature: CopyAtHome, CAH, bit to help reduce the write data bandwidth for CopyBack Write transactions	B13.10.30 *CopyAtHome, CAH* B2.3.2.3 *CopyBack Write*
New feature: Page-based Hardware Attribute (PBHA)	B11.5 *Page-based Hardware Attributes* B13.10.31 *Page-based Hardware Attribute, PBHA* B16.1.18 *PBHA_Support*
New feature: Realm Management Extension (RME)	B2.7.2 *Physical Address Space, PAS* B16.1.16 *RME_Support* B16.2.8 *BROADCASTCMOPOPA* Chapter B8 *DVM Operations*
New feature: Non-shareable and CMO	B16.1.17 *Nonshareable_Cache_Maint*

C3.8 Changes between Issue F and Issue F.b

Table C3.8: Changes between Issue F and Issue F.b

Change	Location
Defect: MPAM field width has been corrected to 12 bits	B11.4 *MPAM* Table B13.8

Continued on next page

Table C3.8 – *Continued from previous page*

Change	Location
Relaxation: CBusy is allowed to vary within a single DAT packet	B2.8.4 *Data packetization*
Defect: CompAck and NCBWrDataCompAck behavior in OWO Immediate Writes	B2.3.2.1 *Immediate Write* B2.3.2.4 *Combined Immediate Write and CMO* B2.3.2.5 *Combined Immediate Write and Persist CMO*
Clarification: SnpMakeInvalidStash is a permitted snoop for WriteUniquePtlStash	Table B4.25 Table B7.1
Clarification: Permitted cache states for ReadClean	Table B4.4 Table B4.5
Clarification: MakeReadUnique(Excl) responses for Requester in SC state	Table B4.41
Clarification: SnpRespData_I cannot be sent from SC state for SnpUniqueStash	Table B4.50 B13.10.36 *Return to Source, RetToSrc*
Relaxation: Line update rules for CopyBack request with CAH=1	B2.7.8 *CopyAtHome attribute*
Clarification: SnpDVMOp does not need to be sent to original Requester	B8.2.1 *Non-sync type DVM transaction flow*
Clarification: CMO-initiated memory update should use PBHA value from SnpRespData	B11.5.4 *PBHA value consistency*
Defect: Permitted SnpResp.RespErr values in SnpDVMOp	Table B8.2
Clarification: Range field value for GPT TLBI DVMOp transactions	B8.4.3.3 *Invalidation Size in GPT TLBI by PA operations*
Clarification: SLCRepHint not applicable for ReadNoSnpSep	Table C1.3
Clarification: WriteNoSnpZero can be to Device memory	B2.7.3.2.1 *Device memory type*
Clarification: Common field non applicable bit requirements for DataPull updated to reduce gateway complexity related to DataSource[4] modification.	Table B13.9
Clarification: DoNotGoToSD value encoding description is incorrect for SnpQuery and SnpStash*	Table B13.29
Clarification: Comp can be used for WriteBack, WriteClean, and WriteEvictFull	Table B9.7

Continued on next page

Table C3.8 – *Continued from previous page*

Change	Location
Clarification: Regarding interconnect requirements during the Coherency Disconnect state for the System Coherency interface	B15.2 *Handshake*
Clarification: Forwarding snoops cannot request a Snoopee sends data to itself	B4.8.3 *Forwarding Snoop transactions*
Clarification: Permitted Request Node behavior around CopyBack and Atomic transactions when SnoopMe is asserted	B2.6.5.2 *CopyBack Request order*
Enhancement: Updating DVM chapter to align more closely with AXI protocol, including field name changes: – VA Valid to AddrV – VI Valid to VIV – VMID Valid to VMIDV – ASID Valid to ASIDV – DVMOp type to DVMType – Exception Level to Exception – Leaf Entry Invalidation to Leaf – Staged Invalidation to Stage	Chapter B8 *DVM Operations*
Clarification: Effect of **BROADCASTCACHEMAINT** on combined writes	B16.2.2 *BROADCASTCACHEMAINT*
Clarification: When a subsequent transaction can be issued to the same address when DataSepResp and RespSepData are used in the transaction flow.	B2.6.4 *Ordering semantics of RespSepData and DataSepResp*
Clarification: Translation granule size value used in the Range calculation	B8.4.3.1.1 *Range-based payload packing for TLBI by VA and IPA operations*
Clarification: Expectations when two PBHA values for the same location differ	B11.5 *Page-based Hardware Attributes*
Clarification: Atomic Endian field applicable for NonCopyBackWriteData or any CompData packet	B2.8.5.2 *Address and Data alignment*
Clarification: Outcome for Comp in a Combined Write transaction	Table B2.7
Clarification: Silent cache state transition visibility	B4.6 *Silent cache state transitions*

Part D
Glossary

Chapter D1
Glossary

Advanced Microcontroller Bus Architecture, AMBA

The AMBA family of protocol specifications is the Arm open standard for on-chip buses. AMBA provides solutions for the interconnection and management of functional blocks that make up a *System-on-Chip* (SoC). Applications include the development of embedded systems with one or more processors or signal processors and multiple peripherals.

Aligned

A data item stored at an address that is divisible by the highest power of 2 that divides into its size in bytes. Aligned halfwords, words and doublewords therefore have addresses that are divisible by 2, 4 and 8 respectively.

An aligned access is one where the address of the access is aligned to the size of each element of the access.

AMBA

See Advanced Microcontroller Bus Architecture.

At approximately the same time

Two events occur at approximately the same time if a remote observer might not be able to determine the order in which they occurred.

Barrier

An operation that forces a defined ordering of other actions.

Blocking

Describes an operation that prevents following actions from continuing until the operation completes.

A non-blocking operation can permit following operations to continue before it completes.

Byte

An 8-bit data item.

Cache

Any cache, buffer, or other storage structure in a caching Manager that can hold a copy of the data value for a particular address location.

Cache hierarchy

The organisation of different size caches in a hierarchy, typically within the cache with faster access and smaller size closer to the core and larger and slower access ones farther away from the core. The last level of this hierarchy might be conncted to the memory. In this specification, in relation to a referenced cache, above refers to caches closer to the core, and below refers to caches farther from the core.

Cache line

The basic unit of storage in a cache. Its size in words is always a power of two. A cache line must be aligned to the size of the cache line.

The size of the cache line is equivalent to the coherency granule.

Cache state

State of a block of data in a cache, of 64-byte size in this specification. The state determines if the block is cached in any other caches in the system and also if it is different from the copy of the block in memory. See B1.5.2 *Cache state model* for a description of the cache states supported in this specification.

ceil()

A function that returns the lowest integer value that is equal to or greater than the input to the function.

Channel

A set of signals grouped together to communicate a particular set of messages between a Requester and Completer pair. For example, Request channel is used to communicate request messages.

A channel consists of a set of information signals and a separate Valid and Credit signal to provide the channel handshake mechanism.

Coherency granule

The minimum size of the block of memory affected by any coherency consideration. For example, an operation to make two copies of an address coherent makes the two copies of a block of memory coherent, where that block of memory is:

- At least the size of the coherency granule
- Aligned to the size of the coherency granule.

See also Cache line.

Coherent

Data accesses from a set of observers to a memory location are coherent accesses to that memory location by the members of the set of observers are consistent with there being a single total order of all writes to that memory location by all members of the set of observers.

Completer

See Completer.

Component

A distinct functional unit that has at least one AMBA interface. Component can be used as a general term for Manager, Subordinate, peripheral, and interconnect components.

See also Interconnect component, Requester component, Memory Subordinate component, Peripheral Subordinate component, Subordinate component.

Deprecated

Something that is present in the specification for backwards compatibility. Whenever possible you must avoid using deprecated features. These features might not be present in future versions of the specification.

Device

See Peripheral Subordinate component.

Direct Write Transfer

Sending Read data directly from a Snoopee or Subordinate to the Requester bypassing the Home Node.

Downstream

A transaction operates between a Requester component and one or more Subordinate components, and can pass through one or more intermediate components. At any intermediate component, for a given transaction, downstream means between that component and a destination Subordinate component, and includes the destination Subordinate component.

Downstream and upstream are defined relative to the transaction as a whole, not relative to individual data flows within a transaction.

See also Requester component, Peer to Peer, Subordinate component, Upstream.

Downstream cache

See Downstream cache.

Endpoint

See Endpoint.

Final destination

Final destination for a Memory transactin is a peripheral or physical memory, also called an Endpoint.

Flit

See Flit.

GPT

Granule Protection Table. Defines the ranges of physical memory that each PAS can access.

HN

See HN.

ICN

See ICN.

IMPLEMENTATION DEFINED

Means that the behavior is not defined by this specification, but must be defined and documented by individual implementations.

When IMPLEMENTATION DEFINED appears in body text, it is always in small capitals.

IMPLEMENTATION SPECIFIC

Behavior that is not architecturally defined, and might not be documented by an individual implementation. Used when there are a number of implementation options available and the option chosen does not affect software compatibility.

When IMPLEMENTATION SPECIFIC appears in body text, it is always in small capitals.

In a timely manner

See In a timely manner.

Interconnect component

A component with more than one AMBA interface that connects one or more Manager components to one or more Subordinate components

An interconnect component can be used to group together either:

- A set of Managers so that they appear as a single Manager interface
- A set of Subordinates so that they appear as a single Subordinate interface.

See also Component, Requester component, Subordinate component.

IO Coherent node

See IO Coherent node.

Line

See Cache line.

Link

A Link is the connection used for communicating between a Requester and Completer pair.

Link layer Credit

See Link layer Credit.

Load

The action of a Requester component reading the value held at a particular address location. For a processor, a load occurs as the result of executing a particular instruction. Whether the load results in the Requester issuing a read transaction depends on whether the accessed cache line is held in the local cache.

See also Speculative read, Store.

Main memory

The memory that holds the data value of an address location when no cached copies of that location exist. For any location, main memory can be out of date with respect to the cached copies of the location, but main memory is updated with the most recent data value when no cached copies exist.

Main memory can be referred to as memory when the context makes the intended meaning clear.

Memory Management Unit (MMU)

Provides detailed control of the part of a memory system that provides address translation. Most of the control is provided using translation tables that are held in memory, and define the attributes of different regions of the physical memory map.

See also System Memory Management Unit (SMMU).

Memory Subordinate component

A Memory Subordinate component, or *Memory Subordinate*, is a Subordinate component with the following properties:

- A read of a byte from a Memory Subordinate returns the last value written to that byte location.
- A write to a byte location in a Memory Subordinate updates the value at that location to a new value that is obtained by subsequent reads.
- Reading a location multiple times has no side-effects on any other byte location.
- Reading or writing one byte location has no side-effects on any other byte location.

See also Component, Requester component, Peripheral Subordinate component.

Message

See Message.

Observer

A processor or other Requester component, such as a peripheral device, that can generate reads from or writes to memory.

Outstanding request

A transaction is outstanding from the cycle that the request is first issued until either:

- The transaction is fully completed, as determined by the return of all responses that are expected for the transaction.
- It receives RetryAck and PCrdGrant, and is either:
  - Retried using a credit of the appropriate PCrdType, and then is fully completed as determined above.
  - Canceled, and returns the received credit using the PCrdReturn message.

Packet

See Packet.

Page-based Hardware Attributes (PBHA)

Page Based Hardware Attributes (PBHA) is an optional, IMPLEMENTATION DEFINED feature. It allows software to set up to two bits in the translation tables, which are then propagated though the memory system with transactions, and can be used in the system to control system components. The meaning of the bits is specific to the system design.

Peer node

A protocol node of the same type with reference to itself. For example, the peer node for a Request Node is another Request Node.

Peer to Peer

Communication between the same type of nodes. For example, one Request Node to another Request Node.

See also Downstream, Upstream.

Peripheral Subordinate component

A Peripheral Subordinate component is also described as a Peripheral Subordinate. It is recommended that a Peripheral Subordinate has an IMPLEMENTATION DEFINED method of access that is typically described in the data sheet for the component. Any access that is not defined as permitted might cause the Peripheral Subordinate to fail, but must complete in a protocol-correct manner to prevent system deadlock. The protocol does not require continued correct operation of the peripheral.

See also Memory Subordinate component, Subordinate component.

Permission to store

A component has permission to store if it can perform a store to the associated cache line without informing any other caching Manager or the interconnect.

Phit

See Phit.

PoC

See PoC.

PoP

See PoP.

PoPA

See PoPA.

PoS

See PoS.

Prefetching

Prefetching refers to speculatively fetching instructions or data from the memory system. In particular, instruction prefetching is the process of fetching instructions from memory before the instructions that precede them, in simple sequential execution of the program, have finished executing. Prefetching an instruction does not mean that the instruction has to be executed.

In this specification, references to instruction or data fetching apply also to prefetching, unless the context explicitly indicates otherwise.

Protocol credit

See Protocol credit.

Realm Management Extensions (RME)

the Realm Management Extension (RME) is an extension to the Armv9 A-profile architecture. RME is one component of the Arm Confidential Computer Architecture (CCA). Together with the other components of the Arm CCA, RME enables support for dynamic, attestable. and trusted execution environments, Realms, to be run on an Arm PE. RME adds two additional Security states, Root and Realm, and two physical address spaces, Root and Realm, and provides hardware-based isolation that allows execution contexts to run in different Security states and share resources in the system.

Requester

See Requester.

Requester component

A component that initiates transactions.

It is possible that a single component can act as both a Requester component and as a Subordinate component. For example, a Direct Memory Access (DMA) component can be a Manager component when it is initiating transactions to move data, and a Subordinate component when it is being programmed.

See also Component, Interconnect component, Subordinate component.

RN

See RN.

Signed

A value with Two's complement notation, unless otherwise stated.

SN

See SN.

Snoop filter

A precise snoop filter that is able to track precisely the cache lines that might be allocated within a Requester.

Snooped cache

A hardware-coherent cache that receives Snoop transactions.

Speculative read

A transaction that a Manager issues when it might not need the transaction to be performed because it already has a copy of the accessed cache line in its local cache. Typically, a Manager issues a speculative read in parallel with a local cache lookup. This gives lower latency than looking in the local cache first, and then issuing a read transaction only if the required cache line is not found in the local cache.

See also Load.

Stash

The action of placing data in a cache closer to the agent that is expected to be the next user of the data.

Store

The action of a Requester component changing the value held at a particular address location. For a processor, a store occurs as the result of executing a particular instruction. Whether the store results in the Requester issuing a read or write transaction depends on whether the accessed cache line is held in the local cache, and if it is in the local cache, the state it is in.

Subordinate

An agent that agent that receives and responds to requests.

Subordinate component

A component that receives transactions and responds to them.

It is possible that a single component can act as both a Subordinate component and as a Manager component. For example, a Direct Memory Access (DMA) component can be a Subordinate component when it is being programmed and a Manager component when it is initiating transactions to move data.

See also Requester component, Memory Subordinate component, Peripheral Subordinate component.

Synchronization barrier

See Barrier.

System Memory Management Unit (SMMU)

A system-level MMU. That is, a system component that provides address translation from a one address space to another. An SMMU provides one or more of:

- Virtual Address (VA) to Physical Address (PA) translation
- VA to Intermediate Physical Address (IPA) translation
- IPA to PA translation.

See also Memory Management Unit (MMU).

TLB

See Translation Lookaside Buffer (TLB).

Transaction

See Transaction.

Translation Lookaside Buffer (TLB)

A memory structure containing the results of translation table walks. TLBs help to reduce the average cost of a memory access.

See also System Memory Management Unit (SMMU), Translation table, Translation table walk.

Translation table

A table held in memory that defines the properties of memory areas of various sizes from 1KB.

See also Translation Lookaside Buffer (TLB), Translation table walk.

Translation table walk

The process of doing a full translation table lookup.

See also Translation Lookaside Buffer (TLB), Translation table.

Unaligned

An unaligned access is an access where the address of the access is not aligned to the size of an element of the access.

Unaligned memory accesses

Are memory accesses that are not, or might not be, appropriately halfword-aligned, word-aligned, or doubleword-aligned.

UNPREDICTABLE

In the AMBA Architecture, it means that the behavior cannot be relied upon.

UNPREDICTABLE behavior must not be documented or promoted as having a defined effect.

When UNPREDICTABLE appears in body text, it is always in small capitals.

Upstream

A transaction operates between a Requester component and one or more Subordinate components, and can pass through one or more intermediate components. At any intermediate component, for a given transaction, *upstream* means between that component and the originating Requester component, and includes the originating Requester component.

Downstream and upstream are defined relative to the transaction as a whole, not relative to individual data flows within the transaction.

See also Downstream, Requester component, Peer to Peer, Subordinate component.

Write-Back cache

A cache in which when a cache hit occurs on a store access, the data is only written to the cache. Data in the cache can therefore be more up-to-date than data in main memory. Any such data is written back to main memory when the cache line is cleaned or re-allocated. Another common term for a Write-Back cache is a CopyBack cache.

Write-Invalidate protocol

See Write-Invalidate protocol.