AMBA® 5 CHI Architecture Specification

arm

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
ARM IHI 0050C (ID050218)

AMBA 5 CHI Architecture Specification

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.

Release Information

The *Change history* lists the changes made to this specification.

Change history

Date	Issue	Confidentiality	Change
12 June 2014	A	Confidential	First limited release
04 August 2017	B	Non-Confidential	First public release
08 May 2018	C	Non-Confidential	Second public release

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 Limited ("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 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, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

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 shall prevail.

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 CONSQUENTIAL 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.

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement specifically covering this document with Arm, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms.

Words and logos marked with ® or ™ are registered trademarks or trademarks of Arm Limited or its affiliates in the EU and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines http://www.arm.com/about/trademark-usage-guidelines.php.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.

Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.

In this document, where the term Arm is used to refer to the company it means “Arm or any of its subsidiaries as appropriate”.

ARM 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 THE RELEVANT AMBA SPECIFICATION ACCOMPANYING THIS LICENCE. ARM IS ONLY WILLING TO LICENSE 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. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENCE, ARM IS UNWILLING TO LICENSE THE RELEVANT AMBA SPECIFICATION TO YOU AND YOU MAY NOT USE OR COPY THE RELEVANT AMBA SPECIFICATION AND YOU SHOULD PROMPTLY RETURN THE RELEVANT AMBA SPECIFICATION TO ARM.

“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 Arm 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

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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`

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Contents
AMBA 5 CHI ArchitectureSpecification

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Preface

This preface introduces the *AMBA 5 CHI Architecture Specification*. It contains the following sections:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

About this specification

This specification describes the AMBA 5 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

This book is organized into the following chapters:

Chapter 1 *Introduction*

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

Chapter 2 *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 3 *Network Layer*

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

Chapter 4 *Coherence Protocol*

Read this for an introduction to the coherence protocol.

Chapter 5 *Interconnect Protocol Flows*

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

Chapter 6 *Exclusive Accesses*

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

Chapter 7 *Cache Stashing*

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

Chapter 8 *DVM Operations*

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

Chapter 9 *Error Handling*

Read this for a description of the error response requirements.

Chapter 10 *Quality of Service*

Read this for a description of the mechanisms that the protocol includes to support *Quality of Service* (QoS).

Chapter 11 *Data Source and Trace Tag*

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

Chapter 12 *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 13 *Link Handshake*

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

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 14 *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 15 *Properties, Parameters, and Broadcast Signals*

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

Appendix A *Message Field Mappings*

Read this for the field mappings for messages.

Appendix B *Communicating Nodes*

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

Appendix C *Revisions*

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

Glossary Read this for definitions of terms used in this specification.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Conventions

The following sections describe conventions that this specification can use:

- *Typographical conventions*.
- *Timing diagrams*.
- *Signals* on page xiv.
- *Numbers* on page xiv.

Typographical conventions

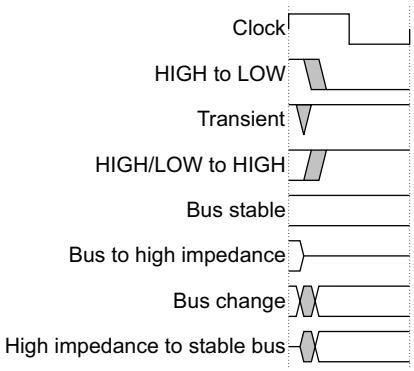
The typographical conventions are:

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 *Key to timing diagram conventions* figure explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must 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.



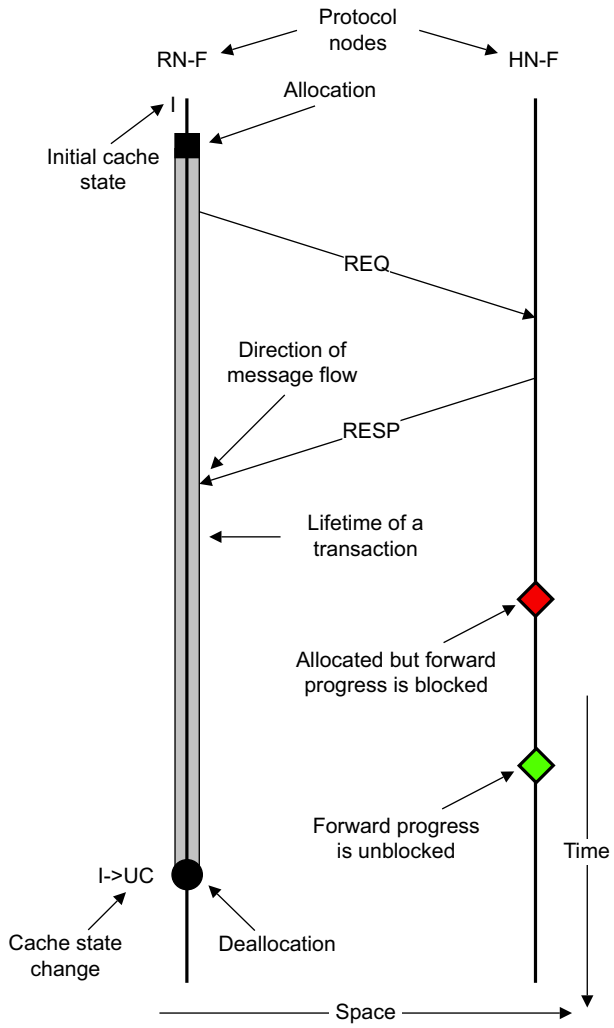
Key to timing diagram conventions

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

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Time-Space diagrams

The *Key to Time-Space diagram conventions* figure explains the format used to illustrate protocol flow.



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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 relevant publications from Arm.

See the Infocenter `http://infocenter.arm.com`, for access to Arm documentation.

Arm publications

- *AMBA® AXI and ACE Protocol Specification* (ARM IHI 0022).

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Feedback

Arm welcomes feedback on its documentation.

Feedback on this specification

If you have comments on the content of this specification, send an e-mail to `errata@arm.com`. Give:

- The title, *AMBA 5 CHI ArchitectureSpecification*.
- The number, ARM IHI 0050C.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 1
Introduction

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

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

1.1 Architecture overview

The CHI architecture provides a comprehensive layered specification to build small, medium, and large systems comprising of multiple components using a scalable coherent hub interface and on-chip interconnect. The CHI architecture permits flexibility on the topology of the component connections, which can be driven from the system performance, power, and area requirements.

The components of CHI based systems can comprise of standalone processors, processor clusters, graphic processors, memory controllers, I/O bridges, PCIe subsystems and the interconnect itself.

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.
  - 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).
- 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

1.1.1 Architecture layers

Functionality is grouped into the following layers:

- Protocol.
- Network.
- Link.

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

Table 1-1 Layers of the CHI architecture

Layer	Communication granularity	Primary function
Protocol	Transaction	The Protocol layer is the top-most 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, and add to the packet, the source and target node IDs required to route the packet over the interconnect to the required destination.
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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

1.2 Topology

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

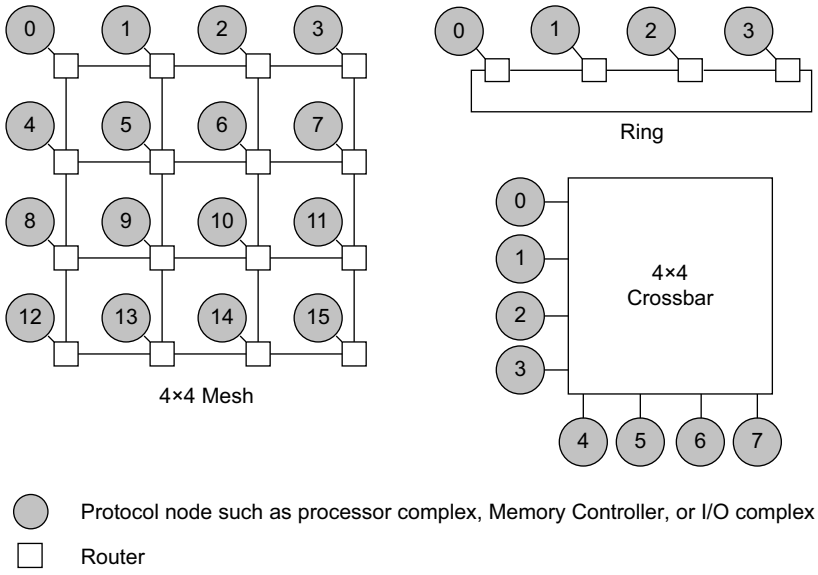


Figure 1-1 Example interconnect topologies

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

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

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

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

1.3 Terminology

The following terms have a specific meaning in this specification:

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

Message 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 might be made up of a number of packets.

Packet The granule-of-transfer over the interconnect between endpoints. A message might 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 that enables it to be routed independently over the interconnect.

Flit 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 The physical layer transfer unit. A flit can be made up of one or more phits. A phit is defined as one transfer between two adjacent network devices.

Note

For CHI, all flits consist of a single phit.

PoS Point of Serialization. A point within the interconnect where the ordering between Requests from different agents is determined.

PoC Point of Coherence. 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 it is the HN-F in the interconnect.

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 and such a component is also referred to as a master. 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 transaction it receives from another component. A Completer can either be an interconnect component or a component, such as a slave, that is outside of the interconnect.

Master An agent that independently issues transactions. Typically a master is the most upstream agent in a system. A master can also be referred to as a Requester.

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

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Endpoint Another name for a slave component. As the name implies, 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. A *Link layer Credit* (L-Credit) can be considered to be a credit for a single hop at the Link layer.

ICN A short form of interconnect, which is the CHI transport mechanism that is used for communication between protocol nodes. An ICN might include a fabric of switches connected in a ring, mesh, crossbar, or some other topology. The ICN might include protocol nodes such as Home Node and Misc Node. The topology of the ICN is IMPLEMENTATION DEFINED.

IPA Intermediate Physical Address. In two stage address translation:

- Stage one results in an Intermediate Physical Address.
- Stage two provides the Physical Address.

RN Request Node. Generates protocol transactions, including reads and writes, to the interconnect.

HN Home Node. Node located within the interconnect that receives protocol transactions from Request Nodes, completes the required Coherency action, and returns a Response.

SN Slave Node. Node that receives a Request from a Home Node, completes the required action, and returns a Response.

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

IO Coherent node An RN that generates some 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.

Write-Invalidate protocol

A protocol in which an RN writing to a cache line that is shared in the system must invalidate all the shared 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. However, in a sufficiently idle system, it will make progress and complete without requiring any explicit action.

Don’t Care A field value that indicates that the field can be set to any value, including reserved or illegal values. Any component receiving a packet with a field value set to Don't Care must ignore the value set for that field.

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

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

1.4 Transaction classification

The protocol transactions that this specification supports, and their major classification, are as follows:

Read

- ReadNoSnp.
- ReadOnce.
- ReadOnceCleanInvalid.
- ReadOnceMakeInvalid.
- ReadClean.
- ReadNotSharedDirty.
- ReadShared.
- ReadUnique.

Dataless

- CleanUnique.
- MakeUnique.
- Evict.
- StashOnceUnique.
- StashOnceShared.
- CleanShared.
- CleanSharedPersist.
- CleanInvalid.
- MakeInvalid.

Write

- WriteNoSnpPtl, WriteNoSnpFull.
- WriteUniquePtl, WriteUniqueFull.
- WriteUniquePtlStash, WriteUniqueFullStash.
- WriteBackPtl, WriteBackFull.
- WriteCleanFull.
- WriteEvictFull.

Atomic

- AtomicStore.
- AtomicLoad.
- AtomicSwap.
- AtomicCompare.

In this specification, unless specifically stated otherwise:

- ReadOnce* represents ReadOnce, ReadOnceCleanInvalid and ReadOnceMakeInvalid.
- WriteNoSnp represents both WriteNoSnpPtl and WriteNoSnpFull.
- WriteUnique represents WriteUniquePtl, WriteUniqueFull, WriteUniquePtlStash and WriteUniqueFullStash.
- WriteBack represents both WriteBackPtl and WriteBackFull.
- StashOnce represents both StashOnceUnique and StashOnceShared.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Other

- DVMOp.
- PrefetchTgt.
- PCrdReturn.

Snoop

- SnpOnceFwd.
- SnpOnce.
- SnpStashUnique.
- SnpStashShared.
- SnpCleanFwd.
- SnpClean.
- SnpNotSharedDirtyFwd.
- SnpNotSharedDirty.
- SnpSharedFwd.
- SnpShared.
- SnpUniqueFwd.
- SnpUnique.
- SnpUniqueStash.
- SnpCleanShared.
- SnpCleanInvalid.
- SnpMakeInvalid.
- SnpMakeInvalidStash.
- SnpDVMOp.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

1.5 Coherence overview

Hardware coherency enables the sharing of memory by system components without the requirement to perform software cache maintenance to maintain coherency between caches.

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

1.5.1 Coherency model

Figure 1-2 shows an example coherent system that includes three master 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 master components.

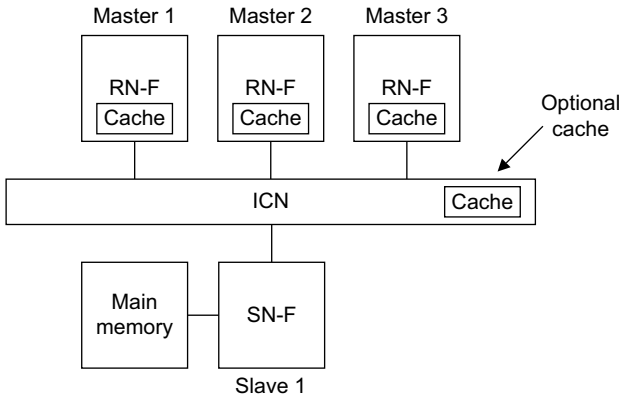


Figure 1-2 Example coherency model

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

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

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

Note

Although not a requirement, it is acceptable to update main memory while cached copies still exist.

The protocol enables master components to determine whether a cache line is the only copy of a particular memory location, or if there might be other copies of the same location, so that:

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

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

1.5.2 Cache state model

To determine whether an action is required when a component accesses a cache line, the protocol defines cache states. 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 this cache. When Shared, the cache line might exist in more than one cache, but this is not guaranteed.

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, and this cache must ensure that main memory is eventually updated.

Full, Partial, Empty A Full cache line has all bytes valid. A Partial cache line might have some bytes valid, but not all bytes valid. An Empty cache line has no bytes valid.

Figure 1-3 shows the seven state cache model. *Cache line states* on page 4-140 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 1-3 UC, UD, SC, and SD are all Full cache line states.

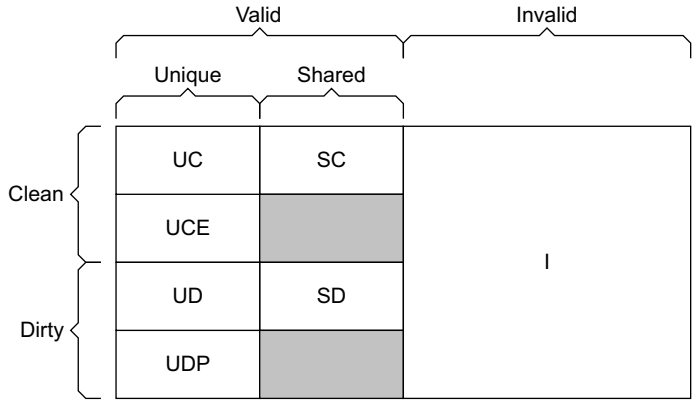


Figure 1-3 Cache state model

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

1.6 Component naming

Components are classified by CHI protocol node type:

RN Request Node. Generates protocol transactions, including reads and writes to the interconnect.

An RN 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 RNs.

An HN is further categorized as:

HN-F Is 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 RN.
- Is expected to be the *Point of Serialization* (PoS) that manages order between memory requests.
- Might include a directory or snoop filter to reduce redundant snoops.

Note

IMPLEMENTATION SPECIFIC, can include an integrated ICN cache.

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

MN Miscellaneous Node:
- Receives a DVM transaction from an RN, completes the required action, and returns a response.

SN Slave Node. An SN receives a request from an HN, completes the required action and returns a response.

An SN is further categorized as:

SN-F A Slave Node type used for Normal memory. It can process Non-snoopable read write, and atomic requests, including exclusive variants of them, and *Cache Maintenance Operation* (CMO) requests.

SN-I A Slave Node type used for peripherals or Normal memory. It can process Non-snoopable read, write and atomic requests, including exclusive variants of them, and CMO requests.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Figure 1-4 shows various protocol node types connected through an interconnect.

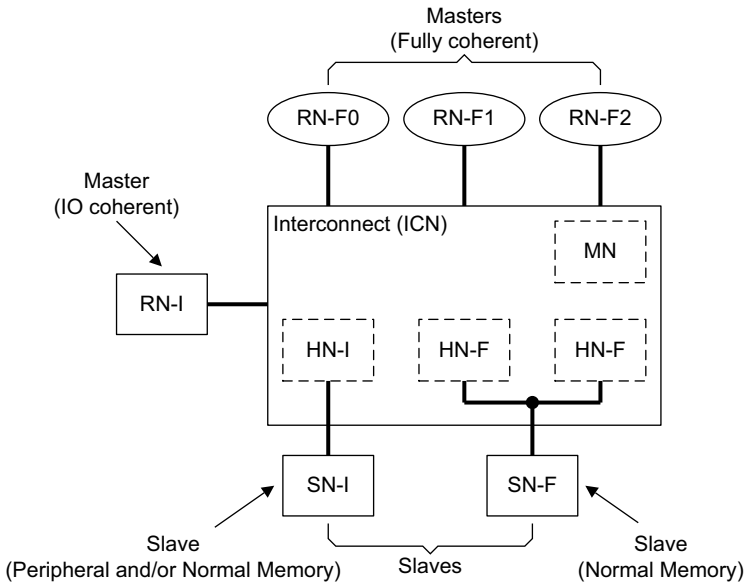


Figure 1-4 Protocol node examples

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

1.7 Read data source

In a CHI-based system, a Read request can obtain data from different sources. Figure 1-5 shows that these sources are:

- Cache within ICN.
- Slave Node.
- Peer RN-F.

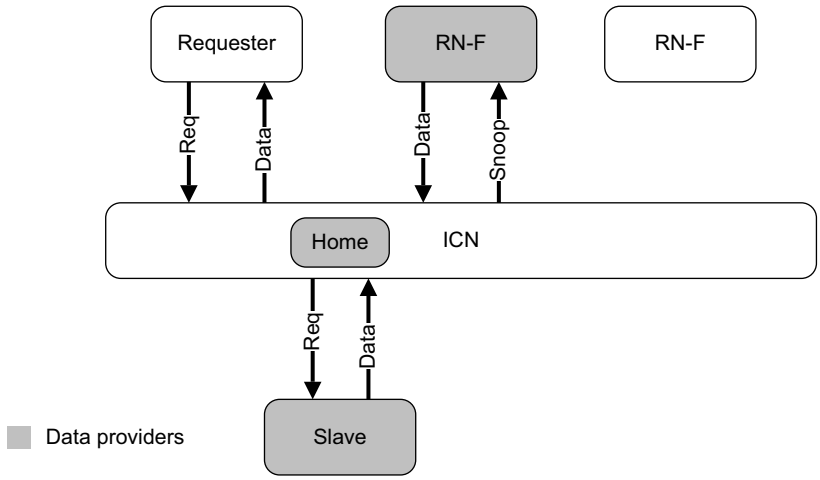


Figure 1-5 Possible Data providers for a Read request

One option for Home is to request that the RN-F or Slave 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 this 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.

This specification expects the following Read latency saving techniques to be used by the system:

Direct Memory Transfer **(DMT)**

Defines the feature that permits the Slave 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 it has sent Data to the Requester and, in some cases, it also has to send a copy of Data to the Home.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 2
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:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.1 Channels overview

Communication between nodes is channel based. Table 2-1 shows the channel naming and the channel designations at the RN and SN nodes.

This section uses shorthand naming for the channels to describe the transaction structure. Table 2-1 shows the shorthand name and the physical channel name that exists on the RN or SN component.

See *Channel* on page 12-301 for the mapping of physical channels on the RN and SN components.

Table 2-1 Channel naming and designation at the RN and SN nodes

Channel	RN channel designation	SN channel designation
REQ	TXREQ. Outbound Request.	RXREQ. Inbound Request.
WDAT	TXDAT. Outbound Data. Used for write data, atomic data, snoop data, forward data.	RXDAT. Inbound Data. Used for write data, atomic data.
SRSP	TXRSP. Outbound Response. Used for Snoop Response and Completion Acknowledge.	-
CRSP	RXRSP. Inbound Response. Used for responses from the Completer.	TXRSP. Outbound Response. Used for responses from the Completer.
RDAT	RXDAT. Inbound Data. Used for read data, atomic data.	TXDAT. Outbound Data. Used for read data, atomic data.
SNP	RXSNP. Inbound Snoop Request.	-

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.2 Channel fields

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

- *Transaction request fields*.
- *Snoop request fields* on page 2-35.
- *Data fields* on page 2-36.
- *Response fields* on page 2-38.

2.2.1 Transaction request fields

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

The term transaction structure is used to describe the different packets that build a transaction and the transaction structure can vary depending on a number of factors. Table 2-2 shows which Request fields can affect the transaction structure. More information on the different transaction structures can be found in *Transaction structure* on page 2-39 and *Flit packet definitions* on page 12-310.

Table 2-2 Request channel fields

Field	Affects structure	Description
QoS	No	Quality of Service priority. Specifies 1 of 16 possible priority levels for the transaction with ascending values of QoS indicating higher priority levels. See Chapter 10 *Quality of Service*.
TgtID	No	Target ID. The node ID of the port on the component to which the packet is targeted. See *Details of transaction identifier fields* on page 2-74 and *System address map* on page 3-132.
SrcID	No	Source ID. The node ID of the port on the component from which the packet was sent. See *Details of transaction identifier fields* on page 2-74.
TxnID	No	Transaction ID. A transaction has a unique transaction ID per source node. See *Details of transaction identifier fields* on page 2-74.
LPID	No	Logical Processor ID. Used in conjunction with the SrcID field to uniquely identify the logical processor that generated the request. See *Logical Processor Identifier* on page 2-95.
ReturnNID	No	Return Node ID. The node ID that the response with Data is to be sent to. See *Details of transaction identifier fields* on page 2-74.
ReturnTxnID	No	Return Transaction ID. The unique transaction ID that conveys the value of TxnID in the data response from the Slave. See *Details of transaction identifier fields* on page 2-74.
StashNID	No	Stash Node ID. The node ID of the Stash target. See *StashNID* on page 12-315.
StashNIDValid	Yes	Stash Node ID Valid. Indicates that the StashNID field has a valid Stash target value. See *StashNIDValid* on page 12-316.
StashLPID	No	Stash Logical Processor ID. The ID of the logical processor at the Stash target. See *StashLPID* on page 12-316.
StashLPIDValid	No	Stash Logical Processor ID Valid. Indicates that the StashLPID field value must be considered as the Stash target. See *StashLPIDValid* on page 12-316.
Opcode	Yes	Request opcode. Specifies the transaction type and is the primary field that determines the transaction structure. See *Request types* on page 4-142 and *REQ channel opcodes* on page 12-318.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 2-2 Request channel fields (continued)

Field	Affects structure	Description
Addr	No	Address. The address of the memory location being accessed for read and write requests. See *Address* on page 2-106 and *Addr* on page 12-322.
NS	No	Non-secure. Determines if the transaction is Non-secure or Secure. See *Non-secure bit* on page 2-107 and *NS* on page 12-322.
Size	Yes	Data size. Specifies the size of the data associated with the transaction. This determines the number of data packets within the transaction. See *Data transfer* on page 2-115.
AllowRetry	Yes	Allow Retry. Determines if the target is permitted to give a Retry response. See *Request Retry* on page 2-126.
PCrdType	No	Protocol Credit Type. Indicates the type of Protocol Credit being used by a request that has the AllowRetry field deasserted. See *Request Retry* on page 2-126.
ExpCompAck	Yes	Expect CompAck. Indicates that the transaction will include a Completion Acknowledge message. See *Transaction structure* on page 2-39 and *Ordering* on page 2-96.
MemAttr	No	Memory attribute. Determines the memory attributes associated with the transaction. See *Memory Attributes* on page 2-107.
SnpAttr	No	Snoop attribute. Specifies the snoop attributes associated with the transaction. See *Likely Shared* on page 2-112.
SnoopMe	No	Snoop Me. Indicates that Home must determine whether to send a snoop to the Requester. See *Atomics* on page 2-62.
LikelyShared	No	Likely Shared. Provides an allocation hint for downstream caches. See *Likely Shared* on page 2-112.
Excl	No	Exclusive access. Indicates that the corresponding transaction is an Exclusive access transaction. See Chapter 6 *Exclusive Accesses*.
Order	Yes	Order requirement. Determines the ordering requirement for this request with respect to other transactions from the same agent. See *Ordering* on page 2-96.
Endian	No	Endianness. Indicates the endianness of Data in the Data packet. See *Endianness* on page 2-119.
TraceTag	No	Trace Tag. Provides additional support for the debugging, tracing, and performance measurement of systems. See Chapter 11 *Data Source and Trace Tag*.
RSVDC	No	User defined. See *RSVDC* on page 12-334.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.2.2 Snoop request fields

A Snoop request contains a subset of the fields defined for a Request packet. Table 2-3 shows which Request fields can affect the transaction structure.

Table 2-3 Snoop request fields

Field	Affects structure	Description
QoS	No	Quality of Service priority. As defined in *Request channel fields* on page 2-33. See Chapter 10 *Quality of Service*.
TxnID	No	Transaction ID. As defined in *Request channel fields* on page 2-33. See *Details of transaction identifier fields* on page 2-74.
FwdNID	No	Forward Node ID. Node ID of the original Requester. See *Details of transaction identifier fields* on page 2-74.
FwdTxnID	No	Forward Transaction ID. The transaction ID used in the Request by the original Requester. See *Details of transaction identifier fields* on page 2-74.
StashLPID	No	Stash Logical Processor ID. As defined in *Request channel fields* on page 2-33. See *Stash messages* on page 7-250.
StashLPIDValid	No	Stash Logical Processor ID Valid. As defined in *Request channel fields* on page 2-33. See *Stash messages* on page 7-250.
VMIDExt	No	Virtual Machine ID Extension. See *DVM Operation types* on page 8-261.
SrcID	No	Source ID. As defined in *Request channel fields* on page 2-33. See *Details of transaction identifier fields* on page 2-74.
Opcode	Yes	Snoop opcode. See *Snoop request fields* and *SNP channel opcodes* on page 12-321.
Addr	No	Address. The address of the memory location being accessed for snoop requests. See *Address* on page 2-106 and *Addr* on page 12-322.
NS	No	Non-secure or Secure access. As defined in *Request channel fields* on page 2-33. See *Non-secure bit* on page 2-107 and *NS* on page 12-322.
DoNotGoToSD	No	Do Not Go To SD state. Controls Snoopee use of SD state. See *Do not transition to SD* on page 2-114.
DoNotDataPull	Yes	Do Not Data Pull. Instructs the Snoopee that it is not permitted to use the Data Pull feature associated with Stash requests. See *Snoop requests and Data Pull* on page 7-244.
RetToSrc	Yes	Return to Source. Instructs the receiver of the snoop to return Data with the Snoop response. See *Shared clean state return* on page 4-194.
TraceTag	No	Trace Tag. As defined in *Request channel fields* on page 2-33. See Chapter 11 *Data Source and Trace Tag*.

Note

This specification does not define a TgtID field for the Snoop Request. See *Target ID determination for Snoop Request messages* on page 3-135.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.2.3 Data fields

Table 2-4 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 2-4 Data packet fields

Field	Description
QoS	Quality of Service priority. As defined in *Request channel fields* on page 2-33. See Chapter 10 *Quality of Service*.
TgtID	Target ID. As defined in *Request channel fields* on page 2-33. See *Details of transaction identifier fields* on page 2-74.
SrcID	Source ID. As defined in *Request channel fields* on page 2-33. See *Details of transaction identifier fields* on page 2-74.
TxnID	Transaction ID. As defined in *Request channel fields* on page 2-33. See *Details of transaction identifier fields* on page 2-74.
HomeNID	Home Node ID. The Node ID of the target of the CompAck response to be sent from the Requester. See *Details of transaction identifier fields* on page 2-74.
DBID	Data Buffer ID. The ID provided to be used as the TxnID in the response to this message. See *Details of transaction identifier fields* on page 2-74 and *Ordering* on page 2-96.
Opcode	Data opcode. Indicates, for example, if the data packet is related to a Read transaction, a Write transaction, or a Snoop transaction. See *DAT channel opcodes* on page 12-322.
RespErr	Response Error status. Indicates the error status associated with a data transfer. See Chapter 6 *Exclusive Accesses* and *Error response fields* on page 9-273.
Resp	Response status. Indicates the cache line state associated with a data transfer. See *Response types* on page 4-163.
FwdState	Forward State. Indicates the cache line state associated with a data transfer to the Requester from the receiver of the snoop. See *FwdState* on page 12-331.
DataPull	Data Pull. Indicates the inclusion of an implied Read request in the Data response. See *Snoop requests and Data Pull* on page 7-244.
DataSource	Data Source. The value indicates the source of the data in a read Data response. See *Data Source indication* on page 11-292.
CCID	Critical Chunk Identifier. Replicates the address offset of the original transaction request. See *Data transfer* on page 2-115.
DataID	Data Identifier. Provides the address offset of the data provided in the packet. See *Data transfer* on page 2-115.
BE	Byte Enable. For a data write, or data provided in response to a snoop, indicates which bytes are valid. See *Data transfer* on page 2-115.
Data	Data payload. See *Data transfer* on page 2-115.
DataCheck	Data Check. Detects data errors in the DAT packet. See *Data Check* on page 9-283.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 2-4 Data packet fields (continued)

Field	Description
Poison	Poison. Indicates that a set of data bytes has previously been corrupted. See *Poison* on page 9-282.
TraceTag	Trace Tag. As defined in *Request channel fields* on page 2-33. See Chapter 11 *Data Source and Trace Tag*.
RSVDC	User defined. See *RSVDC* on page 12-334.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.2.4 Response fields

Table 2-5 describes the fields associated with a Response packet. The fields in a Response packet do not affect the transaction structure.

Table 2-5 Response packet fields

Field	Description
QoS	Quality of Service priority. As defined in *Request channel fields* on page 2-33. See Chapter 10 *Quality of Service*.
TgtID	Target ID. As defined in *Request channel fields* on page 2-33. See *Details of transaction identifier fields* on page 2-74.
SrcID	Source ID. As defined in *Request channel fields* on page 2-33. See *Details of transaction identifier fields* on page 2-74.
TxnID	Transaction ID. As defined in *Request channel fields* on page 2-33. See *Details of transaction identifier fields* on page 2-74.
DBID	Data Buffer ID. As defined in *Data packet fields* on page 2-36. See *Details of transaction identifier fields* on page 2-74 and *Ordering* on page 2-96.
PCrdType	Protocol Credit Type. See *PCrdType* on page 2-128.
Opcode	Response opcode. Specifies the response type. See *RSP channel opcodes* on page 12-320.
RespErr	Response Error status. As defined in *Data packet fields* on page 2-36. See Chapter 6 *Exclusive Accesses* and *Error response fields* on page 9-273.
Resp	Response status. As defined in *Data packet fields* on page 2-36. See *Response types* on page 4-163.
FwdState	Forward State. As defined in *Data packet fields* on page 2-36. See *FwdState* on page 12-331.
DataPull	Data Pull. As defined in *Data packet fields* on page 2-36. See *Snoop requests and Data Pull* on page 7-244.
TraceTag	Trace Tag. As defined in *Request channel fields* on page 2-33. See Chapter 11 *Data Source and Trace Tag*.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.3 Transaction structure

This section describes the structure of the transactions described in *Transaction classification* on page 1-23 together with the channel usage.

Where sufficient, the structure of a transaction is described as seen at a single interface.

The transaction types presented in this section are:

- Request transactions without a Retry.
- Request transactions with a Retry.
- Snoop transactions.

For a Request transaction to complete, a Snoop transaction might be required. However, such dependencies are not visible at the Requester, so these two transaction types are generally presented separately. See Chapter 5 *Interconnect Protocol Flows* for examples of how Request and Snoop flows are related.

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 *Transaction Retry sequence* on page 2-66.

2.3.1 Request transaction structure

The Request transaction structure is described in the following groupings:

- *Snoopable Reads excluding ReadOnce**.
- *ReadNoSnp, ReadOnce, ReadOnceCleanInvalid, ReadOnceMakeInvalid* on page 2-44.
- *Reads with separate Non-data and Data-only responses* on page 2-50.
- *Dataless* on page 2-55.
- *Writes* on page 2-57.
- *Atomics* on page 2-62.
- *DVM* on page 2-64.
- *PrefetchTgt* on page 2-65.

Snoopable Reads excluding ReadOnce*

The Snoopable Read transactions excluding ReadOnce* are:

- ReadClean.
- ReadNotSharedDirty.
- ReadShared.
- ReadUnique.

The Snoopable Read transactions described in this section are used by a fully coherent Requester (RN-F) to carry out a read when the snooping of other Snoopable Requesters (RN-Fs) is required.

The transaction structures for ReadOnce* transactions, which are also Snoopable transactions, are described along with ReadNoSnp in subsequent sections of this chapter.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Transaction structure with DMT

A Snoopable Read transaction with DMT is used when the data can be sent directly from the Slave to the original Requester. The progress of the Snoopable Read transaction with DMT is as follows:

1. The Requester sends a Snoopable read request on the REQ channel:
   - ReadClean.
   - ReadNotSharedDirty.
   - ReadShared.
   - ReadUnique.
2. The ICN sends a ReadNoSnp request to SN on the REQ channel:
3. The SN, as Completer, forwards the read data and any associated transaction response with the CompData opcode directly to the Requester on the RDAT channel.

   The read data can be sent using multiple transfers. See *Data transfer* on page 2-115.
4. Because the transaction request ExpCompAck bit is set, the Requester must return an acknowledgement, using the CompAck opcode on the SRSP channel to indicate that the transaction has completed.

Figure 2-1 shows the transaction structure.

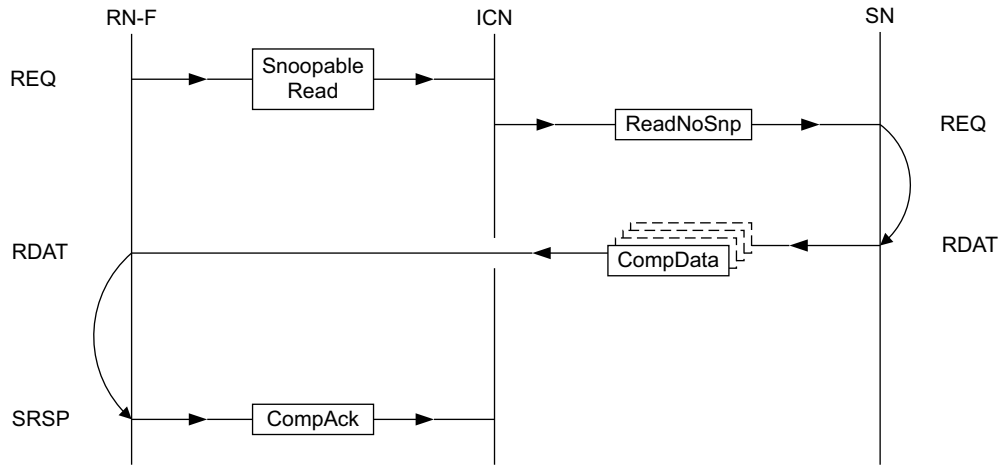


Figure 2-1 Snoopable Read DMT structure

The Request/Response rules are:

- CompData must only be sent by the Completer after the associated request is received.
- The Requester must only send CompAck after at least one CompData packet is received.

Note

Prior to CHI Issue C, CompAck must not be sent until all Data packets of read data have been received.

The Snoopable Read transaction that Figure 2-1 shows can include a separate Comp and Data response instead of the CompData response. The transaction structure for reads with separate Comp and Data response is described in *Reads with separate Non-data and Data-only responses* on page 2-50.

Note

Prior to CHI issue C, a separate Comp and Data response is not permitted.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DMT restrictions

The following restrictions apply to DMT transactions:

- A Requester can reuse the TxnID only after all the responses that could use the TxnID have been returned.
- Home must wait to send a DMT request to SN-F until it is guaranteed that all the following applicable conditions are true:
  - A Snoop request does not need to be sent.
  - If a Snoop request is sent, then the Snoop response is received without a Dirty copy of the cache line being returned.
  - If the Snoop response returns a partial Dirty copy of the cache line, then the DMT can only be sent if the partial data is written to SN-F and a completion for the write is received.
  - If the snoop was a Forwarding type snoop, then it did not result in the cache line being forwarded to the Requester.

Note

Home can enable DMT in combination with DCT but must wait for the DCT response to be received before sending the DMT request to SN-F.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Transaction structure with DCT

A Snoopable Read transaction with DCT is used when the data is to be sent directly from the Snooped RN-F to the original Requester. The progress of the Snoopable read transaction with DCT is as follows:

1. The Requester sends a Snoopable read request on the REQ channel:
   - ReadClean.
   - ReadNotSharedDirty.
   - ReadShared.
   - ReadUnique.
2. The ICN sends a Snp[*]Fwd request to RN-F on the SNP channel.
3. The RN-F as Completer forwards the read data and any associated transaction response to RN with the CompData opcode on the DAT channel.

   The data can be sent using multiple transfers. See *Data transfer* on page 2-115.
4. The RN-F also forwards a SnpRespFwded response to the ICN on the SRSP channel to indicate that read data was forwarded to the Requester.
5. Because the transaction request ExpCompAck bit is set, the Requester must return an acknowledgement, using the CompAck opcode on the SRSP channel to indicate that the transaction has completed.

Figure 2-2 shows the transactions structure.

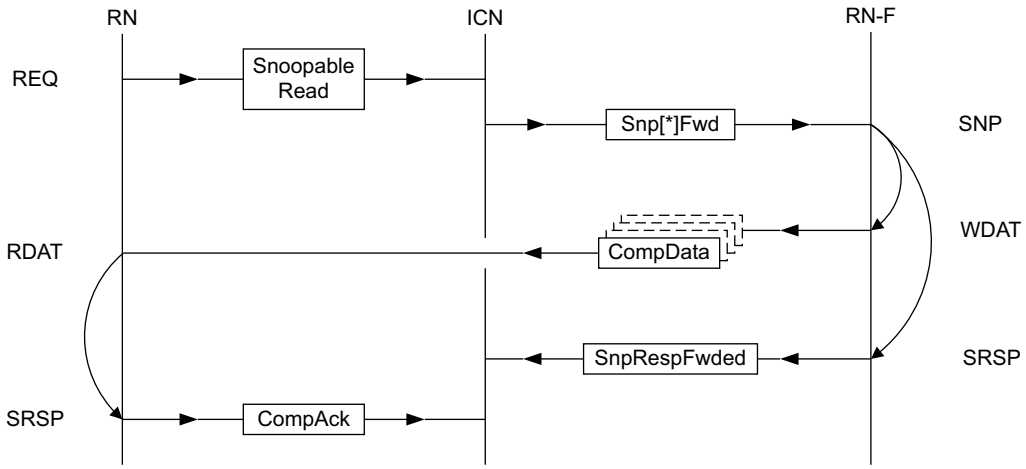


Figure 2-2 Snoopable Read DCT structure

The Request/Response rules are:

- CompData must only be sent by the Completer after the associated request is received.
- CompAck can be sent as soon as the first Data packet of read data has been received.

Note

Prior to CHI Issue C, CompAck must not be sent until all Data packets of read data have been received.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Transaction structure without Direct Data Transfer

This section shows the Read transactions structure without DMT or DCT.

The progress of a Snoopable Read transaction without Direct Data Transfer, from the Requester perspective, is identical to a Snoopable Read transaction with Direct Data Transfer and is as follows:

1. The Requester sends a Snoopable read request on the REQ channel:
   - ReadClean.
   - ReadNotSharedDirty.
   - ReadShared.
   - ReadUnique.
2. The Completer returns the read data and any associated transaction response with the CompData opcode on the RDAT channel.

   The read data can be sent using multiple transfers. See *Data transfer* on page 2-115.
3. Because the ExpCompAck bit is set, the Requester must return an acknowledgement, using the CompAck opcode on the SRSP channel to indicate that the transaction has completed.

   CompAck can be sent as soon as the first Data packet of read data has been received.

   Separate Comp and Data responses are possible. See *Reads with separate Non-data and Data-only responses* on page 2-50.

   Note

   Prior to CHI issue C:
   - CompAck must not be sent until all Data packets of read data have been received.
   - Separate Comp and Data responses are not permitted.

Figure 2-3 shows the transaction structure.

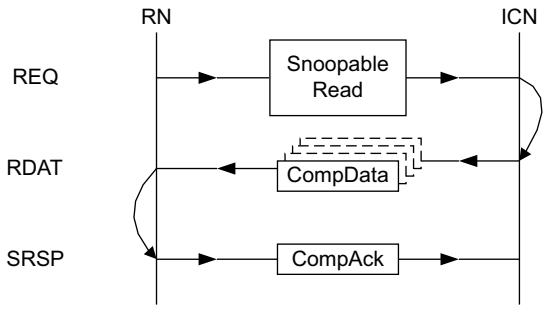


Figure 2-3 Snoopable Read structure without Direct Data Transfer

The Request/Response rules are:

- CompData must only be sent by the Completer after the associated request is received.
- If the data being sent is received from either a Slave or snooped Agent then the ICN is permitted to forward the CompData packets as soon as it receives the first packet from the Slave or snooped Agent.
- CompAck can be sent as soon as the first Data packet of read data has been received.

  Note

  Prior to CHI Issue C, CompAck must not be sent until all Data packets of read data have been received.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ReadNoSnp, ReadOnce, ReadOnceCleanInvalid, ReadOnceMakeInvalid

A ReadNoSnp transaction is used to carry out a read when the snooping of other masters is not required.

Data obtained by ReadNoSnp either comes directly from the Slave Node or via the interconnect.

A ReadOnce, ReadOnceCleanInvalid, and ReadOnceMakeInvalid transaction is used to carry out a read when the snooping of other masters is required but the Requester is not going to allocate the cache line in its own cache.

Note

ReadOnce, ReadOnceCleanInvalid, and ReadOnceMakeInvalid obtain a snapshot of the coherent data value. If a Requester holds this value in a local buffer or cache, the data value will no longer be coherent.

In the remainder of this section ReadOnce* represents the three transaction types, ReadOnce, ReadOnceCleanInvalid, and ReadOnceMakeInvalid.

Data obtained by ReadOnce* either comes directly from the Slave Node or a peer Request Node, or via the interconnect.

ReadNoSnp and ReadOnce* transactions can optionally have an ordering requirement. For transactions that require ordering, the Home must ensure that a transaction is observable before taking any action that could make a later ordered transaction observable.

ReadNoSnp and ReadOnce* transactions can optionally set the ExpCompAck field, indicating that the transaction will include a CompAck response. The use of a CompAck response is not functionally required for ReadNoSnp and ReadOnce* transactions, as the RN issuing the transaction will not hold a copy of the cache line. However, use of CompAck permits the use of DMT and separate Comp and Data responses in some cases.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ReadNoSnp and ReadOnce structure with DMT*

The progress of the ReadNoSnp and ReadOnce* transaction with DMT is as follows:

1. Requester sends a request with the ReadNoSnp or ReadOnce* opcode on the REQ channel.
2. If the request Order field indicates that ordering is required, then a ReadReceipt response must be returned on the CRSP channel when order has been established.
3. ICN sends a ReadNoSnp request to SN on the REQ channel.
4. SN returns a ReadReceipt response to ICN on the CRSP channel if the ReadNoSnp request has Order[1:0] set to `0b01`.
5. SN, as Completer, returns the read data and any associated transaction response with the CompData opcode directly to the Requester on the RDAT channel.

   The read data can be sent using multiple transfers. See *Data transfer* on page 2-115.
6. If the transaction request ExpCompAck bit is set, the Requester must return an acknowledgement, using the CompAck opcode on the SRSP channel to indicate that the transaction has completed.

   CompAck can be sent as soon as the first Data packet of read data has been received.

   Note

   Prior to CHI Issue C, CompAck must not be sent until all Data packets of read data have been received.

Figure 2-4 shows the transaction structure.

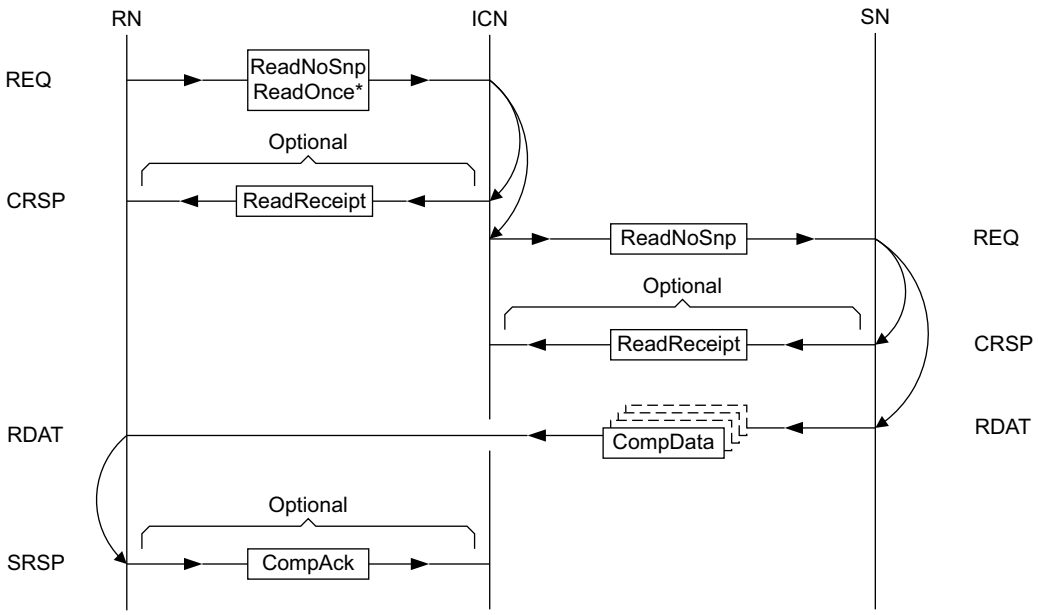


Figure 2-4 ReadNoSnp and ReadOnce* DMT structure

The life time at Home of ReadNoSnp and ReadOnce* transactions that use DMT can be reduced by using the ReadReceipt response from the Slave as a read received acknowledgment to deallocate the transaction at Home. The ReadReceipt from SN, marked as optional in Figure 2-4, becomes required when used to early deallocate the request at Home when used instead of CompAck from the Requester, which typically comes later, to deallocate the request. See *ReadOnce* and ReadNoSnp with early Home deallocation* on page 5-206 for an example of this type of flow. See Table 2-6 on page 2-49 for the relationship between DMT and the use of Read Received Ack, in the form of a ReadReceipt, and CompAck and Ordering requirements in ReadOnce* and ReadNoSnp transactions.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The following requirements apply to this type of lifetime reduction transaction:

- Home must set Order[1:0] to the value `0b01` in the Read request to the Slave Node.
- For a Request with Order[1:0] set to the value `0b01` the Slave must send a ReadReceipt to acknowledge the Read request when it can guarantee that the request is accepted and that it will not send a RetryAck response.
- Home is permitted to deallocate the request after receiving the ReadReceipt without waiting for a CompAck if the requests from RN do not have an ordering requirement.
- Home is permitted to receive CompAck even after the request is deallocated.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ReadOnce structure with DCT*

The progress of the ReadOnce* transaction with DCT is as follows:

1. Requester sends a request with the ReadOnce* opcode on the REQ channel.
2. ICN sends a Snp[*]Fwd request to RN-F on the SNP channel.
3. RN-F, as Completer, forwards the read data and any associated transaction response to RN with the CompData opcode on the DAT channel.

   The read data can be sent using multiple transfers. See *Data transfer* on page 2-115.
4. RN-F also forwards a SnpRespFwded response to ICN on the SRSP channel to indicate that read data was forwarded to the Requester. Alternatively, the response to the ICN can include data and will be a SnpRespDataFwded response.
5. If the transaction request ExpCompAck bit is set, Requester must return an acknowledgement, using the CompAck opcode on the SRSP channel to indicate that the transaction has completed.

   CompAck can be sent as soon as the first Data packet of read data has been received.

   Note

   Prior to CHI Issue C, CompAck must not be sent until all Data packets of read data have been received.

Figure 2-5 shows the transaction structure.

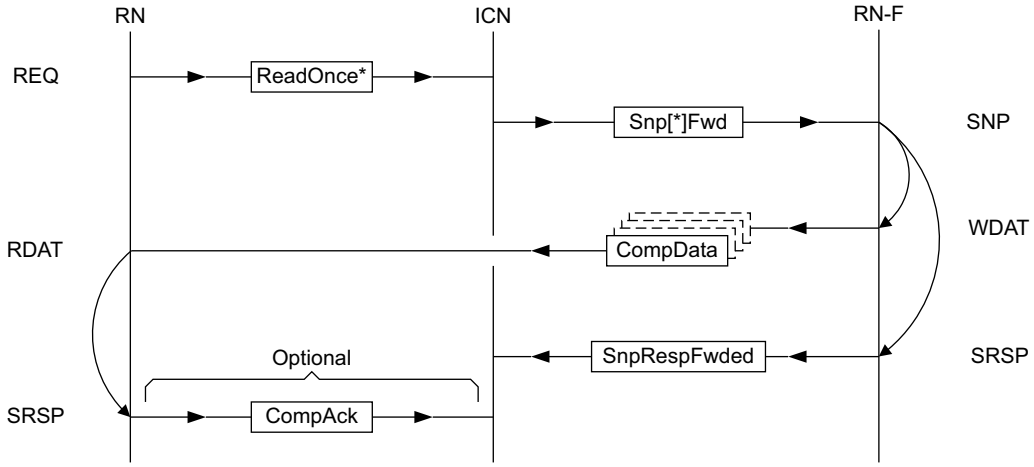


Figure 2-5 ReadOnce* DCT structure

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ReadNoSnp and ReadOnce structure without Direct Data Transfer*

This section shows the Read transaction structure without DMT or DCT.

The progress of a ReadNoSnp and a ReadOnce* transaction without Direct Data Transfer, from the Requester perspective, is identical to a ReadNoSnp and ReadOnce* transaction with Direct Data Transfer and is as follows:

1. Requester sends a request with the ReadNoSnp or ReadOnce* opcode on the REQ channel.
2. If the request Order field indicates that ordering is required then a ReadReceipt response must be returned on the CRSP channel when order has been established.
3. Completer returns the read data and any associated transaction response with the CompData opcode on the RDAT channel.

   The read data can be sent using multiple transfers. See *Data transfer* on page 2-115.
4. If the transaction request ExpCompAck bit is set, Requester must return an acknowledgement, using the CompAck opcode on the SRSP channel to indicate that the transaction has completed.

   CompAck can be sent as soon as the first Data packet of read data has been received.

   Note

   Prior to CHI Issue C, CompAck must not be sent until all Data packets of read data have been received.

Figure 2-6 shows the transaction structure.

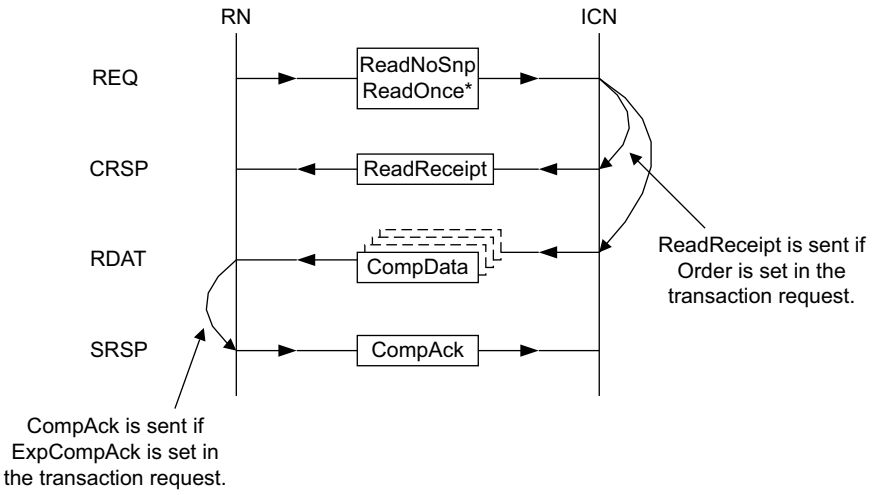


Figure 2-6 ReadNoSnp and ReadOnce* structure without Direct Data Transfer

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The Request/Response rules are:

- ReadReceipt, if part of the transaction:
  - Must only be sent by the Completer after the associated request is received.
  - Typically is sent by the Completer before CompData. However it is permitted to be sent after CompData.
  - Typically is received by the Requester before CompData. However it is permitted to be received after CompData.
- CompData must only be sent by the Completer after the associated request is received.
- If the data being sent is received from either a Slave or snooped Agent then the ICN is permitted to forward the CompData packets as soon as it receives the first packet from the Slave or snooped Agent.
- If CompAck is part of the transaction:
  - The Requester must only send CompAck after at least one CompData packet is received.
  - The Requester is permitted, but not required, to wait for the ReadReceipt before sending CompAck.
  - The Completer is not permitted to wait for the CompAck before sending the ReadReceipt.

Summary of impact of ordering and CompAck rules

Table 2-6 shows the use of DMT and DCT, with different combinations of ordering and CompAck for ReadNoSnp and ReadOnce* from an RN.

Table 2-6 Permitted DMT and DCT for ReadNoSnp and ReadOnce* from an RN

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 the Request Accepted response from SN to ensure the request to SN is not given a RetryAck response.
	1	Y	Y	Home does not need to be notified of transaction completion. For DMT, Home can ensure the request to SN is not given a RetryAck response by either obtaining the Request Accepted response from SN or waiting for the CompAck response.
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 SnpRespDataFwd snoop response to determine transaction completion.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Reads with separate Non-data and Data-only responses

This specification supports a separate Non-data response from Home and a Data-only response from either Home or Slave for all Read* transactions.

Note

Separate Non-data and Data-only response is not supported prior to CHI Issue C.

This feature does not apply to:

- Atomic transactions.
- Exclusive transactions.
- Ordered ReadNoSnp and ReadOnce* transactions without CompAck.

The progress of a Read* with separate Comp and Data responses is as follows:

- Comp from Home and Data from Slave:
  1. Requester sends a request with the Read* opcode on the REQ channel.
  2. Home sends a RespSepData response to the Requester on the CRSP channel.
     - This tells the Requester that the transaction has been ordered at Home with respect to requests to the same address, and that Data for the transaction will be provided in a separate Data response.
  3. Requester sends CompAck acknowledgement to ICN on the SRSP channel.

     For ordered transactions, CompAck must be sent only after the first Data response packet has been received.
  4. ICN sends ReadNoSnpSep request to the SN on the REQ channel.
     - This tells the Slave, as Completer, to send a DataSepResp response.
  5. SN sends ReadReceipt to Home on the CSRP channel.
     - This tells the Home that the request to the Slave will be completed and ensures that the SN will not respond with RetryAck for that request.
  6. SN, as Completer, sends DataSepResp to the Requester returning the read data.

     For ReadOnce and ReadNoSnp requests with an Order requirement, the Home determines that the request has been completed by receiving a CompAck response.
- Comp and Data from Home:
  1. Requester sends a request with the Read* opcode on the REQ channel.
  2. Home sends a RespSepData response to the Requester on the CRSP channel.
     - This tells the Requester that the transaction has been ordered at Home with respect to the same address, and that Data for the transaction will be provided in a separate Data response.
  3. Home, as Completer, sends DataSepResp to the Requester returning the read data
  4. Requester sends CompAck acknowledgement to Home on the SRSP channel.

Figure 2-7 on page 2-51 shows the transaction structure.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

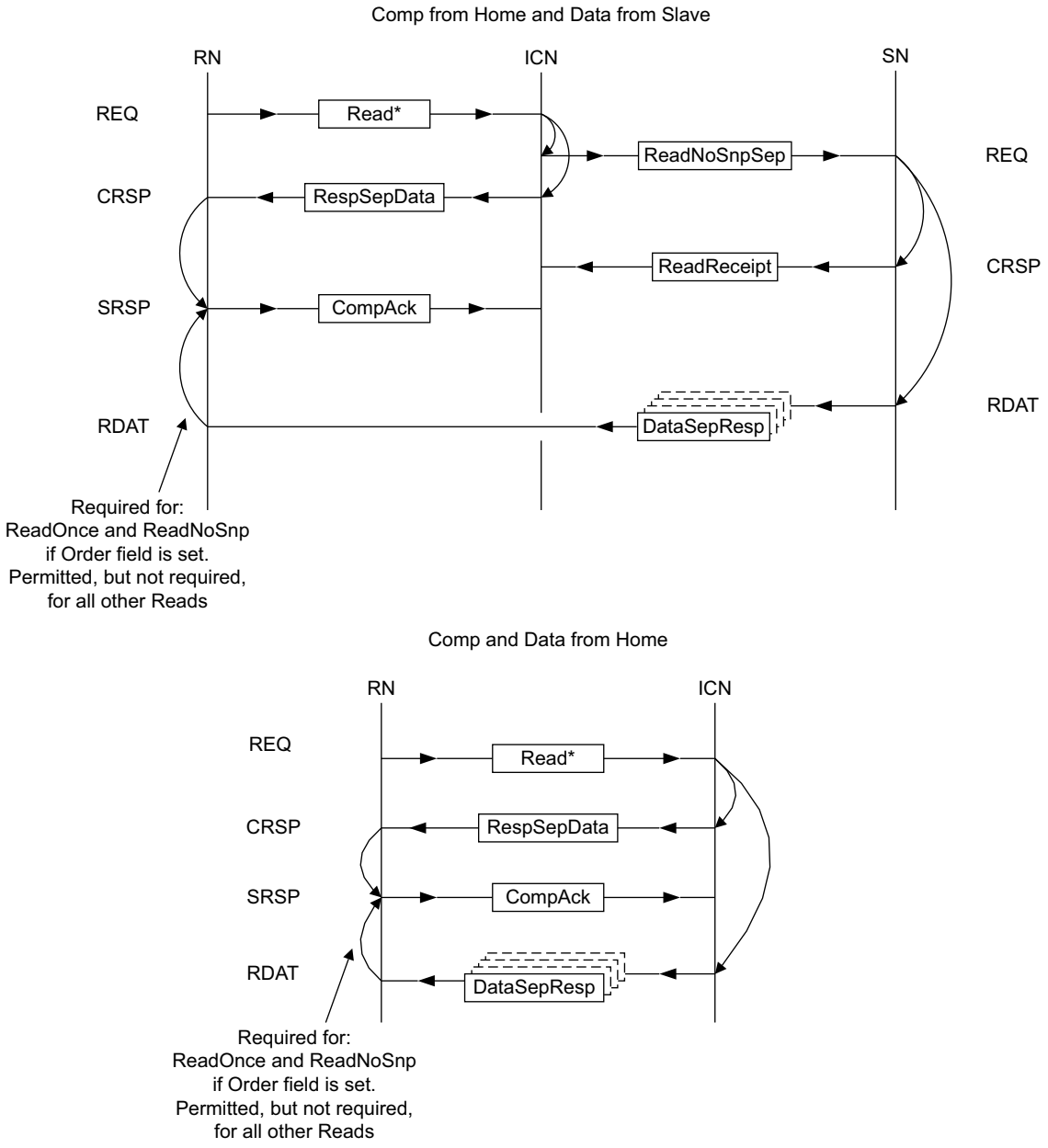


Figure 2-7 Read* DMT structure with separate Non-data and Data-only responses

Note

As Figure 2-7 shows, the CompAck acknowledgement sent by the Requester to ICN is permitted, but not required, to wait for DataSepResp except in the case of an ordered ReadOnce and ReadNoSnp request with CompAck, where it is required to wait for DataSepResp.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The Request and Response rules are:

- DataSepResp and RespSepData must only be sent by the Completer after the associated request is received. RespSepData is permitted from Home only. DataSepResp is permitted to be sent by the Slave or Home.
- CompAck acknowledgement must only be sent by the Requester after the RespSepData response is received:
  - For Non-ordered requests, it is permitted, but not required, for a Requester to wait for any or all data to be received before sending a CompAck acknowledgement.
  - For ReadOnce and ReadNoSnp requests with an Order requirement, the Requester is required to wait for at least one Data response to be received before sending a CompAck acknowledgement.
- It is required that the Completer must not wait for CompAck before sending all Data packets.
- ReadNoSnpSep must only be sent by the Home to the Slave after the associated request is received, and Snoop responses are received for all snoops that are required to be sent.
- The Slave must send the ReadReceipt response to Home only after receiving ReadNoSnpSep and ensuring that it will not respond with RetryAck for that request.
- The ReadReceipt from the Slave is sufficient to indicate to Home that the request to the Slave will be completed.

  A separate Comp response from the Slave to Home is not required and must not be sent. For ReadOnce and ReadNoSnp requests with an Order requirement, the Home determines that the request has been completed by receiving a CompAck response.
- For ReadOnce and ReadNoSnp transactions, with an Order requirement, that include separate Comp and Data responses, the Home must not send a ReadReceipt response. The RespSepData response from Home includes an implied ReadReceipt.

In all cases, where a separate Data response and Home response can be used, it is also permitted to use a combined CompData response.

Table 2-7 on page 2-53 shows the expected and permitted use of separate Non-data and Data-only responses, or the combined CompData response, with different combinations of order and ExpCompAck values for ReadNoSnp and ReadOnce*.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 2-7 Use of separate Comp and Data responses with different combinations of Order and ExpCompAck for ReadNoSnp and ReadOnce*

Transaction	Order[1:0]	ExpCompAck	Notes
ReadNoSnp ReadOnce*	00	0	The options to provide responses for the transaction are: • A single CompData response. This can be: — From the Home. — From a snooped RN-F, when DCT is used. — From the Slave, when DMT is used. Home must request a ReadReceipt from the Slave. • A separate RespSepData and DataSepResp response. RespSepData from the Home. DataSepResp can be: — From the Home. — From the Slave. Home must request a ReadReceipt from the Slave. A ReadReceipt from Home to Requester is not used. A CompAck from Requester to Home is not used.
		1	The options to provide responses for the transaction are: • A single CompData response. This can be: — From the Home. — From a snooped RN-F, when DCT is used. — From the Slave, when DMT is used. Home can optionally request a ReadReceipt from the Slave. • A separate RespSepData and DataSepResp response. RespSepData is always from the Home. DataSepResp can be: — From the Home. — From the Slave. Home can optionally request a ReadReceipt from the Slave. A ReadReceipt from Home to Requester is not used. A CompAck is sent from Requester to Home only after: • CompData is received. • RespSepData is received. It is permitted to also wait for the corresponding DataSepResp response before sending CompAck.
	01	-	This is not permitted for transactions from Requester to Home.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 2-7 Use of separate Comp and Data responses with different combinations of Order and ExpCompAck for ReadNoSnp and ReadOnce* (continued)

Transaction	Order[1:0]	ExpCompAck	Notes
ReadNoSnp ReadOnce*	10 11	0	The options to provide responses for the transaction are: • A single CompData response is provided. This can be: — From the Home. — From a snooped RN-F, when DCT is used. — DMT is not permitted. • A separate RespSepData and DataSepResp response is not permitted. A ReadReceipt from Home to Requester is used. A CompAck from Requester to Home is not used.
		1	The options to provide responses for the transaction are: • A single CompData response. This can be: — From the Home. — From a snooped RN-F, when DCT is used. — From the Slave, when DMT is used. In this case, the Home can optionally request a ReadReceipt from the Slave. • A separate RespSepData and DataSepResp response. RespSepData is always from the Home. DataSepResp can be: — From the Home. — From the Slave. Home can optionally request a ReadReceipt from the Slave. A ReadReceipt from Home to Requester depends on: • If a single CompData response is used, then a ReadReceipt is sent. • If a separate RespSepData and DataSepResp response is used, then no ReadReceipt is sent. RespSepData is used to indicate the same meaning as ReadReceipt. A CompAck is sent from Requester to Home only after: • CompData is received. • Both RespSepData response and DataSepResp response are received.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Dataless

The Dataless transactions are:

- CleanUnique.
- MakeUnique.
- Evict.
- StashOnceUnique.
- StashOnceShared.
- CleanShared.
- CleanSharedPersist.
- CleanInvalid.
- MakeInvalid.

These transactions serve a number of functions such as:

- Obtaining permission to store to a cache.
- Performing cache maintenance.
- Updating the state of a snoop filter.
- Moving data closer to the point of expected future use.

The progress of a Dataless transaction from the Requester perspective is as follows:

1. The Requester sends a Dataless request on the REQ channel.
2. The Completer returns a Comp response on the CRSP channel. Only a single response is given for Dataless transactions.
3. If the transaction request ExpCompAck field is set, the Requester must return an acknowledgement that the transaction has completed with the CompAck opcode on the SRSP channel:
   - ExpCompAck must be asserted for:
     - CleanUnique.
     - MakeUnique.
   - ExCompAck must not be asserted for:
     - Evict.
     - StashOnceUnique.
     - StashOnceShared.
     - CleanShared.
     - CleanSharedPersist.
     - CleanInvalid.
     - MakeInvalid.

Figure 2-8 on page 2-56 shows the transaction structure.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

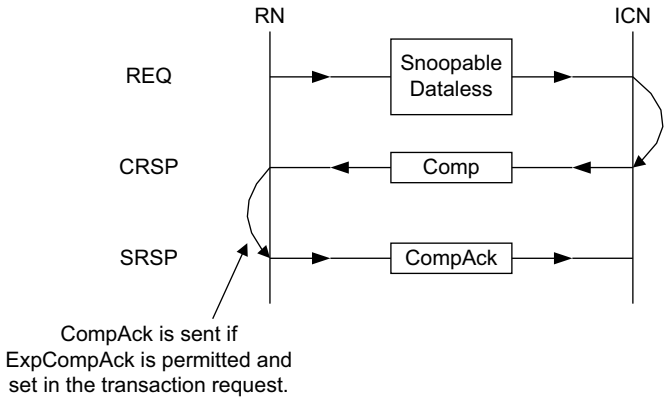


Figure 2-8 Snoopable Dataless transaction structure

The Request/Response rules are:

- Comp must only be sent by the Completer after the associated request is received.
- CompAck must only be sent by the Requester after the associated Comp is received.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Writes

Write transactions include the following:

- WriteNoSnp.
- WriteUnique.
- WriteBack.
- WriteCleanFull.
- WriteEvictFull.

WriteNoSnp

The WriteNoSnp transactions are:

- WriteNoSnpPtl, WriteNoSnpFull.

A WriteNoSnp transaction is used to carry out a store where the snooping of other masters is not required.

The progress of the WriteNoSnp transaction is as follows:

1. The Requester sends a request with the WriteNoSnpPtl or WriteNoSnpFull opcode on the REQ channel.
2. The Completer has one of the following options:
   - Return separate responses:
     - Return a DBIDResp response that provides a data buffer identifier to indicate that it can accept the write data for the transaction.
     - Provide a Comp response to indicate that the transaction is observable by other Requesters.

     Both responses are sent on the CRSP channel.
   - Return a single combined CompDBIDResp response to indicate:
     - It can accept the write data for the transaction.
     - The transaction is observable by other Requesters.

     The combined response is sent on the CRSP channel.
3. The Requester sends the write data and any associated byte enables with the NonCopyBackWrData opcode on the WDAT channel. The write data can be sent using multiple transfers. See *Data transfer* on page 2-115.

Figure 2-9 on page 2-58 shows the transaction structure options.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

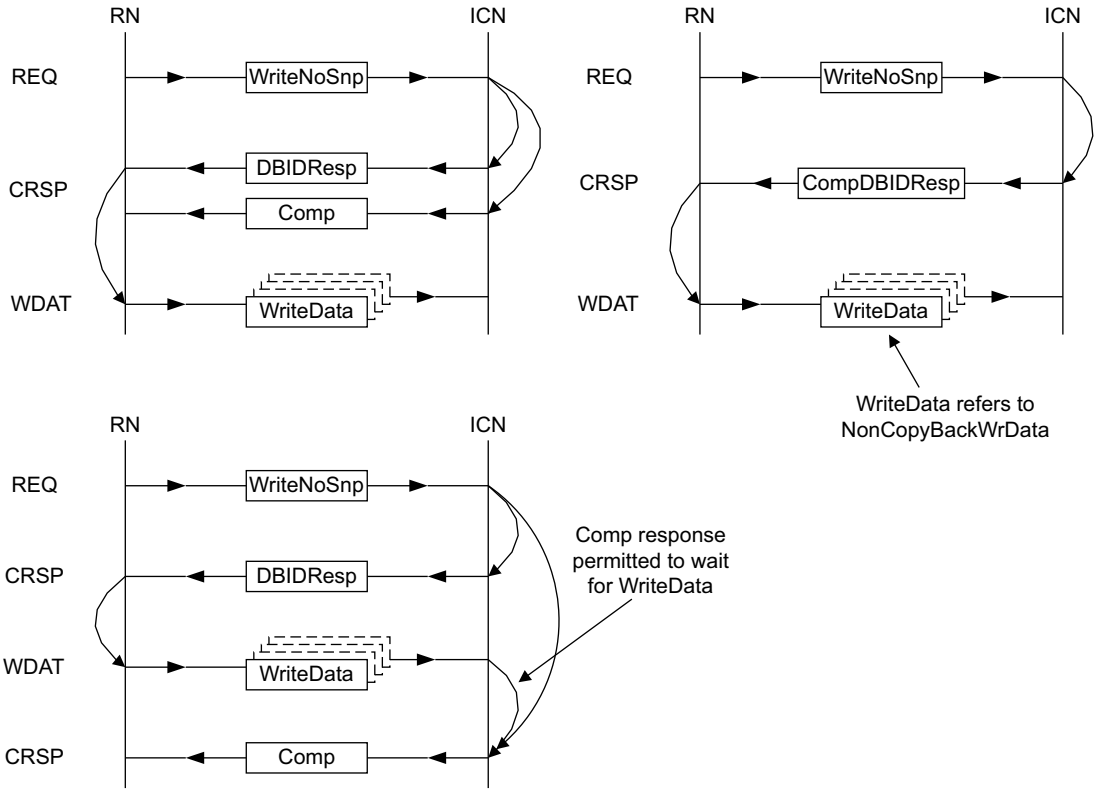


Figure 2-9 WriteNoSnp transaction structure options

The Request/Response rules are:

- The separate DBIDResp and Comp, or the combined CompDBIDResp, must only be sent by the Completer after the associated request is received.
- WriteData must only be sent by the Requester after either DBIDResp or CompDBIDResp is received.
- If the DBIDResp and Comp responses are sent separately:
  - The Requester must send the write data after it has received the DBIDResp response. The Requester must not wait to receive the Comp response before the write data is sent.
  - Typically the DBIDResp is sent by the Completer before Comp. However it is permitted for DBIDResp and Comp to be sent in any order.
  - Typically the DBIDResp is received by the Requester before Comp. However it is permitted for DBIDResp and Comp to arrive in any order.
- The Completer is permitted to wait for the WriteData before sending the Comp response.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

WriteUnique

The WriteUnique transactions are:

- WriteUniquePtl.
- WriteUniqueFull.
- WriteUniquePtlStash.
- WriteUniqueFullStash.

In the remainder of this section WriteUnique represents the four transaction types.

WriteUnique transactions are used to perform a store to a location when the snooping of other Snoopable Requesters (RN-Fs) might be required to obtain permission to store.

There is one optional behavior associated with WriteUnique transactions. The behavior is determined by the Order field in the transaction request. See *Streaming Ordered WriteUnique transactions* on page 2-103 for details on the use of CompAck to force the order in which requests are observed.

The progress of the WriteUnique transaction, from the Requesters perspective, is as follows:

1. The Requester sends a request with the WriteUnique opcode on the REQ channel.
2. The Completer has one of the following options:
   - Return separate responses:
     - Return a DBIDResp response that provides a data buffer identifier indicating that it can accept the write data for the transaction.
     - Return a Comp response to indicate that the transaction is observable by other Requesters.

     Both responses are sent on the CRSP channel.
   - Return a single combined CompDBIDResp response to indicate:
     - It can accept the write data for the transaction.
     - The transaction is observable by other Requesters.

     The combined response is sent on the CRSP channel.
3. The Requester sends the write data and any associated byte enables with the NonCopyBackWrData opcode on the WDAT channel. The write data can be sent using multiple transfers. See *Data transfer* on page 2-115.
4. If the ExpCompAck field is set in the transaction request, then the transaction completes with a CompAck transaction acknowledge. Opportunistically, CompAck can be combined with the Data response and sent as NCBWrDataCompAck.

Figure 2-10 on page 2-60 shows the transaction structures.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

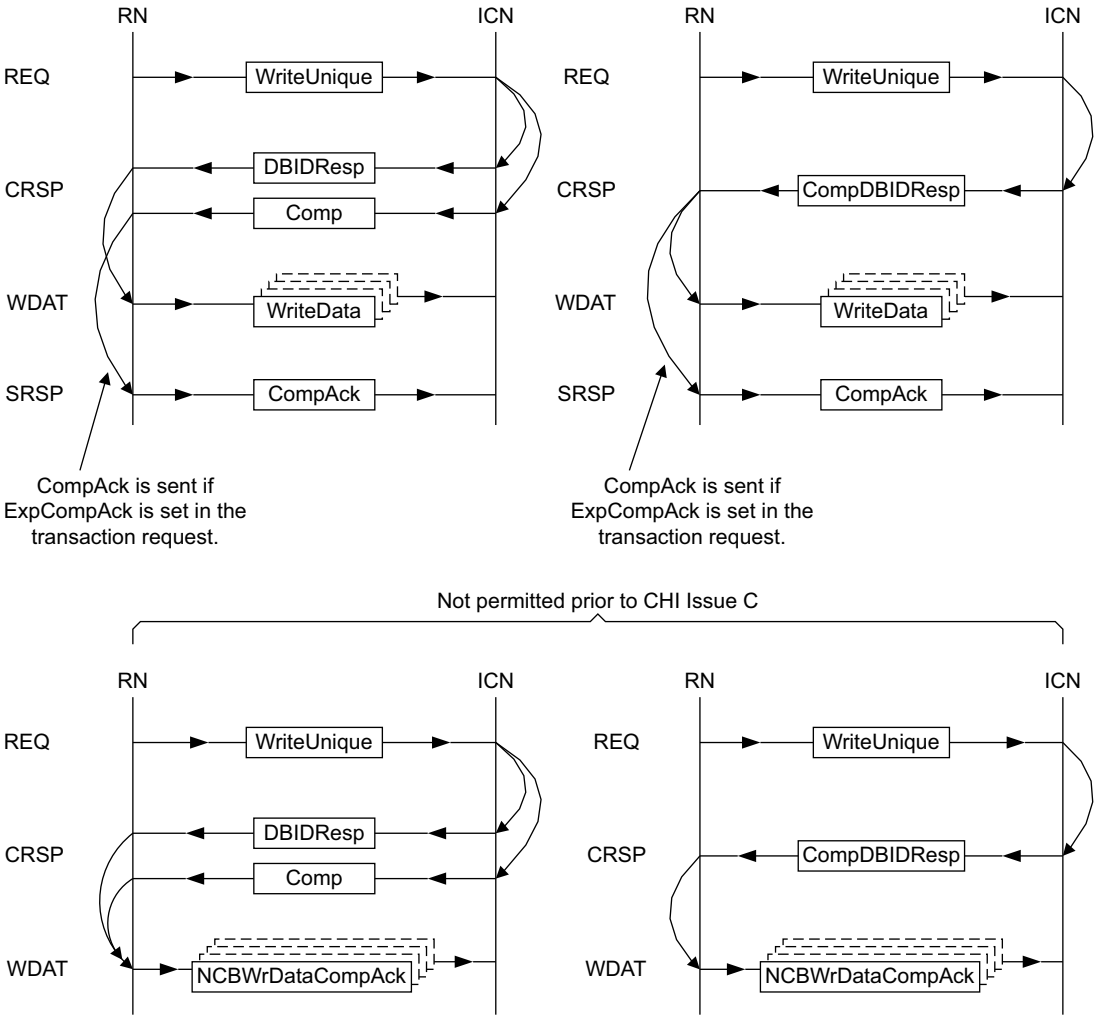


Figure 2-10 WriteUnique transaction structure options

As Figure 2-10 shows, the RN is permitted to combine CompAck and WriteData into the single message NCBWrDataCompAck.

Note

Prior to CHI Issue C, the RN must not combine CompAck and WriteData into the single NCBWrDataCompAck message.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The Request/Response rules are:

- The separate DBIDResp and Comp, or the combined CompDBIDResp, must only be sent by the interconnect after the associated request is received.
- WriteData must only be sent by the Requester after either DBIDResp or CompDBIDResp is received.
- If the DBIDResp and Comp responses are sent separately:
  - The Requester must send the write data after it has received the DBIDResp response. The Requester must not wait to receive the Comp response before the write data is sent.
  - Typically the DBIDResp is sent by the Completer before Comp. However it is permitted for DBIDResp and Comp to be sent in any order.
  - Typically the DBIDResp is received by the Requester before Comp. However it is permitted for DBIDResp and Comp to arrive in any order.
  - The Requester is permitted to wait for the DBIDResp response before sending a combined NCBWrDataCompAck response or separate CompAck and NCBWrData responses.
  - The Completer must not wait for WriteData before sending Comp.
- If the CompAck acknowledge is part of the transaction then CompAck must only be sent by the Requester after either the Comp or CompDBIDResp response is received.
  - The Requester is permitted to send the WriteData and CompAck in any order.
- NCBWrDataCompAck can be sent after receiving either:
  - Comp and DBIDResp.
  - CompDBIDResp.

CopyBack

The CopyBack transactions are:

- WriteBackPtl, WriteBackFull.
- WriteCleanFull.
- WriteEvictFull.

CopyBack transactions, except WriteEvictFull, are used to update main memory or a downstream cache for a coherent location.

A WriteEvictFull transaction is used to update only a downstream cache for a coherent location.

A WriteEvictFull must not propagate beyond its Snoop domain.

The progress of a CopyBack transaction is as follows:

1. The Requester sends a CopyBack request on the REQ channel.
2. The Completer returns a single combined CompDBIDResp response on the CRSP channel to indicate:
   - It can accept the write data for the transaction.
   - This request will complete before any snoop to the same address is received.
3. After the Requester has received the CompDBIDResp response it sends the write data, and any associated byte enables, with the CopyBackWrData opcode on the WDAT channel. The write data can be sent using multiple transfers. See *Data transfer* on page 2-115.

Figure 2-11 on page 2-62 shows the transaction structure.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

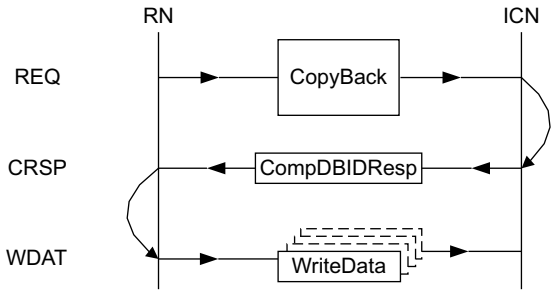


Figure 2-11 CopyBack transaction structure

The Request/Response rules are:

- CompDBIDResp must only be sent by the Completer after the associated request is received.
- WriteData must only be sent by the Requester after the CompDBIDResp response is received.

Atomics

Atomic transactions can be classified in two categories based on their transaction structure:

- The following transaction returns only a completion response:
  - AtomicStore.
- The following transactions return Data with a completion response:
  - AtomicLoad.
  - AtomicSwap.
  - AtomicCompare.

Figure 2-12 shows the structure of Atomic transactions at the Requester interface.

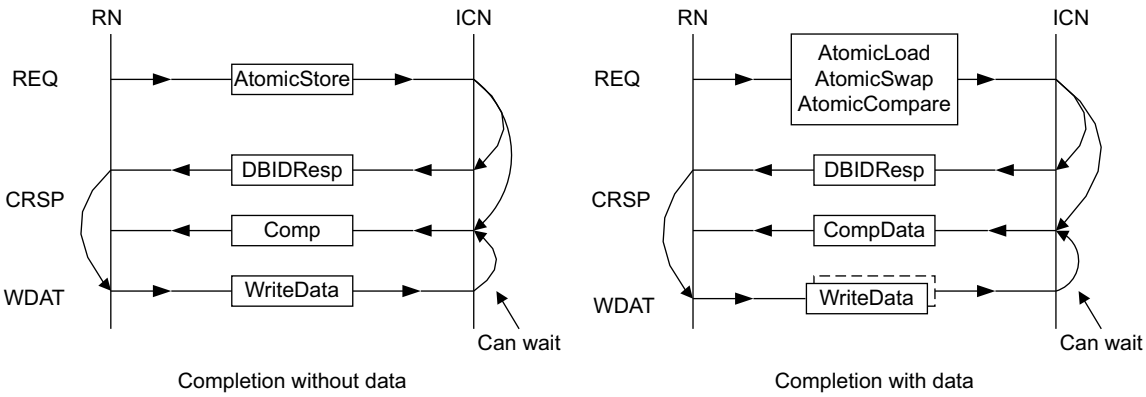


Figure 2-12 Atomic transaction structure

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The progress of an Atomic transaction is as follows:

- The Requester issues the request on the REQ channel.
- The Completer returns a DBIDResp response, on the CRSP channel, to indicate that it can accept the WriteData associated with the transaction:
  - For the AtomicStore transaction, it is permitted to give either a separate DBIDResp and Comp response or a combined CompDBIDResp response.
  - For the AtomicLoad, AtomicSwap, and AtomicCompare transaction, a CompDBIDResp response is not permitted because the completion is included in the CompData response.
- When sending a separate Comp and DBIDResp response:
  - The Completer must not wait for Data from the Requester before sending the DBIDResp response.
  - The Completer is permitted to wait for Data from the Requester before sending the Comp response.

    Note

    Sending a separate Comp, and delaying it until Data is received and the execution of the atomic operation completes, permits the Completer to convey an error in the data received or in the execution of the atomic operation. The precise nature of this error reporting is IMPLEMENTATION DEFINED.
- When sending a combined response, the Completer must not wait for Data from the Requester before sending the CompDBIDResp response.
- On receiving the DBIDResp or CompDBIDResp response the Requester must send the transaction WriteData on the WDAT channel:
  - The Requester must not wait for a CompData or Comp response to send the transaction data.
- The Completer is permitted, when applicable:
  - To give the read data response at any point after receiving the request.
  - To wait until it has received all write data associated with the transaction before giving the read data response.

Note

An advantage of an early DBIDResp response is that it permits pipelining of Atomic transactions to distant locations. In addition, by separating DBIDResp from Comp, the Completer has an opportunity to signal an error to the Requester if the received data is in error, or the atomic operation has an error.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Self-snoop in Atomic transactions

This specification permits self-snooping of the Requester in Atomic transactions. The optional self-snoop is not shown in the figure. Self-snooping is controlled by the SnoopMe bit value in the Atomic request. See *SnoopMe* on page 12-326. The Request-Response rules for self-snooping in Atomic transactions are:

- An RN that does not invalidate its own cached copy of the cache line before sending an Atomic request must rely on self-snooping to:
  - Invalidate its own cached copy of the cache line.
  - Obtain a copy of the cache line if Dirty.
- The Home Node:
  - Must send a snoop to the Requester if the SnoopMe bit is set and Home determines that the cache line is present at the Requester.
  - Is permitted, but not required, to send a snoop to the Requester if the SnoopMe bit is set, and Home determines that the cache line is not present at the Requester.
  - Is permitted, but not required, to snoop the Requester when the SnoopMe bit is not set in an Atomic request.
  - Is expected to send a SnpUnique in response to an Atomic request, but is permitted to send a SnpCleanInvalid.

Note

An RN is permitted:

- To send a CopyBack request while the Atomic request to the same address with SnoopMe asserted is in progress.
- To issue an Atomic request with SnoopMe asserted while a CopyBack request to the same address is in progress.

Miscellaneous request types

Misc transactions include the following:

- DVM.
- PrefetchTgt.

DVM

The DVM transaction is DVMOp.

A DVM transaction is used to send a *Distributed Virtual Memory* (DVM) operation.

The progress of the DVM transaction is as follows:

1. The Requester sends a request with the DVMOp opcode on the REQ channel.
2. The Completer returns a DBIDResp response that provides a data buffer identifier indicating that it can accept the write data for the transaction.
3. The Requester sends the write data for the DVM transaction, with the NonCopyBackWrData opcode, on the WDAT channel. Only a single data transfer occurs for a DVM transaction.
4. The Completer returns a Comp response on the CRSP channel.

Figure 2-13 on page 2-65 shows the transaction structure.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

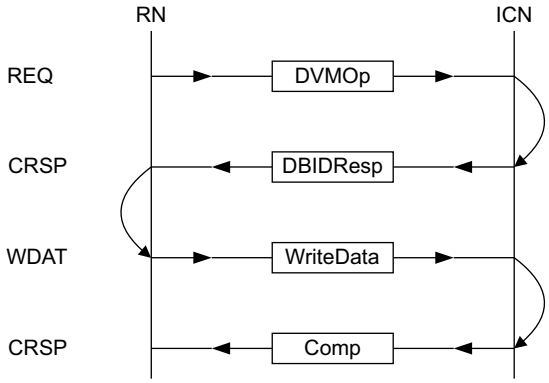


Figure 2-13 DVM transaction structure

The Request/Response rules are:

- DBIDResp must only be sent by the Completer after the associated request is received.
- WriteData must only be sent by the Requester after the DBIDResp response is received.
- Comp must only be sent by the Completer after the data transfer is received.

PrefetchTgt

A Request to a shareable memory address sent from a Request Node directly to a Slave Node. The request can be used by the Slave Node to fetch and buffer data from main memory in anticipation of a subsequent Read request to the same location.

The progress of a PrefetchTgt transaction is as follows:

- The Requester sends a PrefetchTgt request on the REQ channel. The PrefetchTgt transaction does not include a response.

Figure 2-14 shows the transaction structure.

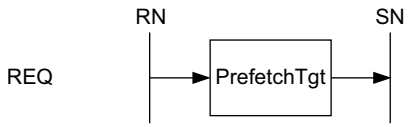


Figure 2-14 PrefetchTgt transaction structure

The Request/Response rules are:

- The Requester can deallocate the request as soon as the request is sent.
- The request must be accepted by the SN without any dependency and no Retry is permitted.
- The SN can drop the Request without taking any action.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.3.2 Transaction Retry sequence

With the exception of PrefetchTgt, all Request transactions, as described in *Request transaction structure* on page 2-39, can begin with a Retry sequence.

Retry sequence

Request transactions are first sent without a *Protocol Credit* (P-Credit). If the transaction cannot be accepted at its Completer, then a RetryAck response must be given that indicates that the transaction has not been accepted and can be sent again when an appropriate credit is provided. When a transaction is sent a second time, with a credit, it is guaranteed to be accepted.

For further details on the Retry process and the use of credits see *Request Retry* on page 2-126.

The transaction Retry sequence is as follows:

1. The Requester sends a request, without a P-Credit, on the REQ channel.
2. The Completer provides a RetryAck response on the CRSP channel.
3. The Completer provides a PCrdGrant response on the CRSP channel, when appropriate, to indicate that a credit is available to re-send the transaction.
4. The Requester sends the transaction again, with a credit, on the REQ channel.

Figure 2-15 shows the RetryAck sequence.

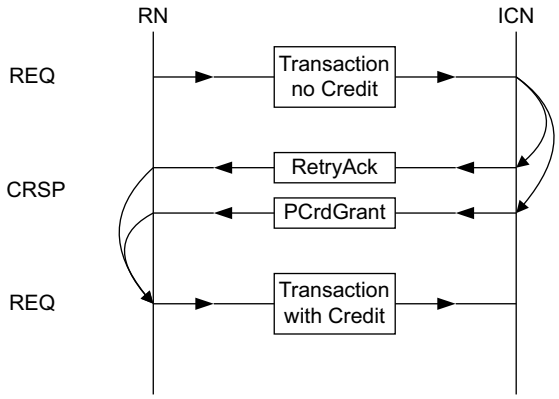


Figure 2-15 Transaction Retry sequence

The transaction Retry sequence rules are:

- RetryAck must only be sent by the Completer after the associated request is received.
- PCrdGrant must only be sent by the Completer after the associated request is received.
- RetryAck is typically sent by the Completer before PCrdGrant. However, it is permitted to send RetryAck after PCrdGrant.
- RetryAck is typically received by the Requester before PCrdGrant. However, it is permitted to receive RetryAck after PCrdGrant.
- The transaction with credit must only be sent by the Requester after both the RetryAck response and an appropriate PCrdGrant response are received.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Not retrying a transaction

The protocol supports deciding not to resend the transaction between the point that it receives a RetryAck and the point that it is resent using a credit. The sequence is identical to the transaction Retry sequence that Figure 2-15 on page 2-66 shows, except that the final transaction with credit is sent as a PCrdReturn transaction. This acts as a null transaction and returns the credit to the Completer.

The sequence and rules are identical to those for a Retry sequence.

Figure 2-16 shows the cancelled transaction sequence.

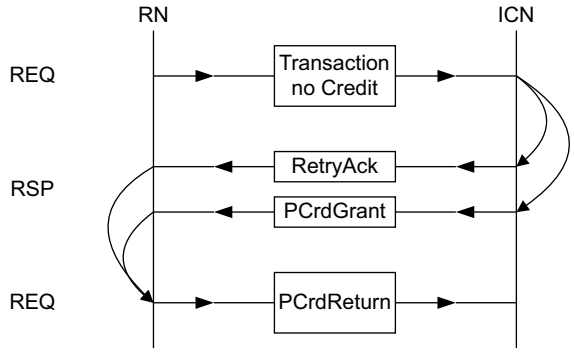


Figure 2-16 Cancelled transaction sequence

2.3.3 Snoop transactions

Snoop transactions are sent from the interconnect to a Request Node:

- An RN-F is fully coherent and is required to accept all Snoop transactions.
- An RN-D supports receiving only SnpDVMOp transactions.
- An RN-F and RN-D must respond to received snoop requests, except for DVMOp(Sync), in a timely manner, without creating any dependency on completion of outstanding requests.

There are several options for the transaction structure of a snoop:

- Snoop with response to Home.
- Snoop with Data to Home.
- Snoop with Data return to Requester and response to Home.
- Snoop with Data return to Requester and Data to Home.
- Snoop DVM operation and response to Misc Node.

A snoop transaction can also be used to stash data at the Snoopee. The options for the transaction structure of a Stash type snoop are:

- Stashing snoop with Data from Home.
- Stashing snoop with Data using DMT.

Note

Figures relating to Snoop transactions show the snooped Request Node (RN) on the right, and the interconnect (ICN) on the left. This is consistent with the ordering of the Request/Snoop process.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Snoop with response without Data to Home

The progress of a Snoop transaction with response to Home is as follows:

1. The interconnect provides a snoop request on the SNP channel that can be any Snoop transaction supported by the RN.
2. The RN returns the SnpResp snoop response on the SRSP channel.

Figure 2-17 shows the transaction structure.

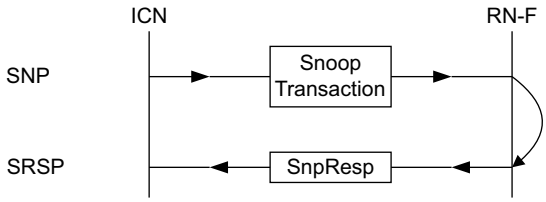


Figure 2-17 Snoop transaction structure with response to Home

The snoop with response to Home rules are:

- SnpResp must only be sent by the RN after the associated Snoop request is received.

Snoop with response with Data to Home

The progress of a Snoop transaction with Data to Home is as follows:

1. The interconnect provides a snoop request on the SNP channel. This can be one of the following Snoop transactions:
   - SnpOnceFwd, SnpOnce.
   - SnpCleanFwd, SnpClean.
   - SnpNotSharedDirtyFwd, SnpNotSharedDirty.
   - SnpSharedFwd, SnpShared.
   - SnpUniqueFwd, SnpUnique.
   - SnpCleanShared.
   - SnpCleanInvalid.
2. The RN returns the data and associated response using the SnpRespData or SnpRespDataPtl opcode on the DAT channel.

Figure 2-18 shows the transaction structure.

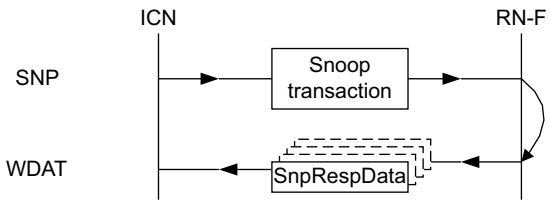


Figure 2-18 Snoop transaction structure with data to Home

The snoop with data to Home rules are:

- SnpRespData or SnpRespDataPtl, as required, must only be sent by the RN-F after the associated snoop request is received.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Snoop with Data forwarded to Requester without or with Data to Home

The progress of a Snoop transaction with Data forwarded to the Requester is as follows:

1. The interconnect provides a Snoop request on the SNP channel. This can be one of the following Snoop transactions:
   - SnpOnceFwd.
   - SnpCleanFwd.
   - SnpNotSharedDirtyFwd.
   - SnpSharedFwd.
   - SnpUniqueFwd.
2. The snooped RN forwards the Data to the Requester using the CompData opcode on the WDAT channel and either:
   - Sends a response to Home using the SnpRespFwded opcode on the SRSP channel.
   - Sends Data to Home using the SnpRespDataFwded opcode on the WDAT channel.

Figure 2-19 shows the transaction structure with response without Data to Home.

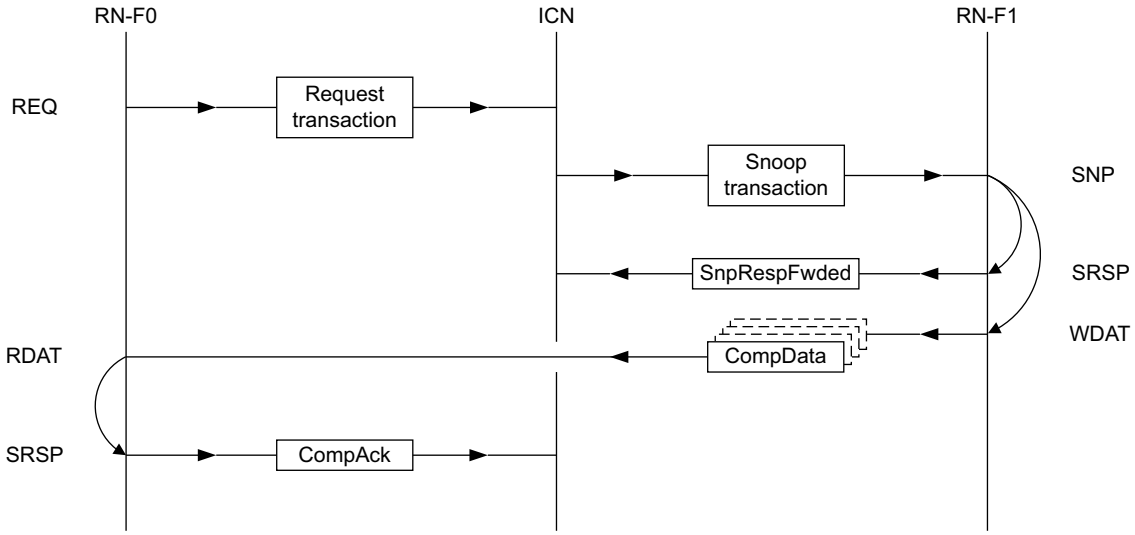


Figure 2-19 Snoop with Data forwarded to Requester with response to Home

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Figure 2-20 shows the transaction structure with response with Data to Home.

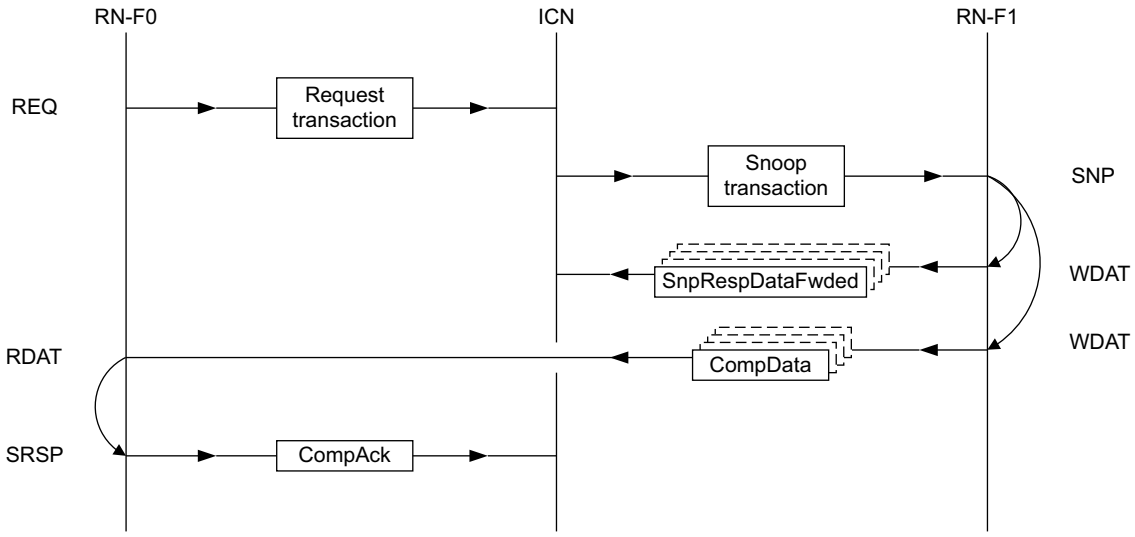


Figure 2-20 Snoop with Data forwarded to Requester with Data to Home

The Snoop request/response rules for Forward snoops are:

- SnpRespFwded or SnpRespDataFwded must only be sent by the Snoopee after the associated Snoop request is received.
- CompData must only be sent by the Snoopee after the associated Snoop request is received.
- SnpRespFwded or SnpRespDataFwded and CompData responses by the Snoopee can be sent in any order.
- CompAck must be sent only after receiving the first Data response packet.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Stash snoops

Figure 2-21 shows an example of a stash type snoop with Data Pull, Data response from Snoopee, and Data from Home. The RN-F provides data in response to the snoop. The RN-F is then returned CompData in response to the Read transaction initiated by the SnpRespData_Read response.

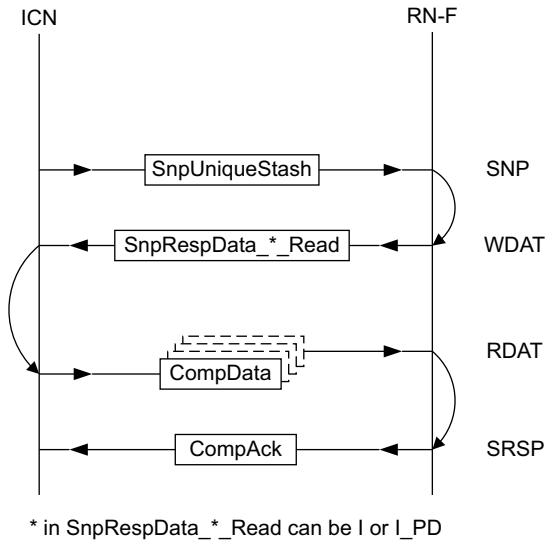


Figure 2-21 Stash type snoop with Data Pull, Data response from Snoopee, and Data from Home

Figure 2-22 shows an example of a stash type snoop with Data Pull, no Data response from Snoopee, and DMT. Data is provided by a DMT read from memory.

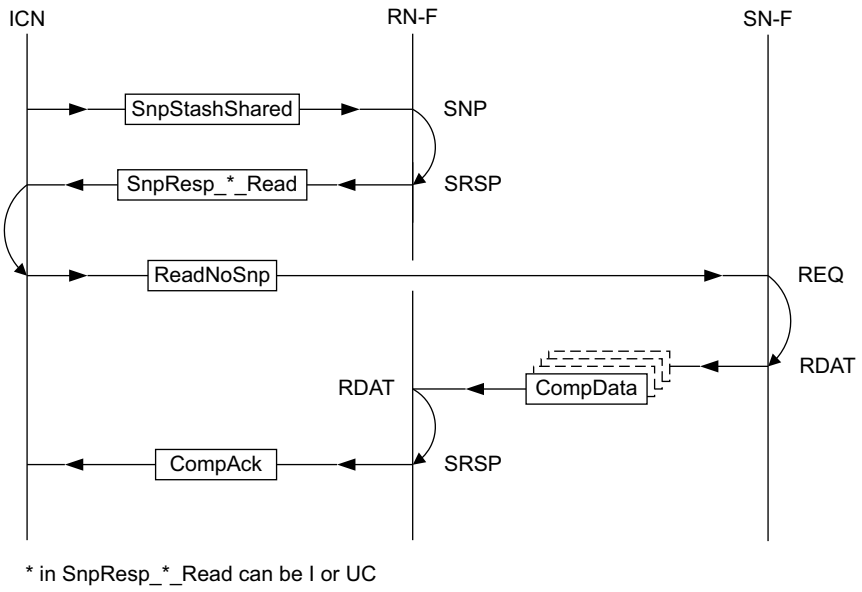


Figure 2-22 Stash type snoop with Data Pull, no Data response from Snoopee, and DMT

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Snoop DVMOp

The progress of a SnpDVMOp transaction is as follows:

1. The interconnect provides two snoop requests with the SnpDVMOp opcode on the SNP channel.
2. The RN returns a single SnpResp snoop response on the SRSP channel.

Figure 2-23 shows the transaction structure.

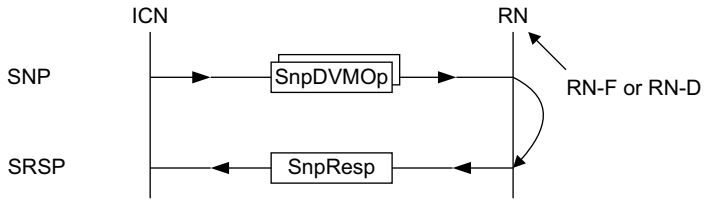


Figure 2-23 SnpDVMOp transaction structure

The SnpDVMOp rules are:

- The SnpResp response must only be sent by the RN after both snoop requests are received.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.4 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 different identifier fields are:

***Target Identifier* (TgtID), *Source Identifier* (SrcID)**

These identifiers route packets across the interconnect. See *Details of transaction identifier fields* on page 2-74 and Chapter 3 *Network Layer*.

***Transaction Identifier* (TxnID), *Data Buffer Identifier* (DBID), *Return Transaction Identifier* (ReturnTxnID), *Forward Transaction Identifier* (FwdTxnID)**

These fields relate all the packets associated with a single transaction. See *Details of transaction identifier fields* on page 2-74.

***Data Identifier* (DataID), *Critical Chunk Identifier* (CCID)**

These fields identify the individual data packets within a transaction. See *Data packetization* on page 2-117.

***Logical Processor Identifier* (LPID), *Stash Logical Processor Identifier* (StashLPID)**

These fields identify individual processing agents within a single Requester. See *Logical Processor Identifier* on page 2-95.

***Stash Node Identifier* (StashNID)**

This field identifies the node that is the Stash target. See *Supporting REQ packet fields* on page 7-250.

***Return Node Identifier* (ReturnNID), *Forward Node Identifier* (FwdNID)**

These fields identify the node that the response with Data is to be sent to. See *Details of transaction identifier fields* on page 2-74.

***Home Node Identifier* (HomeNID)**

This field is used to identify the node that the CompAck response is to be sent to. See *Details of transaction identifier fields* on page 2-74.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.5 Details of transaction identifier fields

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

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.

An 8-bit field is defined for the TxnID to accommodate up to 256 outstanding transactions. A Requester is permitted to reuse a TxnID value after it has received either:

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

Transaction identifier field flows on page 2-77 gives more detailed rules for the different transaction types. The TxnID field is not applicable in a PrefetchTgt request and can take any value.

Note

In CHI Issue B, the TxID is not applicable in the PrefetchTgt request and must be set to zero.

A value used in the TxnID field of a Request from Home to Slave 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 *Request Retry* on page 2-126.

A transaction request from Home to Slave includes a ReturnNID that is used to determine the TgtID for the Data response from the Slave. 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, and non-store Atomic requests from Home to Slave. The field is inapplicable in all other requests from Home to Slave and must be set to zero.

ReturnNID is inapplicable from Requester to Home and Requester to Slave and must be set to zero in all requests.

A transaction request from Home to Slave also includes a ReturnTxnID field to convey the value of TxnID in the data response from the Slave. 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, and non-store Atomic requests from Home to Slave. The field is inapplicable in all other requests from Home to Slave and must be set to zero.

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

CompData from Home, and from the Slave node, includes the HomeNID field that is used by the Requester to identify the target of the CompAck that it might need to send in response to CompData. HomeNID is applicable in CompData and DataSepResp and is inapplicable, and must be set to zero, 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, this specification requires that these values are included to assist in debugging and protocol checking.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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

The FwdNID field is only applicable in:

- SnpSharedFwd.
- SnpCleanFwd.
- SnpOnceFwd.
- SnpNotSharedDirtyFwd.
- SnpUniqueFwd.

It is inapplicable and must be set to zero in all other snoops.

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 RN-F. Its value must be the TxnID of the original Request.

The FwdTxnID field is only applicable in:

- SnpSharedFwd.
- SnpCleanFwd.
- SnpOnceFwd.
- SnpNotSharedDirtyFwd.
- SnpUniqueFwd.

It is inapplicable and must be set to zero in all other snoops.

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 Write, Atomic, and DVMOp transactions.
- CompData response of Stash transactions for Data Pull purposes.
- CompAck response of Read, Dataless, and WriteUnique 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 CompDBIDResp for all Write transactions.
- DBIDResp or CompDBIDResp for Atomic transactions.
- DBIDResp 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 message for a Write transaction must include the same DBID field value in the Comp and DBIDResp message.

A Comp response message sent separate from a DBIDResp message for a Atomic transaction is permitted, but is not required, to include the same DBID field value in Comp and DBIDResp 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. *Transaction identifier field flows* on page 2-77 gives more detailed rules for the different transaction types.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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

- The DBID values in other Snoop responses to Stash type 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 for:

- Read transactions without CompAck.
- Dataless transactions without CompAck.
- Snoop response to a Stash type snoop that does not include Data Pull.
- Snoop response to a Non-stash type 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 256 DBID responses to be active at the same time, then it must use SrcID in combination with DBID 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 *Transaction ordering* on page 2-99.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.6 Transaction identifier field flows

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

In the associated figures:

- The fields included in each packet are:
  - For a Request packet: TgtID, SrcID, TxnID, StashNID, StashLPID, ReturnNID and ReturnTxnID.
  - For a Response packet: TgtID, SrcID, TxnID and DBID.
  - For a Data packet: TgtID, SrcID, TxnID, HomeNID and DBID.
  - For a Snoop packet: SrcNID, 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, it indicates 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 re-mapped 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 3 *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 IDs that are not relevant for the example have been omitted for clarity.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.6.1 Read transactions

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

- *Identifier field flow with DMT in Read transactions*.
- *Identifier field flow with DMT and separate Comp and Data in Read transactions* on page 2-80.
- *Identifier field flow with DCT in Read transactions* on page 2-82.
- *Identifier field flow without Direct Data Transfer in Read transactions* on page 2-84.

Identifier field flow with DMT in Read transactions

Figure 2-24 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 ICN is assigned by ICN, whereas the ReturnNID, which is used as TgtID in the Data response, is set to the value of SrcID of the received Read request.

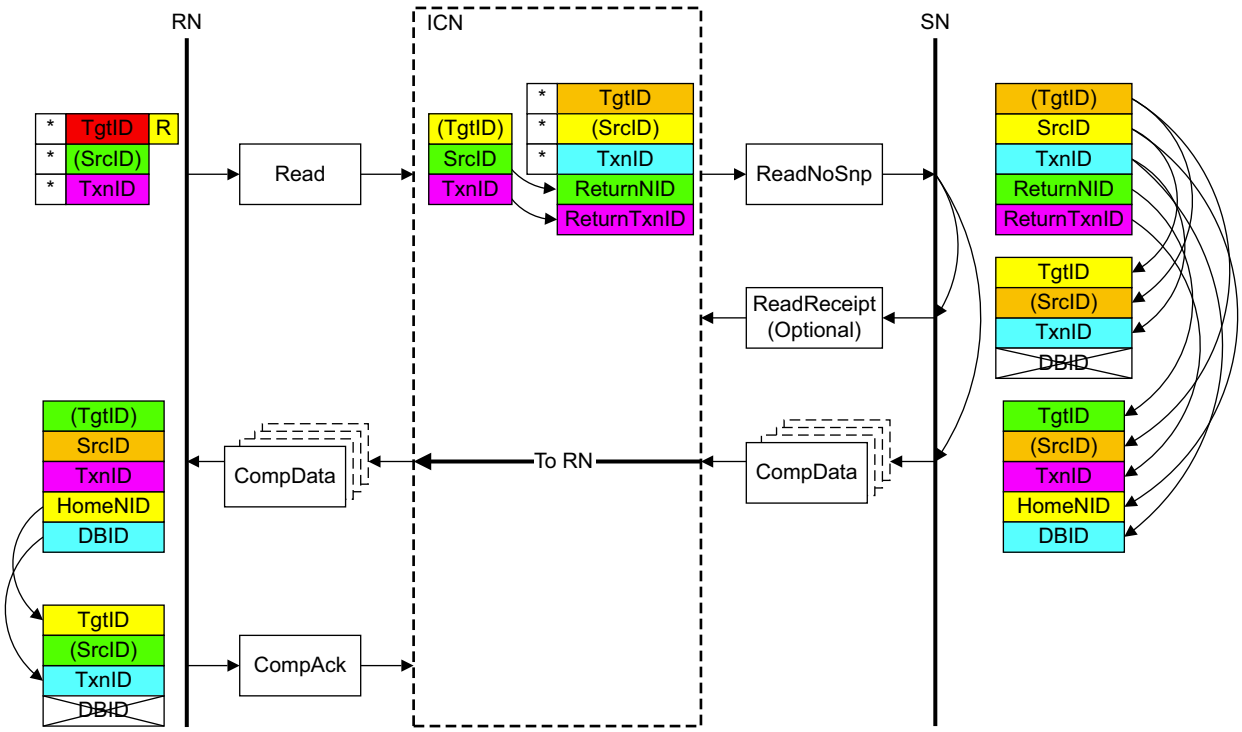


Figure 2-24 ID value transfer in a DMT transaction

The required steps in the flow that Figure 2-24 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2. The recipient Home Node in the ICN generates a Request to the Slave Node.

   The identifier fields of the request are generated as follows:
   - The TgtID is set to the value required for the Slave.
   - 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 Slave requires a ReadReceipt, the Slave 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 Slave. 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 Slave 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 Slave. 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 CompAck acknowledgment.

   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.

Details of transaction identifier fields on page 2-74 details when the TxnID value and DBID value can be reused.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Identifier field flow with DMT and separate Comp and Data in Read transactions

Figure 2-25 shows how the identifier field values are derived in DMT transaction messages that use separate Comp and Data.

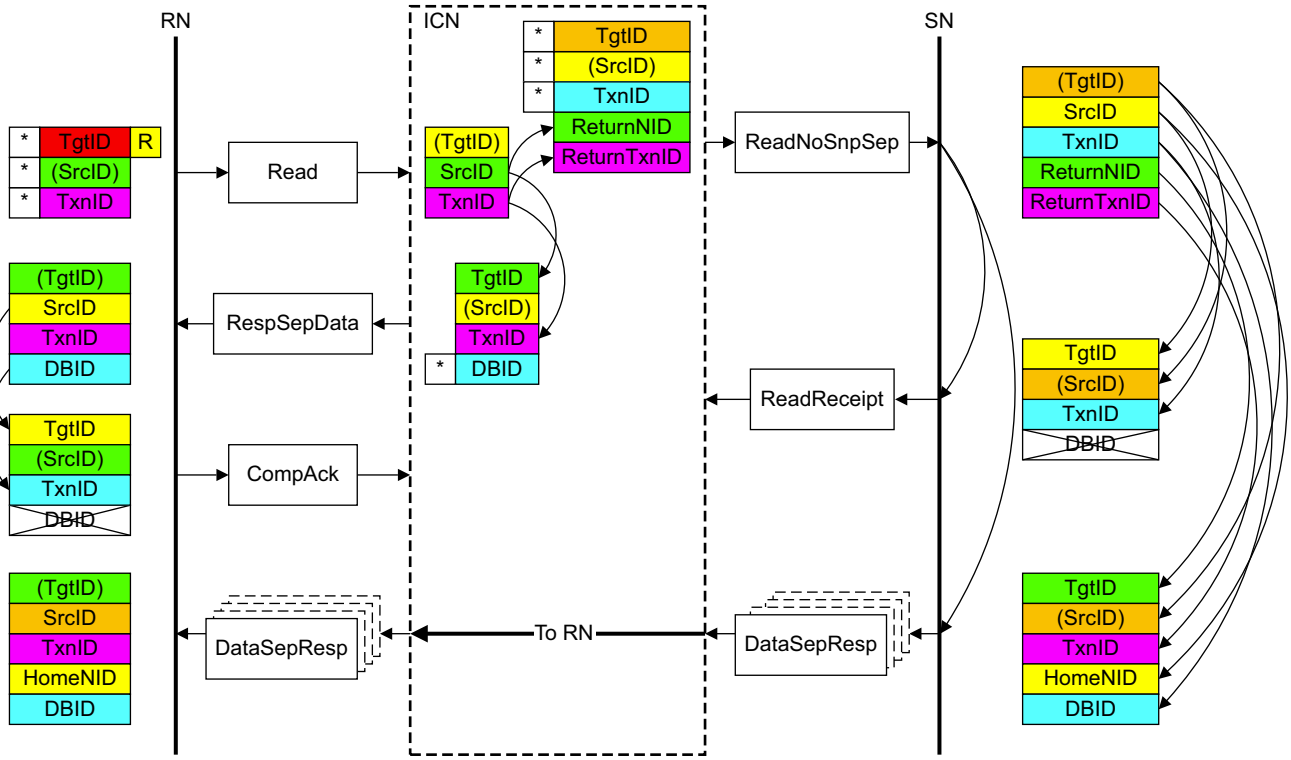


Figure 2-25 ID value transfer in a DMT transaction with separate Comp and Data

The required steps in the flow that Figure 2-25 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 ICN generates a request to the Slave Node.

   The identifier fields of the request are generated as follows:

   - The TgtID is set to the value required for the Slave.
   - 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

3. The recipient Home Node in the ICN provides the separate read response.

   The identifier fields of the read response are generated as follows:
   - 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 Slave.

4. The Requester receives the read response and sends a CompAck acknowledgment.

   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 TxID is set to the unique DBID value generated by the Home.
   - The DBID value is not valid.

5. The request to the Slave requires a ReadReceipt, the Slave 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 Slave. This also matches the TgtID received.
   - The TxnID is set to the same value as the TxnID of the request.

6. The Slave 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 Slave. 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Identifier field flow with DCT in Read transactions

Figure 2-26 shows how the identifier field values are derived in DCT transaction messages. In this example, the data is forwarded to RN and a Snoop response is sent to HN-F with or without data.

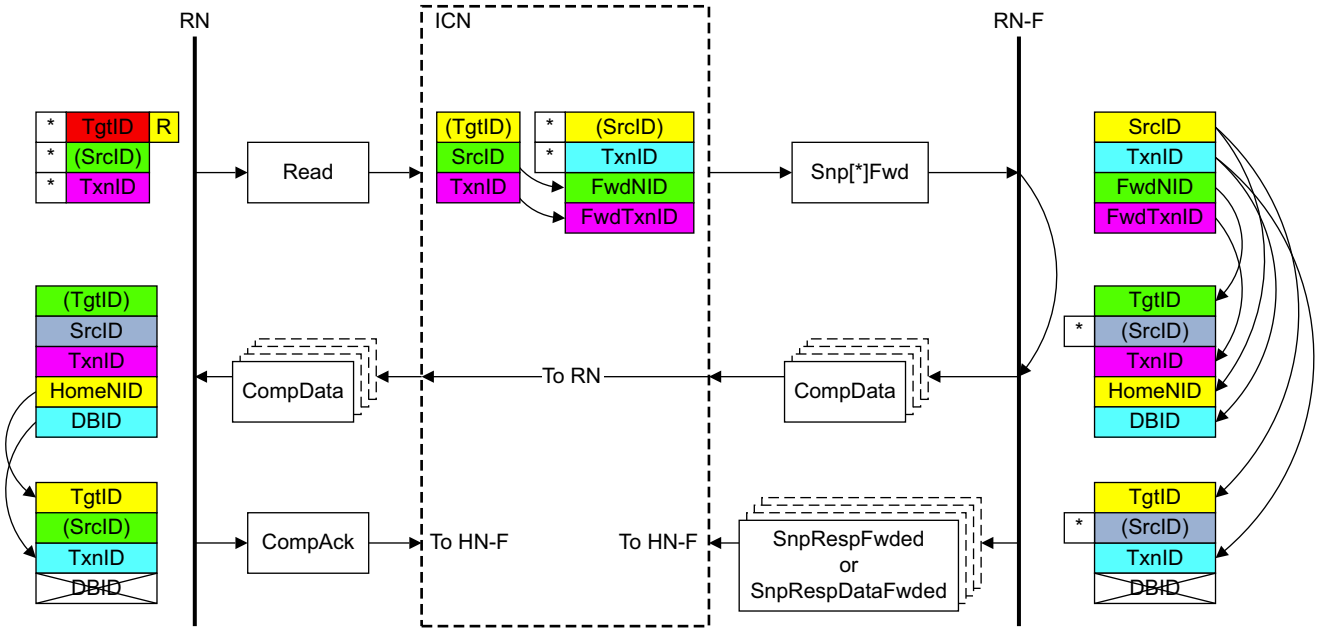


Figure 2-26 ID value transfer in a DCT transaction

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 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 ICN generates a fowarding 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 CompAck acknowledgment.

   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 ICN to Requester can also be included.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Identifier field flow without Direct Data Transfer in Read transactions

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 an RN 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 2-27 shows the identifier field flow.

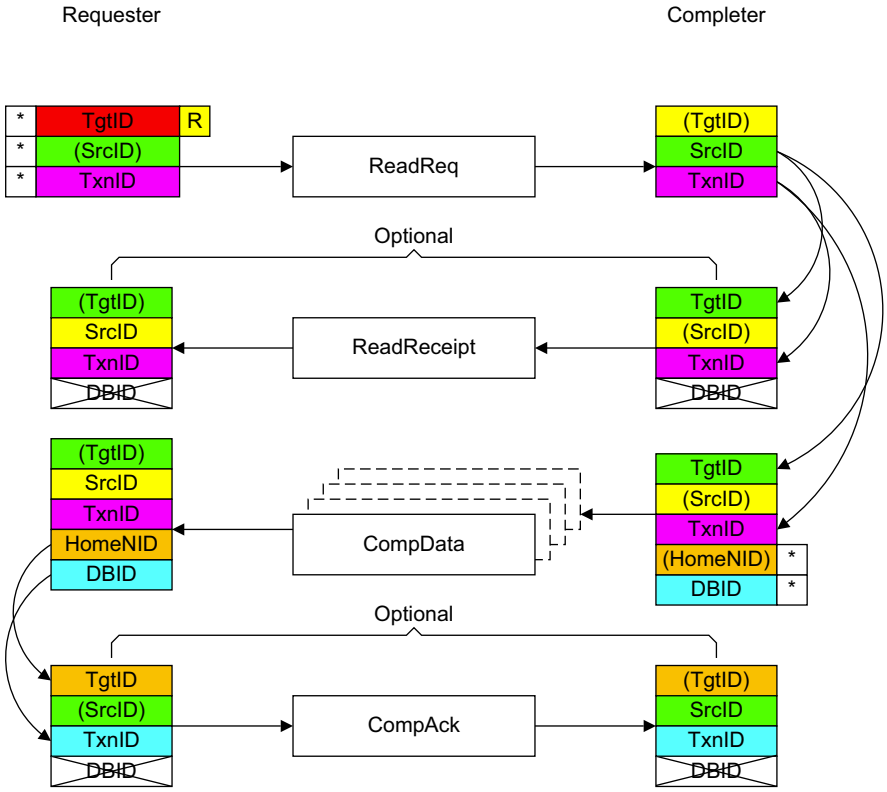


Figure 2-27 Identifier field flow for a Read request with ReadReceipt and CompAck

The required steps in the flow that Figure 2-27 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 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 Completer generates a unique DBID value if ExpCompAck in the request is asserted.
4. The Requester receives the read data and sends a CompAck acknowledgment.

   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.

2.6.2 Dataless transactions

For Dataless transactions the use of identifier fields is similar to *Identifier field flow without Direct Data Transfer in Read transactions* on page 2-84. 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.6.3 Write transactions

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

- *CopyBack and single response WriteNoSnp transaction*.
- *WriteNoSnp transaction with multiple responses* on page 2-87.
- *WriteUnique transaction* on page 2-89.
- *StashOnce transaction* on page 2-91.

CopyBack and single response WriteNoSnp transaction

This section describes the use of the identifier fields for a write transaction with a single combined CompDBIDResp response. Further details on the meaning of the response fields, and when a combined response is used, can be found in *Response types* on page 4-163.

Figure 2-28 shows the transaction identifier field flow.

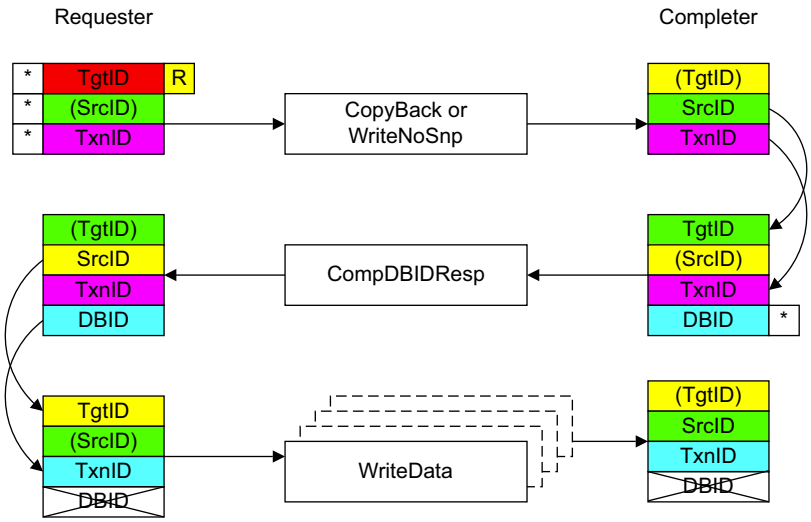


Figure 2-28 Identifier field flow for CopyBack and single response WriteNoSnp

The required steps in the flow that Figure 2-28 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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, to determine which transaction the write data is associated with.

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

WriteNoSnp transaction with multiple responses

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

Figure 2-29 shows the transaction identifier field flow.

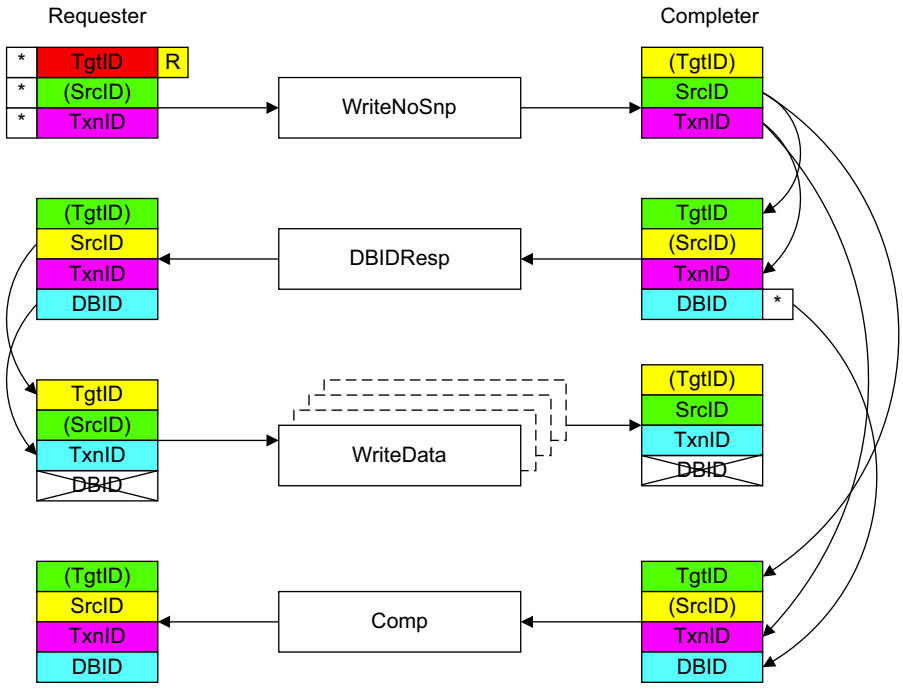


Figure 2-29 Identifier field flow for WriteNoSnp with multiple responses

The use 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The required steps in the flow that Figure 2-29 on page 2-87 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 it has completed the transaction.

   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.

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

After receiving all 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. The specification requirement is that the values used are identical.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

WriteUnique transaction

This section describes the use of the identifier fields for a WriteUnique transaction. The WriteUnique transaction can, under certain circumstances, additionally 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, then 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 2-30 shows the transaction identifier field flow with a combined CompDBIDResp response.

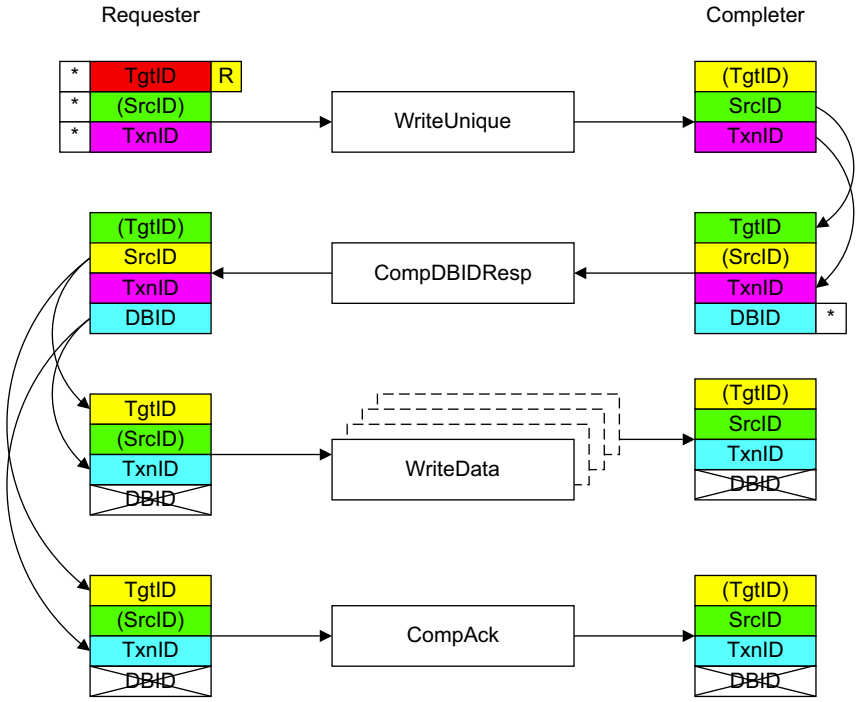


Figure 2-30 Identifier field flow for a WriteUnique

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Figure 2-31 shows the transaction identifier field flow with a combined WriteData and CompAck response.

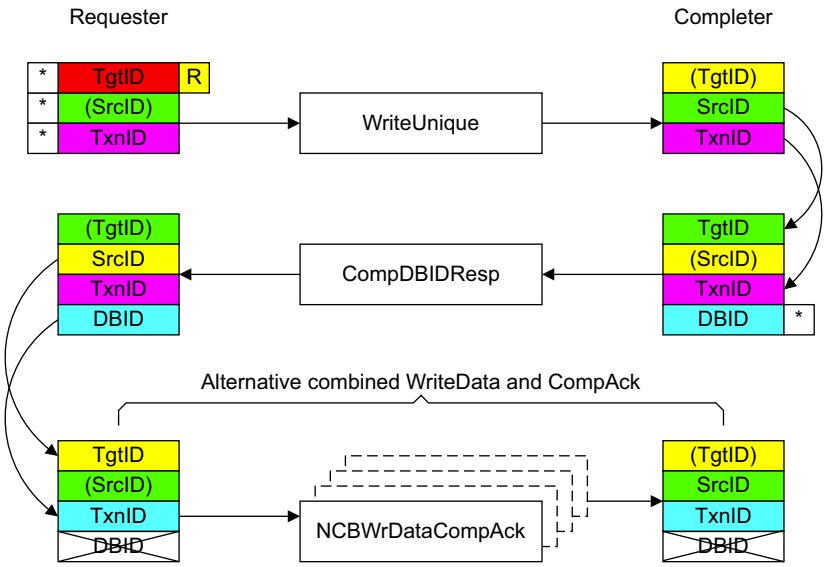


Figure 2-31 Identifier field flow for a WriteUnique with combined WriteData and CompAck

Note

Prior to CHI.C combining WriteData and CompAck is not permitted.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

StashOnce transaction

This section describes the use of the identifier fields for a StashOnce transaction with DataPull.

Figure 2-32 shows the transaction identifier field flow.

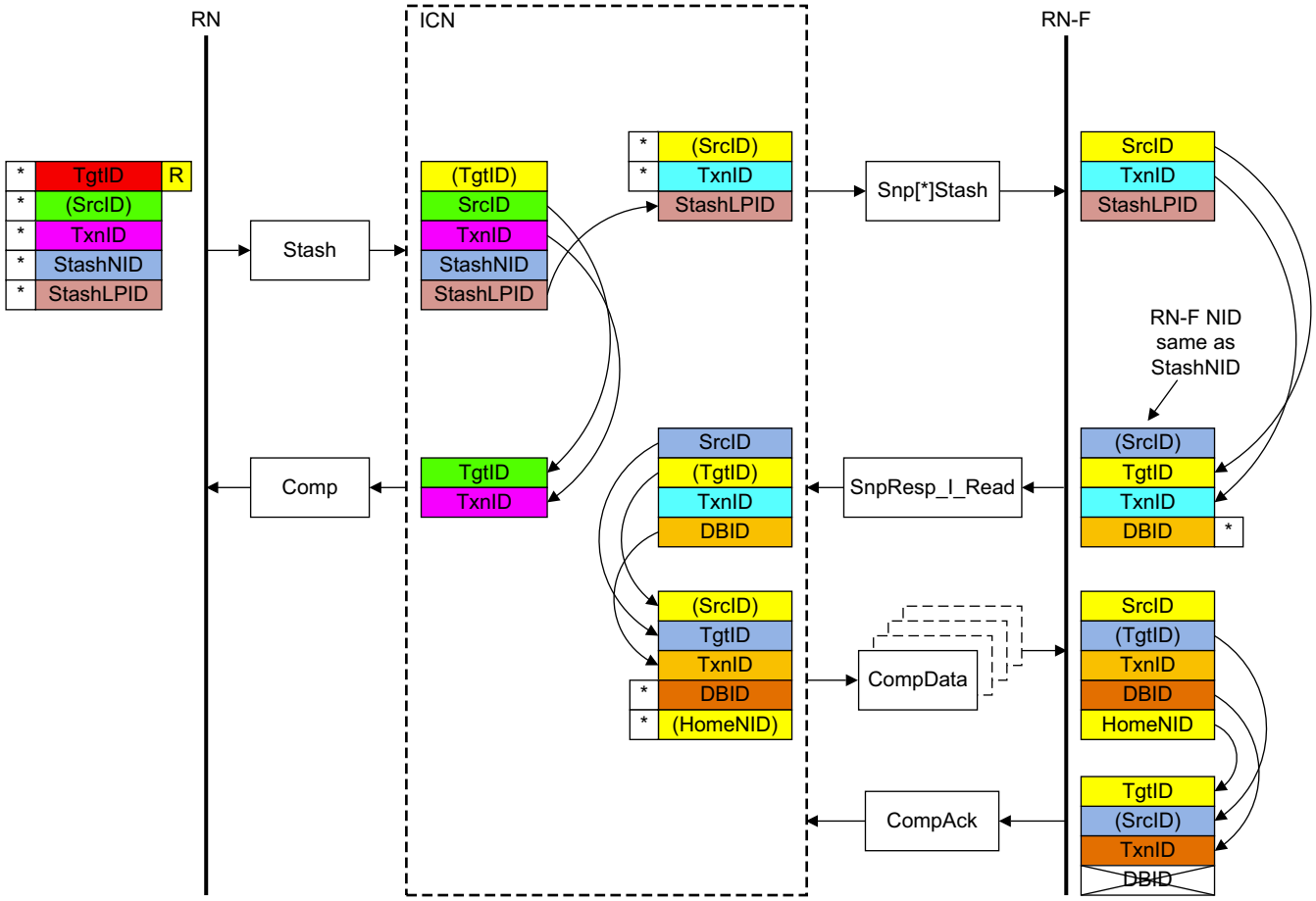


Figure 2-32 ID value transfer in a Stash transaction

The required steps in the flow that Figure 2-32 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 the RN-F to send the Stash to.
   - The Requester includes the StashLPID field to indicate the logical processor within the RN-F.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2. The Home Node in the ICN receives the Stash request packet and 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.

3. The snooped RN-F generates a Snoop response. In this example, it includes a Data Pull indication.

   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.

4. 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.

5. The RN-F receives the read data and sends a CompAck acknowledgment.

   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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.6.4 DVMOp transaction

The use of the TgtID, SrcID, TxnID and DBID identifier fields for a DVMOp transaction is identical to that for the *WriteNoSnp transaction with multiple responses* on page 2-87.

2.6.5 Transaction requests with Retry

For transactions that receive a RetryAck response, there are specific rules on how the identifier fields are used. See *Request Retry* on page 2-126, for more details on the Retry mechanism, and *Protocol Credit Return transaction* on page 2-94, for rules about the return of unused credits.

Figure 2-33 shows the transaction identifier field flow for a transaction request with retry.

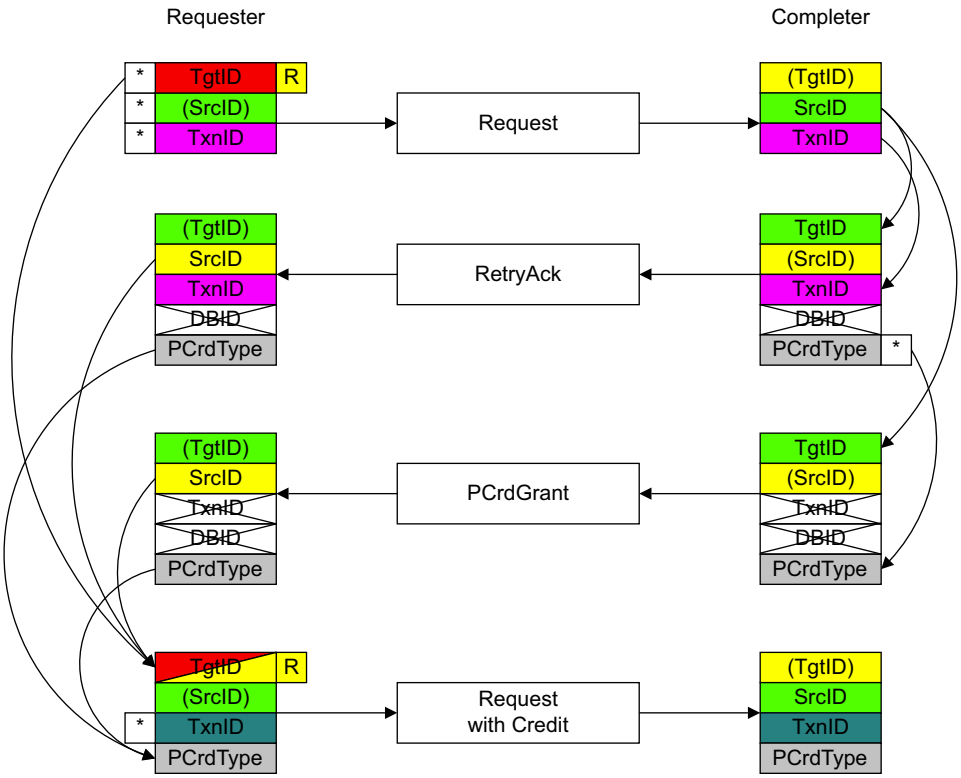


Figure 2-33 Identifier field flow for a transaction request with retry

The required steps in the flow that Figure 2-33 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2. The Completer receives the Request packet and determines that it is going to send a RetryAck response. 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 it sends a credit to the Requester, 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 zero.
   - The DBID field is not used and must be set to zero.
   - 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 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.

2.6.6 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 zero.

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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.7 Logical Processor Identifier

The specification defines a *Logical Processor Identifier* (LPID) field within a transaction request. This field is used when a single Requester contains more than one logically separate processing agent.

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

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

  See Chapter 6 *Exclusive Accesses* for further details.

Note

For other transactions, the LPID value can be used to indicate the original logical processor that caused a transaction to be issued. However, this information is not required in CHI and is optional.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.8 Ordering

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

- *Multi-copy atomicity*.
- *Completion Response and Ordering*.
- *Completion acknowledgement* on page 2-97.
- *Transaction ordering* on page 2-99.

For the meaning of the terms EWA, Device, and Cacheable see *Memory Attributes* on page 2-107.

2.8.1 Multi-copy atomicity

This specification requires a multi-copy atomic architecture. All compliant components must ensure that all write-type 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, that is, they are observed in the same order by all Requesters, although some Requesters might 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 NS attribute are the same.

2.8.2 Completion Response and Ordering

To guarantee the ordering of a transaction with respect to later transactions, either from the same agent or from another agent, the Comp, RespSepData, or CompData response is used as follows:

- For a Read transaction to a Non-cacheable or Device location, a RespSepData or CompData response guarantees that the transaction is observable to a later transaction from any agent to the same endpoint address range. The size of the endpoint address range is IMPLEMENTATION DEFINED.
- For a Read transaction to a Cacheable location, a CompData or DataSepResp response guarantees that the transaction is observable to a later transaction from any agent to the same location.
- For a Read transaction to a Cacheable location, a RespSepData response guarantees that no earlier transaction will send a snoop to this Requester, and all later transactions will send a snoop only if required after the Home receives the CompAck response for this transaction.
- For a Dataless transaction that is only permitted to a Cacheable memory location, a Comp response guarantees that the transaction is observable to a later transaction from any agent to the same memory location. In addition, for CleanSharedPersist transactions, an HN must send a Comp response only after the HN receives a Completion from downstream that indicates that the location has been made persistent.
- For a Write or an Atomic transaction to Non-cacheable or Device nRnE or Device nRE, a Comp or CompData response guarantees that the transaction is observable to a later transaction from any agent to the same endpoint address range.
- For a Write or Atomic transaction to a Cacheable or Device RE location, a Comp or CompData response guarantees that the transaction is observable to a later transaction, from any agent, to the same location.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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.

If the Comp response for a Write transaction, with EWA asserted, to a Non-cacheable or Device location does not guarantee that the transaction is observable to a later transaction from any agent to the same endpoint address range, then one of the following techniques can be used to ensure ordering to the same endpoint address range:

- If the Write transaction has an Endpoint Order requirement, then a later transaction from the same agent that also has an Endpoint Order requirement and is to the same endpoint address range will be ordered. See *Transaction ordering* on page 2-99.
- The CompData response of a later Read transaction to the same location ensures the ordering of the Write transaction with respect to a later transaction from any agent to any location within the same endpoint address range.

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

2.8.3 Completion acknowledgement

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 Acknowledgment 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 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 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*, then it is guaranteed not to receive a Snoop request to the same address between the point that it receives Comp and the point that it sends CompAck.

The use of CompAck for a transaction is determined by the Requester setting the ExpCompAck field in the original request. The rules for an RN 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*.
- Although not required, an RN-F is permitted 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.
- An RN that wants to make use of DMT must include a CompAck response in ordered ReadNoSnp and ReadOnce* transactions.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- For Write transactions, CompAck can only be used for WriteUnique transactions. See *Streaming Ordered WriteUnique transactions* on page 2-103.

For transactions between an RN and an HN, where the HN is the Completer, the HN must support the use of CompAck for all transactions that are required or permitted to use CompAck.

An SN 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 2-8 shows the Request types that require a CompAck response, and the corresponding Requester types that are required to provide that response.

Table 2-8 Requester CompAck requirement

Request type	CompAck Required	
	RN-F	RN-D, RN-I
ReadNoSnp	Optional	Optional
ReadOnce*	Optional	Optional
ReadClean	Yes	-
ReadNotSharedDirty	Yes	-
ReadShared	Yes	-
ReadUnique	Yes	-
CleanUnique	Yes	-
MakeUnique	Yes	-
CleanShared	No	No
CleanSharedPersist	No	No
CleanInvalid	No	No
MakeInvalid	No	No
WriteBack	No	-
WriteClean	No	-
WriteUnique	Optional	Optional
Evict	No	-
WriteEvictFull	No	-
WriteNoSnp	No	No
Atomics	No	No
StashOnce	No	No

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.8.4 Ordering semantics of RespSepData and DataSepResp

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

A Home must ensure that all required Snoop transactions are completed before initiating a transaction, such as a ReadNoSnpSep, which could result in the DataSepResp response being sent to the Requester.

When a Requester receives a RespSepData response from Home, the request that it applies to has been ordered at Home and the Requester will not receive any snoops for transactions that are scheduled before it. The Home, before sending RespSepData response to the Requester, 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 CompAck acknowledgement, this Requester is indicating that it will accept responsibility to hazard snoops for any transaction that is scheduled after it. The following rules apply:

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

Note

This specification requires that CompAck must not be given when only DataSepResp is received.

2.8.5 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 an RN, HN pair and a HN-I, SN-I pair. Between an HN-F, SN-F pair the order field is used to obtain a request accepted acknowledgment.

Requester Order between an RN, HN pair and a 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. This guarantee also includes the guarantee of Request Order.

Ordered Write Observation

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 will send a positive acknowledgement only when it has accepted the request.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 2-9 shows the Order field encodings.

Table 2-9 Order field encodings

Order[1:0]	Description	Notes	Between pairs
0b00	No ordering required	-	All pairs
0b01	Request Accepted	Applicable in Read request from HN-F to SN-F only. Reserved in all other cases.	HN-F to SN-F
0b10	Request Order/Ordered Write Observation required	Reserved in Read requests from HN-F to SN-F.	RN to HN HN-I to SN-I
0b11	Endpoint Order required, which includes Request Order		

Ordering requirements

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

- ReadNoSnp.
- ReadNoSnpSep.
- ReadOnce*.
- WriteNoSnp.
- WriteUnique.
- 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 will maintain requests in the order they were received:
  - For requests that require Request Order it will maintain order between requests to the same address from the same source.
  - For requests that require Endpoint Order it will 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 or a non-Snoopable Atomic transaction requires Request Order or Endpoint Order:

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

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

- The Requester requires a DBIDResp to determine when it can send the next ordered request.
- The Completer sending a DBIDResp response means that it will maintain order between requests to the same address from the same source.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When a WriteUnique transaction requires Ordered Write Observation:

- CompAck is required. The RN must assert ExpCompAck.
- The RN requires a DBIDResp.
- The Completer is a PoS. A PoS sending DBIDResp 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 Ordered Write Observation.
  - The write is not made visible until CompAck is received.

Table 2-10 shows the ordering guarantee that is obtained for different combinations of transactions that require order.

The transactions that Table 2-10 shows as First Transaction and Second Transaction are from the same Requester.

When transactions from the same Requester specify a different ordering requirement, the ordering guarantee that is provided is the least restrictive of the two.

Table 2-10 Order between transactions

First Transaction	Second Transaction	Order Guarantee
No ordering	No ordering	No ordering
No ordering	Request Order	No ordering
No ordering	Endpoint Order	No ordering
Request Order	No ordering	No ordering
Request Order	Request Order	Request Order
Request Order	Endpoint Order	Request Order
Endpoint Order	No ordering	No ordering
Endpoint Order	Request Order	Request Order
Endpoint Order	Endpoint Order	Endpoint Order

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 will not send a RetryAck response.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Read Request Order example

Figure 2-34 shows the request ordering of three read requests.

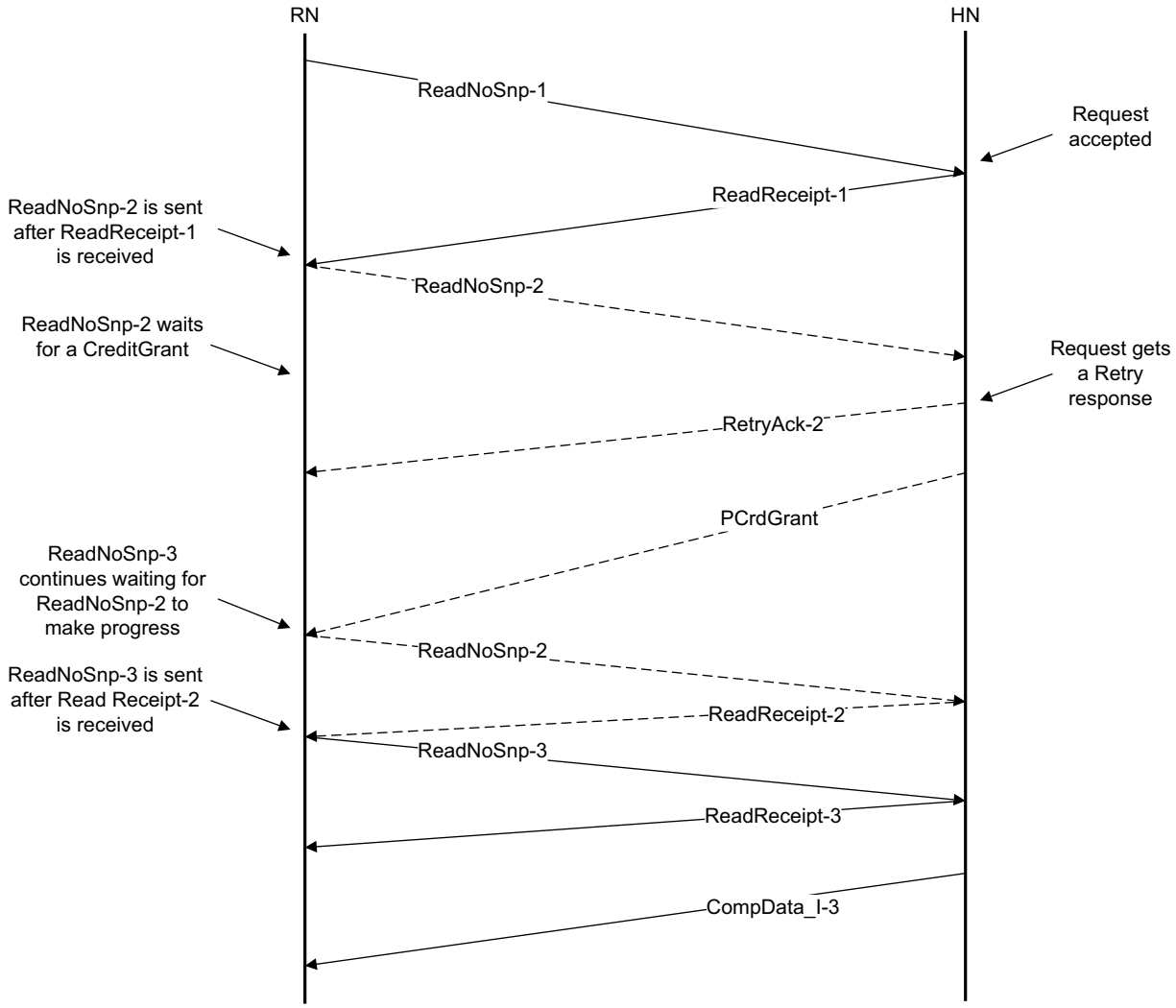


Figure 2-34 Series of ordered read requests

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Three ordered requests are sent from RN to HN as follows:

1. RN sends the ReadNoSnp-1 request to HN.
2. HN accepts the request and returns the ReadReceipt-1 response to RN.
3. After the ReadReceipt-1 response is received, RN sends the ReadNoSnp-2 request to HN.
4. HN cannot immediately accept the ReadNoSnp-2 request and returns the RetryAck-2 response to RN.
5. RN must now wait for a PCrdGrant to be sent from HN before resending the ReadNoSnp-2 request. RN does not send ReadNoSnp-3 at this point, as it wants to order ReadNoSnp-3 behind ReadNoSnp-2. This ordering requires that ReadNoSnp-2 must be accepted at HN before ReadNoSnp-3 is sent to HN.
6. After receipt of an appropriate PCrdGrant, RN resends the ReadNoSnp-2 request.
7. HN accepts the request and returns a ReadReceipt-2 response to RN.
8. After receipt of the ReadReceipt-2 response, RN sends the ReadNoSnp-3 request to HN.
9. HN accepts the request and returns the ReadReceipt-3 response to RN.
10. HN completes the Request transactions by sending a combined Completion and Data response to the RN for each request.

Note

Figure 2-34 on page 2-102 shows a single ordered stream of three reads from RN. However, an RN can have multiple streams of reads, in which case requests must be ordered within a stream, but ordering dependency does not exist between streams. One example of this is when the streams are from different threads within the RN, in which case, the RN waits for the ReadReceipt of the previous request from the same thread only before sending out the next ordered request from that stream.

CopyBack Request order

An RN-F must wait for the CompDBIDResp 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 CompDBID response is received for an outstanding CopyBack to the same cache line.

Streaming Ordered WriteUnique transactions

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

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

- The Requester must set the Order field to require Ordered Write Observation and ExpCompAck on the WriteUnique request.
- The Ordered Write Observation requirement in a WriteUnique 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 for a WriteUnique transaction before sending the next WriteUnique request.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- The Requester, after receiving Comp for the corresponding WriteUnique, as well as Comp responses for all earlier related ordered WriteUniques, must send a CompAck response, or if write data is to be sent, then combine CompAck with the write data response into a NCBWrDataCompAck response.

  Note

  Waiting to send CompAck until all prior ordered WriteUniques have received their Comp responses ensures that they have completed their operations at their respective HN-Fs and any Requester observing the WriteUnique for which CompAck is sent will also observe all prior ordered WriteUniques.

- HN-F must wait for a CompAck response from RN before deallocating a WriteUnique transaction and making the write visible to other observers.

Optimized Streaming Ordered WriteUniques

The Streaming Ordered WriteUniques mechanism can be further optimized. If a previously sent WriteUnique is to a different target, then the Requester does not need to wait for the DBIDResp for the request before sending the next ordered WriteUnique. However, if the interconnect can remap the TgtID, then the Requester must presume that all WriteUniques are targeting the same HN-F and must not use the optimized version of the Streaming Ordered WriteUniques flow.

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

Note

- A technique for avoiding resource related deadlock or livelock issues is to limit Streaming Ordered WriteUniques optimization to one Requester in the system. All other Requesters in the system can use the Streaming Ordered WriteUniques solution without the optimization.
- In a typical system, the optimized Streaming Ordered WriteUniques 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.
- Optimized Streaming Ordered WriteUnique can be used by more than one Requester by making use of WriteDataCancel messages to avoid Resource related deadlocks and livelocks.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Figure 2-35 shows a typical transaction flow in which an RN-I uses Streaming Ordered WriteUniques. This flow prevents a read acquiring the new value of Write-B before Write-A has completed.

Note

For clarity, in Figure 2-35 the Write-B DBIDResp and the NCBWrData flow is omitted.

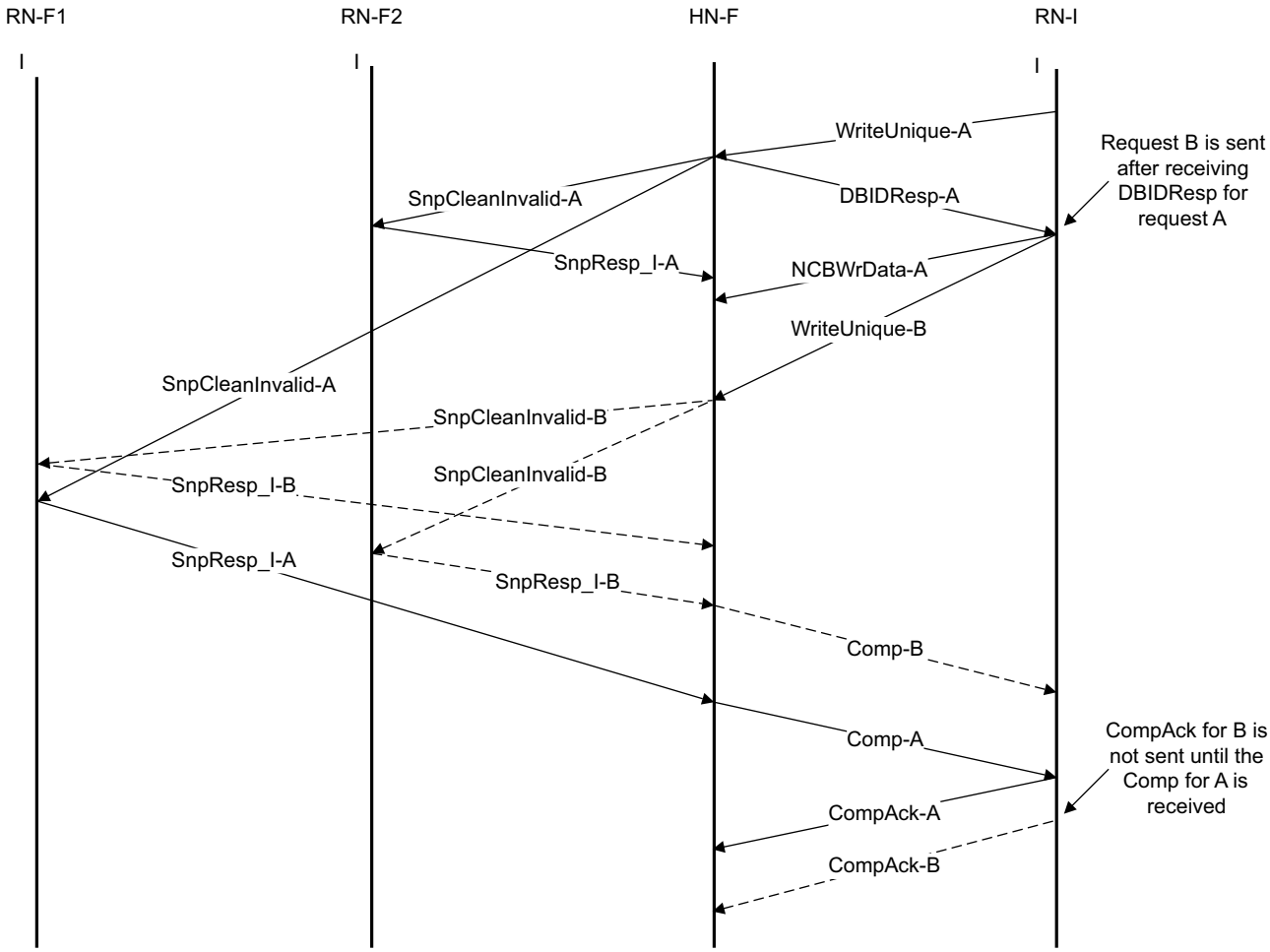


Figure 2-35 Streaming Ordered WriteUniques transaction flow

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.9 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.

2.9.1 Address

This specification supports:

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

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

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

MPA is the maximum PA supported.

Table 2-11 shows the relationship between the physical address field width and the supported virtual address.

Table 2-11 Addr field width and supported PA and VA size

REQ Addr field width in bits	Maximum address supported in bits	
	PA	VA
44	44	49
45	45	51
46 to 52	46 to 52	53

See *DVMOp and SnpDVMOp packet* on page 8-262 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 physical address 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.

The REQ and SNP channel messages address field in the REQ channel is a 44-bit to 52-bit field labeled Addr[(43-51):0] and in the SNP channel it is a 41-bit to 49-bit field labeled Addr[(43-51):3]. This field is used by the different message types as follows:

- For Read, PrefetchTgt, Dataless, Write, and Atomic transactions the Addr field includes the address of the memory location being accessed.
- For a Snoop request, except SnpDVMOp, the field includes the address of the location being snooped:
  - Addr[(43-51):6] is the cache line address and 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 *Critical Chunk Identifier* on page 2-120. This specification recommends that the snooped cache returns the data in wrap order with the critical chunk returned first.

    Note

    Addr[3] is supplied, but is not used.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- For a DVMOp and SnpDVMOp request the Addr field is used to carry information related to a DVM operation. See Chapter 8 *DVM Operations*.
- The Addr field value is not used for the PcrdReturn transaction and must be set to zero.

2.9.2 Non-secure bit

Secure and Non-secure transactions are defined to support Secure and Non-secure operating states.

This bit is defined so that when it is asserted the transaction is identified as a Non-secure transaction.

For Snoopable transactions this field can be considered as an additional address bit that defines two address spaces, a Secure address space, and a Non-secure address space. Any aliasing between the Secure and Non-secure address spaces must be handled correctly.

Note

Hardware coherency does not manage coherency between Non-Secure and Secure address spaces.

The NS assertion requirements are:

- Can be asserted in any Read, Dataless, Write and Atomic transaction.
- Can be asserted in PrefetchTgt transaction.
- Is not applicable in the DVMOp or PCrdReturn transaction, and must be set to zero.

2.9.3 Memory Attributes

The *Memory Attributes* (MemAttr) consist of *Early Write Acknowledgement* (EWA), Device, Cacheable, and Allocate.

EWA

EWA indicates whether the write completion response for a transaction:

- Is permitted to come from an intermediate point in the interconnect, such as a Home Node.
- Must come from the final endpoint that a transaction is destined for.

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 *Completion Response and Ordering* on page 2-96.

If EWA is deasserted, the write completion response for the transaction must come from the endpoint.

Note

It is permitted for an implementation not to use the EWA attribute, in this case completion must be given from the endpoint.

The EWA assertion requirements are:

- Can take any value in a ReadNoSnp and ReadNoSnpSep transaction.
- Can take any value in a WriteNoSnp transaction.
- Can take any value in CMO transactions.
- Can take any value in Atomic transactions.
- Must be asserted in any Read or Dataless transaction that is not a ReadNoSnp, ReadNoSnpSep, or CMO transaction.
- Must be asserted in any Write transaction that is not a WriteNoSnp transaction.
- Is inapplicable in the DVMOp or PCrdReturn transactions and must be set to zero.
- Is inapplicable in PrefetchTgt transaction and can take any value.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Device

Device attribute indicates if the memory type is either Device or Normal.

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 either WriteNoSnpFull or WriteNoSnpPtl.
- 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 bit value is inapplicable and can take any value.

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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 not be asserted for any Device memory transaction.
- Must be asserted for any Read transaction except for ReadNoSnp and ReadNoSnpSep.
- Must be asserted for any Dataless transaction except for CleanShared, CleanSharedPersist, CleanInvalid, MakeInvalid.
- Must be asserted for any Write transaction except WriteNoSnpFull and WriteNoSnpPtl.
- Can take any value for ReadNoSnp, ReadNoSnpSep, WriteNoSnpFull, and WriteNoSnpPtl to a Normal memory location.
- Can take any value for CleanShared, CleanSharedPersist, CleanInvalid and MakeInvalid.
- Can take any value for an Atomic transaction.
- Is inapplicable in DVMOp and PCrdReturn transactions and must be set to zero.
- Is inapplicable in the PrefetchTgt transaction and can take any value.

Note

In a transaction that can take any Cacheable value, the value is typically determined from the page table attributes.

Allocate

The Allocate attribute is a an allocation hint. It 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, it 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, it is permitted to allocate the transaction.

The Allocate attribute value requirements are:

- Can be asserted for transactions that have the Cacheable attribute asserted.
- 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.
- Must not be asserted for Normal Non-cacheable memory transactions.
- Is inapplicable in DVMOp, PCrdReturn and Evict transactions and must be set to zero.
- Is inapplicable in the PrefetchTgt transaction and can take any value.

Propagation of Attr

The MemAttr bits EWA, Device, Cacheable, and Allocate, must be preserved on a request from HN to SN that is sent in response to a request to HN. 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.

The SnpAttr attribute bit value does not need to be preserved but must be set to `0b0`.

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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.9.4 Transaction attribute combinations

Table 2-12 lists the legal combinations of MemAttr, SnpAttr, and Order field values and the equivalent ARM memory type. The Order field is described in *Ordering* on page 2-96.

Table 2-12 Legal combinations of MemAttr, SnpAttr, and Order field values

MemAttr[3:0]				SnpAttr	LikelyShared	Order[1:0]		ARM Memory Type
[1] Device	[3] Allocate	[2] Cacheable	[0] EWA			[1]	[0]	
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
	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 and Atomic transactions only.

b. Order = `0b01` is not used for transactions' ordering. See *Order field encodings* on page 2-100.

c. Non-cacheable Non-bufferable is an AXI memory type, not an ARM memory type.

d. LikelyShared = 1 is only permitted for ReadShared, ReadNotSharedDirty, ReadClean, WriteBackFull, WriteCleanFull, WriteEvictFull, WriteUnique and StashOnce transactions.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Memory type

This section specifies the required behavior for each of the memory types that Table 2-12 on page 2-110 shows.

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.

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.

Device RE

The required behavior for the Device RE memory type is same as for the Device nRE memory type except that:

- 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.

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.

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:
  - It 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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.

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, that is, it is a recommendation to the memory system that, for performance reasons, the transaction is not allocated. However, the allocation of the transaction is not prohibited.

Write-back Allocate

The required behavior for the WriteBack Allocate memory type is the same as for WriteBack 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.

2.9.5 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 are:

- Can be asserted in:
  - ReadClean.
  - ReadNotSharedDirty.
  - ReadShared.
  - StashOnceUnique.
  - StashOnceShared.
  - WriteUniquePtl.
  - WriteUniqueFull.
  - WriteUniquePtlStash.
  - WriteUniqueFullStash.
  - WriteBackFull.
  - WriteCleanFull.
  - WriteEvictFull.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Must not be asserted in any other Read or Write transaction.
- Must not be asserted in any Dataless or Atomic transaction.
- Is not applicable in DVMOp or PCrdReturn transaction, and must be set to zero.
- Is not applicable in PrefetchTgt transaction and can take any value.

2.9.6 Snoop Attribute

The *Snoop Attribute* (SnpAttr) indicates if a transaction requires snooping.

Table 2-13 shows the SnpAttr field encodings.

Table 2-13 SnpAttr field encodings

SnpAttr	Snoop attribute
0	Non-snoopable
1	Snoopable

Table 2-14 shows the snoop attributes for the different transaction types.

Table 2-14 Snoop attributes for the different transaction types

Transaction	Non-snoopable	Snoopable
ReadNoSnp, ReadNoSnpSep	Y	-
ReadOnce*, ReadClean, ReadShared, ReadNotSharedDirty, ReadUnique	-	Y
CleanUnique, MakeUnique, StashOnce	-	Y
CleanShared, CleanSharedPersist, CleanInvalid, MakeInvalid	Y	Y
Evict	-	Y
WriteNoSnp	Y	-
WriteBack, WriteClean, WriteEvictFull	-	Y
WriteUnique	-	Y
Atomic transactions	Y	Y
DVMOp	n/a[a]	n/a[a]
PrefetchTgt	n/a[b]	n/a[b]

a. Not applicable, must be set to zero.
b. Not applicable, can take any value.

The SnpAttr field value in a CMO, and in ReadNoSnp and ReadNoSnpSep from Home to Slave must be set to zero, irrespective of the field value in the Request from the original Requester to Home.

Note

For transactions that can take more than one value of SnpAttr, the value is typically determined from page table attributes.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.9.7 Do not transition to SD

Do not transition to SD is a modifier on Non-invalidating snoops.

It specifies that the Snoopee must not transition to SD state as a result of the Snoop request.

The modifier is applicable to the following Snoop requests:

- SnpCleanFwd.
- SnpClean.
- SnpNotSharedDirtyFwd.
- SnpNotSharedDirty.
- SnpSharedFwd.
- SnpShared.

The field value must be set to 1 in the following Snoop requests:

- SnpUniqueFwd.
- SnpUnique.
- SnpCleanShared.
- SnpCleanInvalid.
- SnpMakeInvalid.

The field can take any value in the following Snoop requests and the Snoopee is permitted to ignore the value:

- SnpOnceFwd.
- SnpOnce.

See *DoNotGoToSD* on page 12-327 for the field value encoding.

2.9.8 Mismatched Memory attributes

It is permitted for two different agents to access the same location using mismatched MemAttr or SnpAttr memory attributes, at the same point in time.

Memory accesses from the different agents made with mismatched snoopability or cacheability attributes are considered as software protocol errors. A software protocol error can cause loss of coherency and result in the corruption of data values. It is required that the system does not deadlock on a software protocol error, and that transactions always make forward progress.

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 that it is performing a ReadNoSnp or WriteNoSnp transaction to. In this situation there is no defined relationship between the Snoop transaction and the transaction that the RN-F has issued.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.10 Data transfer

Read transactions, Write transactions, Atomic transactions, and Snoop responses with data, include a data payload. This section defines the data alignment rules, and the data bytes that are accessed for different combinations of address, transaction size, and memory type.

2.10.1 Data size

The Size field in a packet is used, in combination with other fields, to determine the number of bytes transferred. Table 2-15 shows the Size field value encodings. Snoop transactions do not include a Size field. All snoop data transfers are 64-byte.

Table 2-15 Size field value encodings

Size[2:0]	Bytes
0b000	1
0b001	2
0b010	4
0b011	8
0b100	16
0b101	32
0b110	64
0b111	Reserved

2.10.2 Bytes access in memory

The MemAttr[1] bit field determines if the memory type is Device or Normal. See *Memory Attributes* on page 2-107. 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

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, byte enables must only be asserted for the bytes that are accessed. See *Byte Enables* on page 2-116.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.10.3 Byte Enables

Byte Enables, also referred to as BE, are used alongside Write transactions, and Snoop responses with Data.

For Write transactions, an asserted byte enable indicates that the associated data byte is valid and must be updated in memory or cache. A deasserted byte enable indicates that the associated data byte is not valid and must not be updated in memory or cache.

In Write Data and Snoop response Data a byte enable value of zero must set the associated data byte value to zero.

If a snoop occurs between the sending of the Request and sending of the Data, and the Dirty copy of the Data from the CopyBack is passed to the corresponding snoop response, then a CopyBackWrData_I packet is sent as the Data response for the Copyback request indicating to the Home that the Copyback is canceled. A Requester must deassert all BE values in a CopyBackWrData_I packet. The Requester must also deassert all BE values in a WriteDataCancel packet that are a result of canceling of a WriteUniquePtl, WriteUniquePtlStash or WriteNoSnpPtl transaction.

The following Write transactions must have all byte enables asserted during the data transfers, except when the write data is a CopyBackWrData_I packet:

- WriteNoSnpFull.
- WriteBackFull.
- WriteCleanFull.
- WriteEvictFull.
- WriteUniqueFull.
- WriteUniqueFullStash.

The following Write transactions are permitted to have any combination of byte enables asserted during the data transfers. This includes asserting all and asserting none:

- WriteBackPtl.
- WriteUniquePtl.
- WriteUniquePtlStash.

For the WriteNoSnpPtl transaction the following rules apply:

- For a transaction to Normal memory, any combination of byte enables can be asserted during the data transfers. This includes asserting all and asserting none.
- For a transaction to Device memory, byte enables must only be asserted for bytes at or above the address specified in the transaction. Any combination of byte enables can be asserted that meets this requirement. This includes asserting all and asserting none.

For all Write transactions, byte enables that are not within the data window, specified by Addr and Size, must be deasserted.

For Atomic transactions, byte enables that are not within the data window, as specified below by Addr and Size, must be deasserted:

- If Addr is aligned to Size, then the Data window is `[Addr:(Addr+Size-1)]`.
- If Addr is not aligned to Size, then the Data window is `[(Addr-Size/2):(Addr+Size/2-1)]`.
- For Atomic transactions all byte enables within the data window must be asserted.

For snoop responses with data that use the SnpRespData opcode, all byte enables must be asserted.

For snoop responses with data that use the SnpRespDataPtl opcode, any combination of byte enables can be asserted alongside the data transfers. This includes asserting all and asserting none.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.10.4 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.

This specification supports the following data bus widths:

- 128-bit.
- 256-bit.
- 512-bit.

The Data Identifier and Critical Chunk Identifier 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 field value must be set to Addr[5:4] because the DataID field represents Addr[5:4] of the lowest addressed byte within the packet.

Table 2-16 shows the relationship between the DataID field and the bytes that are contained within the packet, for different data bus widths.

Table 2-16 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, it can be determined from the address at the start of the transaction that some data packets will not contain valid data and are redundant. However, this specification requires that these data packets are transferred.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.10.5 Size, Address and Data alignment in Atomic transactions

This section describes the data size and alignment requirements for Atomic transactions. It contains the following sub-sections:

- *Size*.
- *Address and Data alignment*.
- *Endianness* on page 2-119.

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 2-17 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 2-17 Atomic transaction outbound and inbound data sizes

Atomic transaction	Outbound	Inbound
AtomicStore	1, 2, 4 or 8 byte	-
AtomicLoad	1, 2, 4 or 8 byte	Same as outbound
AtomicSwap	1, 2, 4 or 8 byte	Same as outbound
AtomicCompare	2, 4, 8, 16 or 32 byte	Half size of outbound

Address and Data alignment

In the AtomicStore, AtomicLoad and AtomicSwap transactions:

- The byte address is aligned in the Data packet to the outbound data size.
- The position of data bytes in the Data 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 AtomicSwap and AtomicCompare transaction is provided as if it were for 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)

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Alignment example

Figure 2-36 shows examples of data placement with different addresses and different Data size.

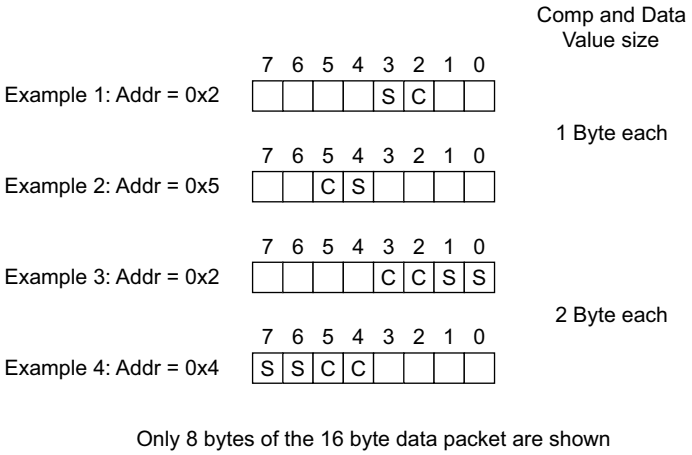


Figure 2-36 Data value packing for AtomicCompare transaction

In the first example that Figure 2-36 shows, the addressed byte location is `0x2` and the total size of data is two bytes. In this case, the Compare and Swap data must be placed in an address location aligned to a two 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 one byte.

In the third example that Figure 2-36 shows, the addressed location is `0x2` and the total size of data is four bytes. In this case, the Compare and Swap data must be placed in an address location aligned to a four 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 two bytes.

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 *Endian* on page 12-325.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.10.6 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 data bus width of 256 bits only the upper order CCID and DataID bits must match.

2.10.7 Critical chunk first wrap order

The Sender of Data is permitted, but is 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 that it is capable of sending Data packets in critical chunk first wrap order.
  - **False** The Sender signals that it is not capable of sending Data packets in critical chunk first wrap order.
- CCF_Wrap_Order at the ICN:
  - **True** ICN guarantees that it will maintain the Data packets of a transaction in the order they are received.
  - **False** ICN signals that it does not guarantee it can maintain the Data packets of a transaction in the order they are received.
- CCF_Wrap_Order at the Receiver that is not an ICN:
  - **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 then 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 knows that it is connected to an out-of-order interconnect, then it might be able to simplify its Data packet path by not returning the Data packets in critical chunk first wrap order.

If the interconnect has the CCF_Wrap_Order property set to True, then a component interfacing to that interconnect, if capable, can send Data packets in critical chunk first wrap order, and the receiver can make use of possible latency optimization due to receiving the critical chunk first.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.10.8 Data Beat ordering

This specification permits reordering of data packets within a transaction 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 *Critical chunk first wrap order* on page 2-120.

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 might overlap and not be required if the required bytes are included in a previous step.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

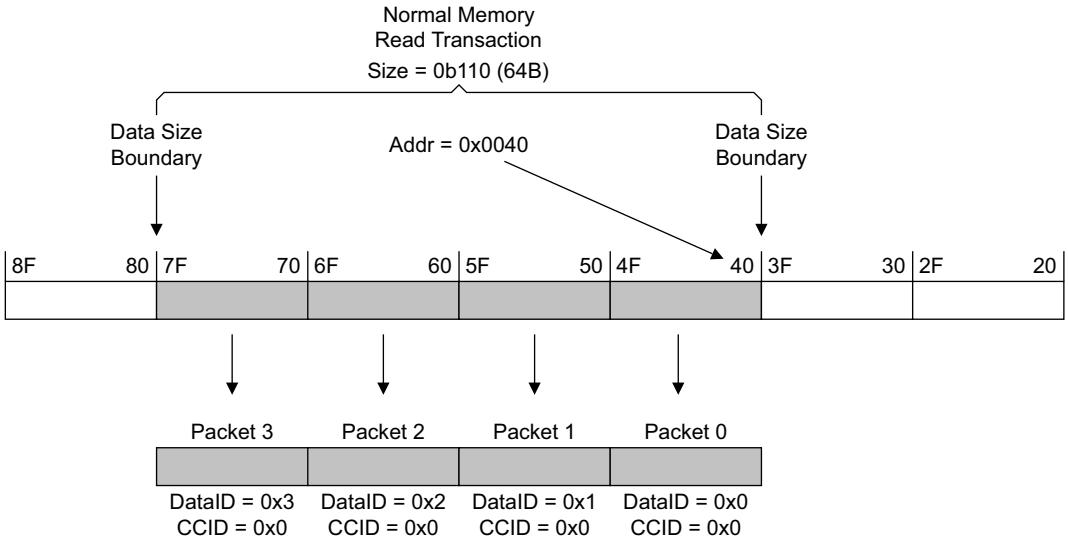
2.10.9 Data transfer examples

This section gives a number of examples of the data transfer requirements defined in this specification.

In most of the examples, the size of the transaction is 64-byte and the data bus width is 128-bit. This requires 4 data packets for each transaction.

In the following examples, the accompanying text highlights some interesting aspects. It is not intended to describe all aspects of the example.

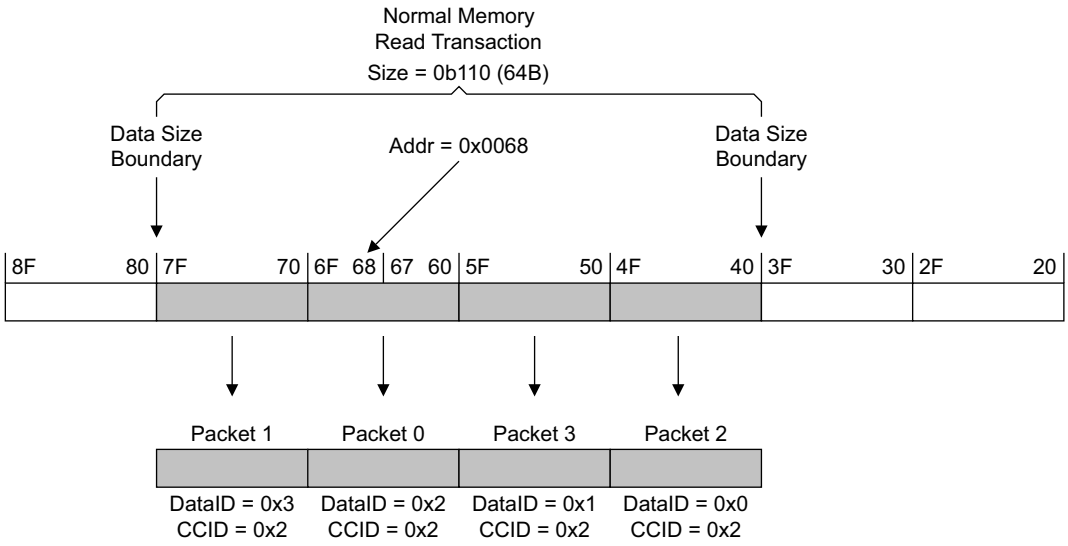
Example 2-1 Normal memory 64-byte Read transaction from an aligned address



- 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.

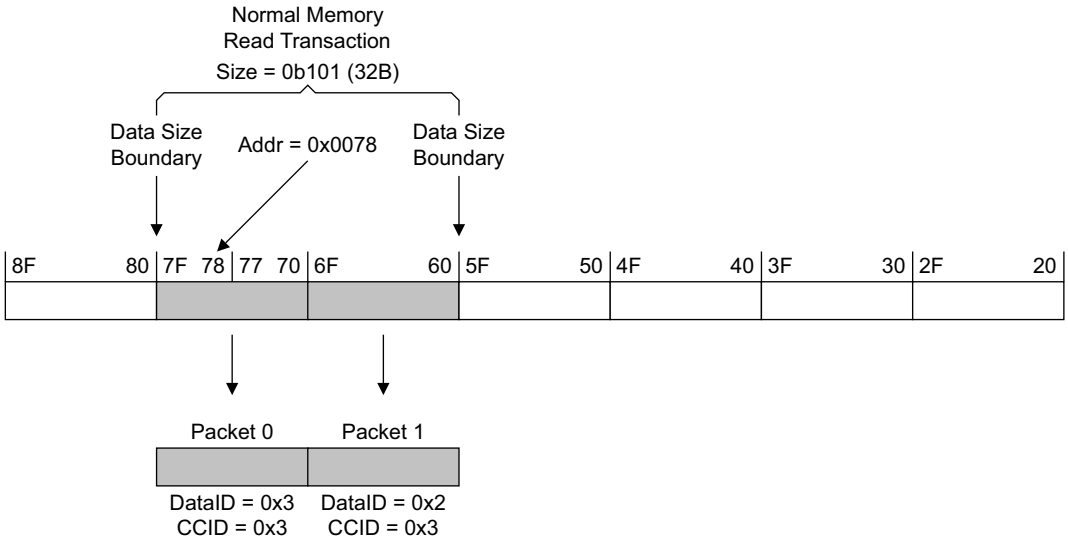
Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Example 2-2 Normal memory 64-byte Read transaction from an unaligned address



- 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.

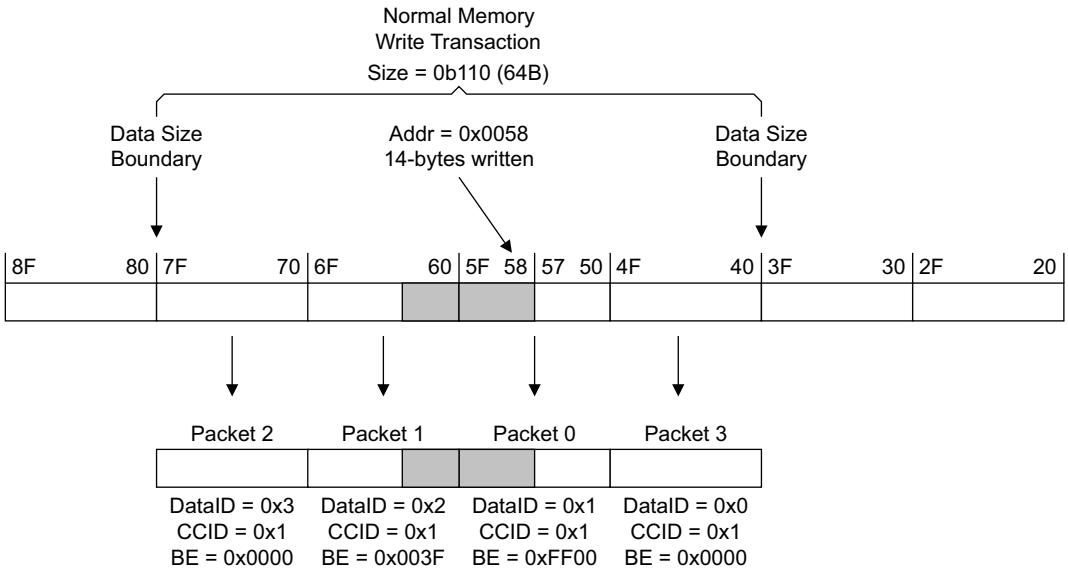
Example 2-3 Normal memory 32-byte Read transaction from an unaligned address



- 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

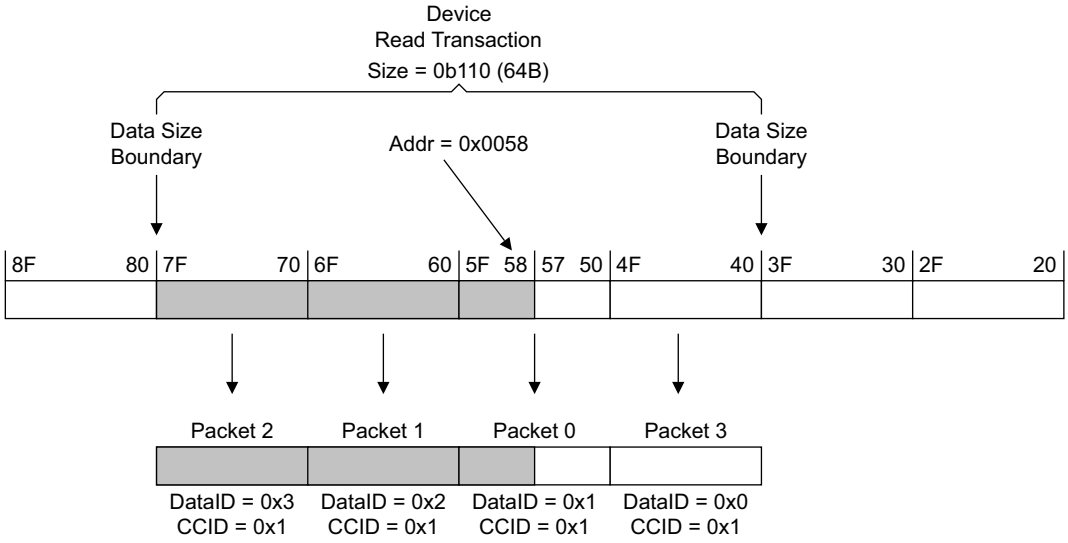
Example 2-4 Normal memory 14-byte consecutive write transaction from an unaligned address



- 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 byte enables. However, other combinations of byte enables are permitted. See *Byte Enables* on page 2-116.

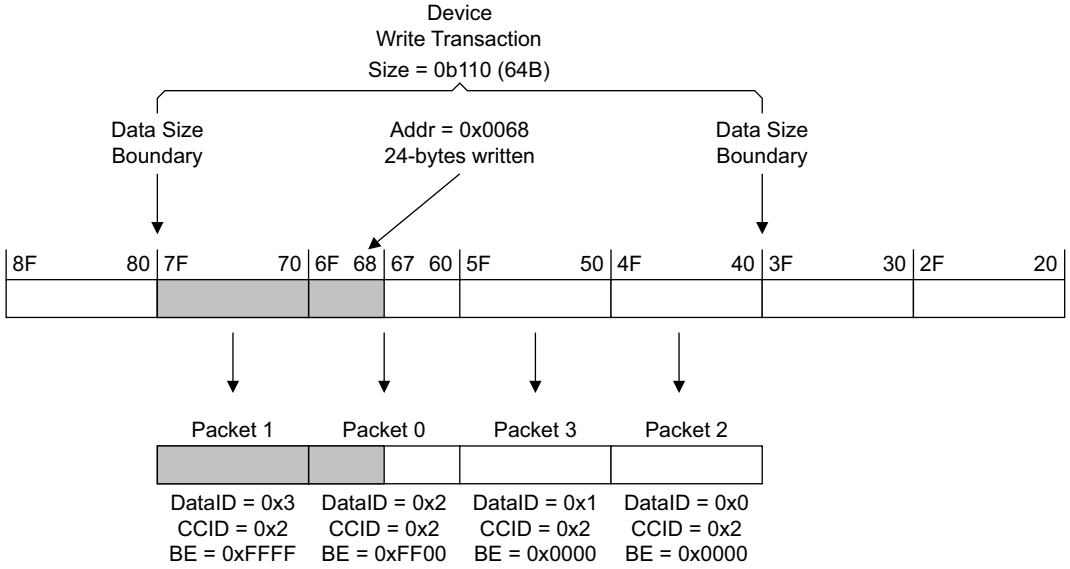
Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Example 2-5 Device Read transaction from an unaligned address



- 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.

Example 2-6 Device write transaction to an unaligned address



- Byte enables 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 byte enables meeting this criteria are asserted.
- Byte enables for bytes below the start address must not be asserted.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.11 Request Retry

This specification provides a request retry mechanism that ensures when a request reaches a Completer it is either accepted, or is given a RetryAck response, to prevent blocking of the REQ channel.

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 it receives a response indicating that the request has either been accepted, or must be sent again at a later point in time. To meet this requirement, with the exception of 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 it is not able to accept. Typically, it will not be able to accept a request when it has limited resources and insufficient storage to hold the current request until some earlier transactions have completed.

When a Completer gives a RetryAck response it is responsible for recording where the request came from, as determined by the SrcID of the request. The Completer is also responsible for determining and recording the type of Protocol Credit required to process the request. The PCrdType field in the RetryAck encodes the type of Protocol Credit that will be granted by the Completer. When required resources become available, at a later point in time, the Completer must then 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.

It is possible that a reordering interconnect can reorder the responses such that the PCrdGrant is received by the Requester before the RetryAck response for the transaction is received. In this case, the Requester must record the credit it has received, including the credit type, so that it can assign the credit appropriately when it does receive the RetryAck response.

Note

It is expected to be rare that a PCrdGrant would be re-ordered with respect to a RetryAck response, as the delay between a RetryAck and a PCrdGrant response will typically be much longer than any delay caused by interconnect re-ordering.

When the Requester receives a credit, it can then resend the request with an indication that it has been allocated a credit. This is done by deasserting the AllowRetry field. This 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 for the following:

- TgtID. See *Target ID determination for Request messages* on page 3-134.
- QoS.
- TxnID.
- ReturnTxnID for ReadNoSnp and ReadNoSnpSep from HN to SN.
- RSVDC.
- AllowRetry, which must be deasserted.
- PCrdType, which must be set to the value in the Retry response for the original transaction.
- TraceTag.

There is no fixed relationship between credits and particular transactions. If a Requester has received multiple RetryAck responses for different transactions and it 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 received a RetryAck response with that particular Protocol Credit Type.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The retry mechanism supports up to sixteen different credit types. This lets the Completer use different credit types for different resources. For example, a Completer might 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, this specification recommends that the PCrdType value of `0b0000` is used. See *PCrdType* on page 2-128.

A Completer that is giving RetryAck responses must be able to record all the RetryAck responses that it has given to ensure it can correctly distribute credits. 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 transactions it issues so that the Completer is never required to track more than 256 transactions that require a PCrdGrant response. This is achieved by limiting the maximum number of outstanding transactions to 256 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, DBIDResp, Comp, and CompDBIDResp.
- It receives RetryAck and PCrdGrant and is either:
  - Retried using a credit of the appropriate PCrdType and then is fully completed as determined by the return of all responses.
  - Cancelled and returns the received credit using the PCrdReturn message.

A Requester can reuse the TxnID value used by a request either:

- As soon as it receives a RetryAck response for that request.
- As soon as it receives all the required responses 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 Chapter 10 *Quality of Service* for further details.

2.11.1 Credit Return

It is possible for a Requester to be given more credits than it requires.

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 it 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 been given a RetryAck response then it must be acceptable for both transactions 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* (NOP) 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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.

2.11.2 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.

AllowRetry

The AllowRetry field indicates if the Request transaction can be given a RetryAck response. See Table 12-27 on page 12-325 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.

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 *PCrdType* on page 12-328 for the PCrdType value encodings.
- For destinations that have a single credit class, or do not implement credit type classification, this specification recommends 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, will 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 Chapter 10 *Quality of Service* for more details on the distribution of credits for the purposes of QoS.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

2.11.3 Transaction Retry flow

Figure 2-37 shows a typical Transaction Retry flow.

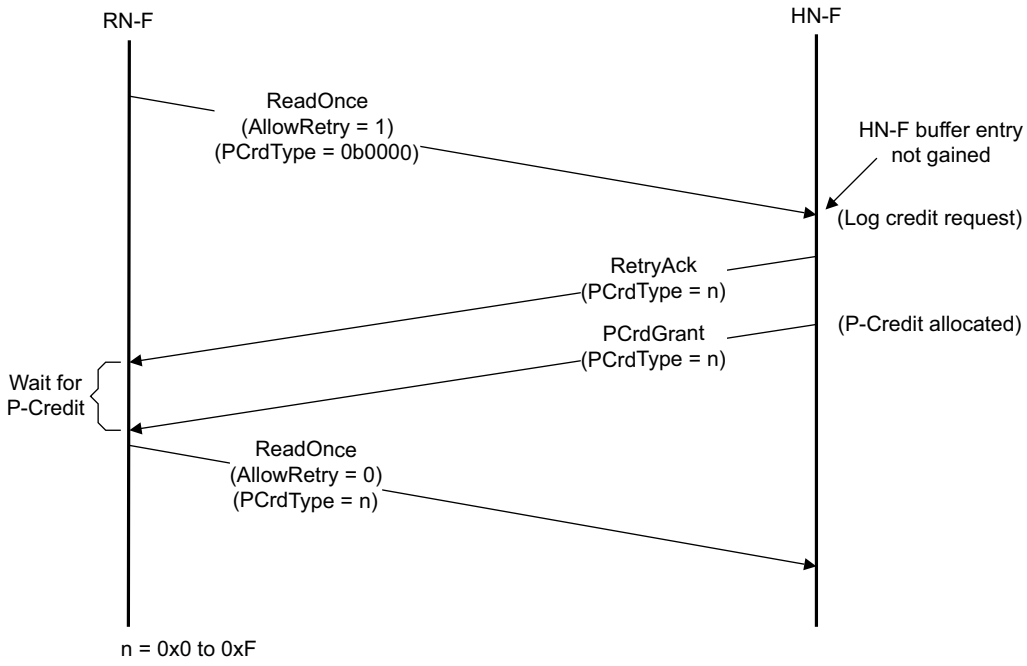


Figure 2-37 Transaction Retry flow

The steps that Figure 2-37 shows are:

1. The Requester sends a ReadOnce request.
   - This is done without a credit, so AllowRetry is asserted.
2. The Completer receives the request and sends a RetryAck response because it is not able to process the transaction at this time.
   - The request is logged and a PCrdType is determined at the Completer.
3. The Completer sends a P-Credit, using the PCrdGrant response, when it has allocated resource for the transaction.
   - The PCrdGrant includes the PCrdType allocated for the original request.
4. The Requester re-sends the 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 it has sent the associated RetryAck response.

Note

The Requester might receive PCrdGrant before RetryAck.

The second attempt at a transaction must not be sent until both a RetryAck response and an appropriate P-Credit is received for the transaction.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 3
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:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

3.1 System address map

Each Requester, that is, each RN and HN in the system, must have a *System Address Map* (SAM) to determine the target ID of a request. The scope of the SAM might 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. This specification recommends that any address that does not correspond to a physical component is sent to an agent that can provide an appropriate error response.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

3.2 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.

This specification supports a variable NodeID field width of 7 to 11 bits.

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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

3.3 Target ID determination

This section describes how the target ID is determined for the different message types. It contains the following sections:

- *Target ID determination for Request messages*.
- *Target ID determination for Response messages* on page 3-135.
- *Target ID determination for Snoop Request messages* on page 3-135.

3.3.1 Target ID determination for Request messages

For mapping of target ID in requests from the RN, this specification requires the *System Address Map* (SAM) logic to be present in the RN or in the interconnect. In the case of the interconnect, it might remap the target ID in the Request packet provided by the RN.

The target ID of a Request message is determined in the following manner using the system address map logic.

Except for PCrdReturn:

- If the request does not use a pre-allocated credit, then the target ID 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 an RN to the same Address, where one is a PrefetchTgt, target different nodes. PrefetchTgt always targets an SN. All other Requests from an RN that use an Address to Node ID mapper target an HN.
- If the request uses pre-allocated credit, the target ID of the request must be obtained from either the source ID of the RetryAck, provided as a response to the original Request message, or the target ID of the original request.

For PCrdReturn:

- The target ID provided by the RN must match the source ID included in the prior PCrdGrant which provided the credit being returned.

An RN must expect the interconnect to remap the target ID of a request.

For transactions from an RN, with the exception of PrefetchTgt which is targeted to an SN-F, this specification expects 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. This might occur, 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.

An HN might also use address map logic to determine the target Slave Node ID for each Request.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

3.3.2 Target ID determination for Response messages

Response packets are issued as a result of a received message. The target ID in Response packets must match either the SrcID or the HomeNID, or the ReturnNID or the FwdNID in the received message that resulted in the response being sent. Table 3-1 shows the source of the Response packet target ID for each Response message type and the field in the received message that determines the target ID.

Table 3-1 Source of response packet target ID

Response Message	Target ID obtained from		
	At the HN	At the SN	At the RN
RetryAck	Request.SrcID	Request.SrcID	-
PCrdGrant	Request.SrcID	Request.SrcID	-
ReadReceipt	Request.SrcID	Request.SrcID	-
RespSepData	Request.SrcID	-	-
Comp	Request.SrcID	Request.SrcID	-
DataSepResp	Request.SrcID	Request.ReturnNID	-
CompData	Request.SrcID or SnpResp*.SrcID[a]	Request.ReturnNID	Snoop.FwdNID
CompAck	-	-	Comp.SrcID or RespSepData.SrcID or CompData.HomeNID
DBIDResp	Request.SrcID	Request.SrcID	-
WriteData	-	-	DBIDResp.SrcID or CompDBIDResp.SrcID
NCBWrDataCompAck	-	-	Comp.SrcID or DBIDResp.SrcID or CompDBIDResp.SrcID
SnpResp*[b]	-	-	Snoop.SrcID

a. For Data Pull requests where Snoop response can be SnpResp or SnpRespData or SnpRespDataPtl.

b. SnpResp, SnpRespData, SnpRespDataPtl, SnpRespFwded, and SnpRespDataFwded.

3.3.3 Target ID determination for Snoop Request messages

A Snoop Request does not include a target ID. The protocol does not define an architectural mechanism to address and send a Snoop Request to a target. It is expected that this mechanism will be IMPLEMENTATION DEFINED and is outside the scope of this specification.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

3.4 Network layer flow examples

This section shows transaction flows at the network layer. It contains the following sections:

- *Simple flow*.
- *Flow with interconnect based SAM* on page 3-137.
- *Flow with interconnect based SAM and Retry request* on page 3-137.

3.4.1 Simple flow

Figure 3-1 is an example of a simple transaction flow and shows how the TgtID is determined for the requests and responses:

1. RN0 sends a request with Target ID 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 SN.
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 required, a CompAck with TgtID of HN0 derived from the HomeNID in the Data Response packet to complete the transaction.

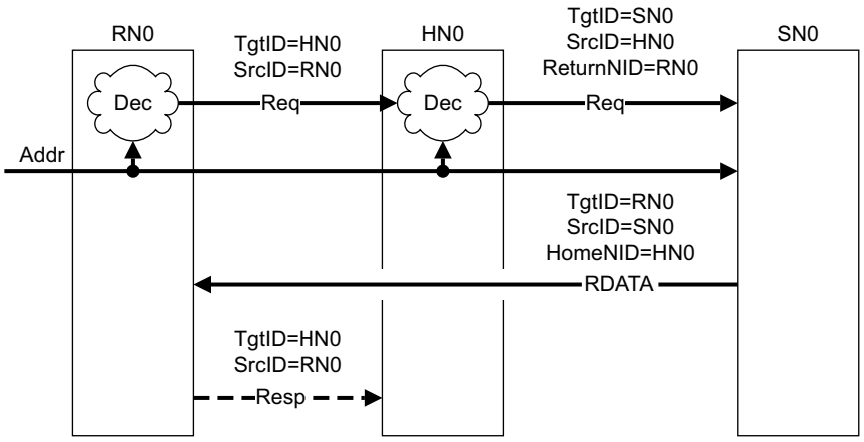


Figure 3-1 Target ID assignment without remapping

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

3.4.2 Flow with interconnect based SAM

Figure 3-2 shows a case where remapping of the target ID occurs in the interconnect.

Note

Only the target ID of the request from the RN is remapped. The TgtID in all other packets in the transaction flow is determined in a similar manner to *Simple flow* on page 3-136.

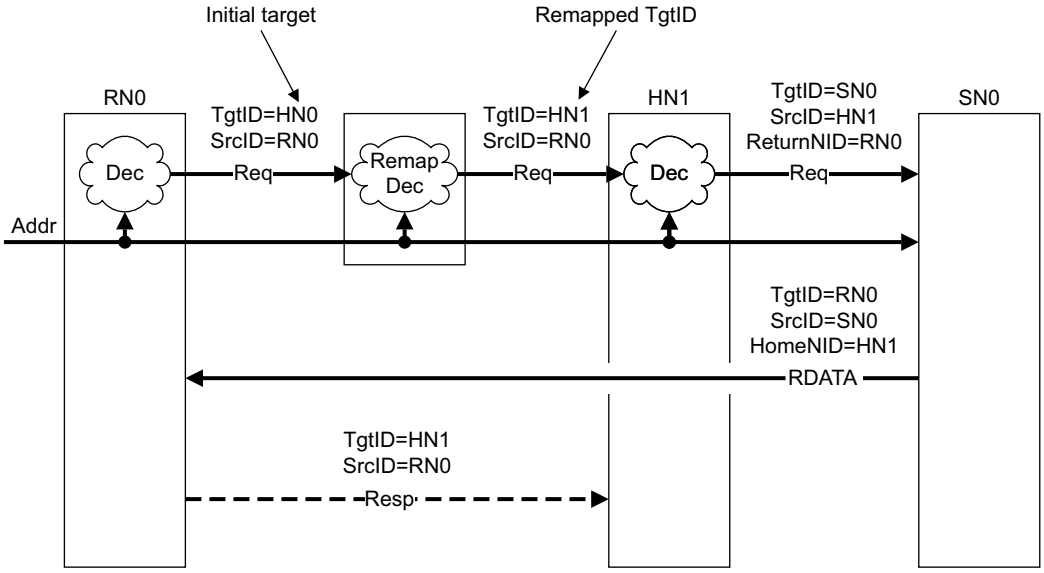


Figure 3-2 Target ID assignment with remapping logic

3.4.3 Flow with interconnect based SAM and Retry request

Figure 3-3 on page 3-138 shows a case of a request getting retried.

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 *Flow with interconnect based SAM*.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

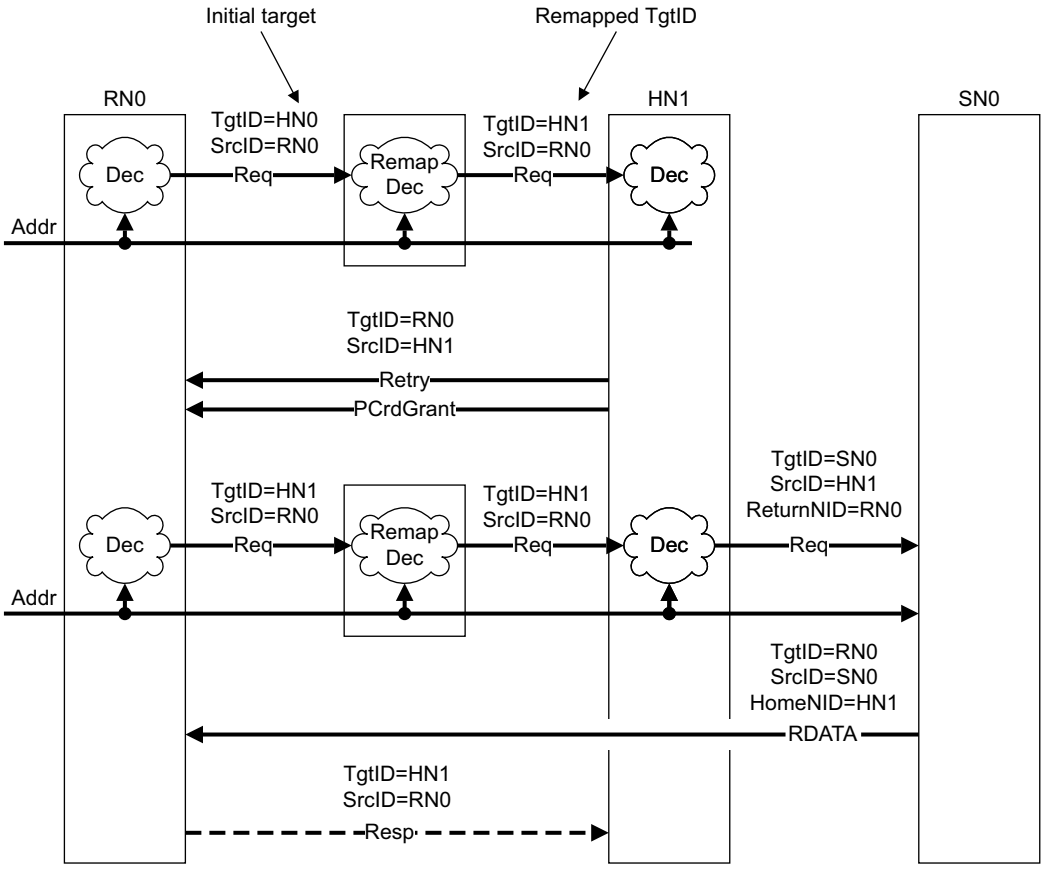


Figure 3-3 Remapping of TgtID and retried request

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 4
Coherence Protocol

This chapter describes the coherence protocol and contains the following sections:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.1 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.
  - 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. Only a part of the cache line is Valid and Dirty.
- The cache line has been modified with respect to memory.
- When the cache line is evicted, it must be merged with data from next level cache or memory 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 forward the cache line directly to the Requester even when instructed by the snoop.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SC Shared Clean:

- Other caches might have a shared copy of the cache line.
- The cache line might have been modified with respect to memory.
- It is not the responsibility of this cache 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.
  - Can return data if RetToSrc bit is set.
  - Is forwarded directly to the Requester when instructed by the snoop.

SD Shared Dirty:

- Other caches might 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.
  - Forwarded directly to the Requester when instructed by the snoop.

A cache is permitted to implement a subset of these states.

4.1.1 Empty cache line ownership

An empty cache line is a cache line that is held in a Unique state, so no other copies of the cache line exist, but none of the data bytes are Valid. This cache line state is UCE.

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 it requests permission to store, and that copy of the cache line is invalidated before the Requester obtains permission to store, this results in the Requester having an empty cache line with permission to store.

4.1.2 Ownership of cache line with partial Dirty data

Once ownership of a cache line without data is obtained, the Requester is permitted 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.2 Request types

Protocol requests are categorized as follows:

- Read requests:
  - A data response is provided to the Requester.
  - Can result in data movement among other agents in the system.
  - Can result in a cache state change at the Requester.
  - Can result in a cache state change at other Requesters in the system.
- Dataless requests:
  - No data response is provided to the Requester.
  - Can result in data movement among other agents in the system.
  - Can result in a cache state change at the Requester.
  - Can result in a cache state change at other Requesters in the system.
- Write requests:
  - Move data from the Requester.
  - Can result in data movement among other agents in the system.
  - Can result in a cache state change at the Requester.
  - Can result in a cache state change at other Requesters in the system.
- Atomic requests:
  - Move data from the Requester.
  - A data response is provided to the Requester in some Request types.
  - Can result in data movement among other agents in the system.
  - Can result in a cache state change at the Requester.
  - Can result in a cache state change at other Requesters in the system.
- Other requests:
  - Do not involve any data movement in the system.
  - Used for assisting with *Distributed Virtual Memory* (DVM) maintenance.
  - Used to warm the memory controller for a following read request.

The following subsections enumerate the resulting transactions and their characteristics.

Note

In *Read transactions* on page 4-143, *Dataless transactions* on page 4-147 and *Write transactions* on page 4-150, information is provided on the expected communicating node pairs. It is also 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 in the case of 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. See Appendix B *Communicating Nodes* for complete information on communication node pairs.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.2.1 Read transactions

ReadNoSnp

Read request by an RN to a Non-snoopable address region, or from HN to obtain a copy of the addressed data from the Slave:

- Data is included with the completion response.
- Data size is up to a cache line length, based on size attribute value in the request, irrespective of memory attributes.
- Data will not be cached at the Requester in a system coherent manner.

  Note

  It is permitted to retain a copy of the data obtained in a local cache, or buffer, but this copy of the data will not remain coherent.

- Can have exclusive attribute asserted. See Chapter 6 *Exclusive Accesses* for details.
  - Data cannot be obtained directly from the Slave Node using DMT if the Exclusive bit is set.
- Permitted, but not required, to assert ExpCompAck in the Request.
- Permitted to assert Order field in the Request.
- Permitted to use DMT if ExpCompAck is asserted in the Request.
- Permitted to use DMT if both ExpCompAck and Order are deasserted in the Request.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F, HN-I).
  - ICN(HN-F) to SN-F.
  - ICN(HN-I) to SN-I.

ReadNoSnpSep

A read request to tell the Completer to send only a Data response:

- Data only returned in the response.
- Data size is a cache line length.
- Must not assert ExpCompAck in the Request.
- The Order field of the request must be set to `b01`.
- Communicating node pairs:
  - ICN(HN-F) to SN-F.
  - ICN(HN-I) to SN-I

ReadOnce

Read request to a Snoopable address region to obtain a snapshot of the coherent data.

- Data is included with the completion response.
- Data size is a cache line length.
- Data will not be cached at the Requester.

  Note

  It is permitted to retain a copy of the data obtained in a local cache, or buffer, but this copy of the data will not remain coherent.

- Permitted, but not required, to assert ExpCompAck in the Request.
- Permitted to assert Order field in the Request.
- Permitted to use DMT if ExpCompAck is asserted in the Request.
- Permitted to use DMT if both ExpCompAck and Order are deasserted in the Request.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F).

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ReadOnceCleanInvalid

Read request to a Snoopable address region to obtain a snapshot of the coherent data:

- Data is included with the completion response.
- Data size is a cache line length.
- Data will not be cached in a coherent state at the Requester.
- Permitted, but not required to assert ExpCompAck in the request.
- Permitted to assert Order field in the request.
- Permitted to use DMT if ExpCompAck is asserted in the Request.
- Permitted to use DMT if both ExpCompAck and Order field are deasserted in the Request.
- It is recommended, but not required that a snooped cached copy is invalidated.
- If a Dirty copy is invalidated, it must be written back to memory.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F).

Note

ReadOnceCleanInvalid is used instead of ReadOnce or ReadOnceMakeInvalid where the application determines that the data is still Valid, but will not be 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 invalidation in the ReadOnceCleanInvalid transaction is a hint. Completion of the transaction does not guarantee 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.
- Apart from the interaction with Exclusive accesses, the ReadOnceCleanInvalid transaction only provides a hint for deallocation of a cache line and has no other impact on the correctness of a system.

ReadOnceMakeInvalid

Read request to a Snoopable address region to obtain a snapshot of the coherent data:

- Data is included with the completion response.
- Data size is a cache line length.
- Data will not be cached in a coherent state at the Requester.
- Permitted, but not required, to assert ExpCompAck in the Request.
- Permitted to assert Order field in the Request.
- Permitted to use DMT if ExpCompAck is asserted in the Request.
- Permitted to use DMT if both ExpCompAck and Order field are deasserted in the Request.
- It is recommended, but not required, that all snooped cached copies are invalidated.
- If a Dirty copy is invalidated, it does not need to be written back to memory.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F).

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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, 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 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 it is known that the loss of a Dirty cache line is harmless.
- For a ReadOnceMakeInvalid transaction, it is required that the invalidation of the cache line is committed prior to the read data response for the transaction. The invalidation of the cache line is not required to have completed at this point, but it must be ensured that any later write transaction from any agent, which starts after this point, is guaranteed not to be invalidated by this transaction.

ReadClean

Read request to a Snoopable address region:

- Data is included with the completion response.
- Data size is a cache line length.
- Data must be provided to the Requester in clean state only:
  - UC, or SC.
- Can have exclusive attribute asserted. See Chapter 6 *Exclusive Accesses* for details.
  - Data cannot be obtained directly from the Slave Node using DMT if the Exclusive bit is set.
- Communicating node pairs:
  - RN-F to ICN(HN-F).

ReadNotSharedDirty

Read request to a Snoopable address region.

- Data is included with the completion response.
- Data size is a cache line length.
- Requester will accept the data in any valid state except SD:
  - UC, UD, SC.
- Can have exclusive attribute asserted. See Chapter 6 *Exclusive Accesses* for details.
  - Data cannot be obtained directly from the Slave Node using DMT if the Exclusive bit is set.
- Communicating node pairs:
  - RN-F to ICN(HN-F).
- Request is included in this specification for use by caches that do not support the SharedDirty state.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ReadShared Read request to a Snoopable address region.

- Data is included with the completion response.
- Data size is a cache line length.
- Requester will accept the data in any valid state:
  - UC, UD, SC, or SD.
- Can have exclusive attribute asserted. See Chapter 6 *Exclusive Accesses* for details.
  - Data cannot be obtained directly from the Slave Node using DMT if the Exclusive bit is set.
- Communicating node pairs:
  - RN-F to ICN(HN-F).

ReadUnique Read request to a Snoopable address region to carry out a store to the cache line.

- All other cached copies must be invalidated.
- Data is included with the completion response.
- Data size is a cache line length.
- Data must be provided to the Requester in unique state only:
  - UC, or UD.
- Communicating node pairs:
  - RN-F to ICN(HN-F).

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.2.2 Dataless transactions

CleanUnique Request to a Snoopable address region to change the state 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.

- Data is not included with the completion response.
- Any dirty copy of the cache line at a snooped cache must be written back to the next level cache or memory.
- Can have exclusive attribute asserted. See Chapter 6 *Exclusive Accesses* for details.
- Communicating node pairs:
  - RN-F to ICN(HN-F).

MakeUnique Request to Snoopable address region to obtain ownership of the cache line without a data response. This request is used only when the Requester guarantees that it will carry out a store to all bytes of the cache line.

- Data is not included with the completion response.
- Any dirty copy of the cache line at a snooped cache must be invalidated without carrying out a data transfer.
- Communicating node pairs:
  - RN-F to ICN(HN-F).

Evict Used to indicate that a Clean cache line is no longer cached by an RN.

- Data is not sent for this transaction.
- The cache line must not remain in the cache.
- Communicating node pairs:
  - RN-F to ICN(HN-F).

StashOnceUnique

Request to a Snoopable address region. Request includes the Node ID of another RN and the Request can optionally include the ID of a Logical Processor within that node. It is recommended, but not required, that the other agent is snooped to indicate that it reads the addressed cache line and ensures that it is in a cache state suitable for writing to the cache line. The expected Read request is ReadUnique, or CleanUnique.

- Data is not included with the completion response.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F).

StashOnceShared

Request to a Snoopable address region. Request includes the Node ID of another RN and the Request can optionally include the ID of a Logical Processor within that node. It is recommended, but not required, that the other agent is snooped to indicate that it reads the addressed cache line. The expected Read request is ReadShared or ReadNotSharedDirty. When a valid target is not specified, then the addressed cache line can be fetched to be cached at the request Completer.

- Data is not included with the completion response.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F).

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Cache maintenance transactions

A *Cache Maintenance Operation* (CMO) assists with software cache management. The protocol includes the following four transactions to support a CMO:

CleanShared Ensures that all cached copies are changed to a Non-dirty state and any Dirty copy is written back to memory.

- Data is not included with the completion response. The Resp field value in the completion, indicating cache state, must be ignored by both the Requester and the Home.
- Sending of CleanShared to the interconnect from an RN and from the interconnect to an SN is controlled by the **BROADCASTPERSIST** (**BP**) and **BROADCASTCACHEMAINTENANCE** (**BCM**) interface signals.

  See *Optional interface broadcast signals* on page 15-366.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F, HN-I).
  - ICN(HN-F) to SN-F.
  - ICN(HN-I) to SN-I.

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).

- Data is not included with the completion response. The Resp field value in the completion, indicating cache state, must be ignored by both the Requester and the Home.
- Sending of CleanSharedPersist to the interconnect from an RN and from the interconnect to an SN is controlled by the **BP** and **BCM** interface signals.

  See *Optional interface broadcast signals* on page 15-366.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F, HN-I).
  - ICN(HN-F) to SN-F.
  - ICN(HN-I) to SN-I.

CleanInvalid Invalidates all cached copies and any Dirty copy is written to memory.

- Data is not included with the completion response.
- Sending of CleanInvalid to the interconnect from an RN and from the interconnect to an SN is controlled by the **BP** and **BCM** interface signals.

  See *Optional interface broadcast signals* on page 15-366.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F, HN-I).
  - ICN(HN-F) to SN-F.
  - ICN(HN-I) to SN-I.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MakeInvalid Invalidates all cached copies and any Dirty copy can be discarded.

- Data is not included with the completion response.
- Sending of MakeInvalid to the interconnect from an RN and from the interconnect to an SN is controlled by the **BP** and **BCM** interface signals.

  See *Optional interface broadcast signals* on page 15-366.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F, HN-I).
  - ICN(HN-F) to SN-F.
  - ICN(HN-I) to SN-I.

ExpectCompAck must not be asserted in CleanShared, CleanInvalid, MakeInvalid, and CleanSharedPersist transactions for any Requester type,

Note

Permitting CMOs to be forwarded downstream of the Home Node incorporates system topologies where some observers might directly access locations downstream of the Home Node and software cache maintenance is required to make cached data visible to such observers.

A CMO intended for a particular address must not be sent to the interconnect before all previous transactions sent to the same address have completed.

A transaction, except Evict, WriteEvictFull, and PrefetchTgt, intended for a particular address, must not be sent to the interconnect before a previous CMO sent to the same address has completed.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.2.3 Write transactions

Write transactions move data from a Requester to a Completer, this might 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 byte enables with the data.

WriteNoSnpFull Write a full cache line of data from an RN to a Non-snoopable address region, or a write for a full cache line of data from Home to Slave.

- Data size is a cache line length.
- All byte enables must be asserted.
- Can have the exclusive attribute asserted. See Chapter 6 *Exclusive Accesses*.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F, HN-I).
  - ICN(HN-F) to SN-F.
  - ICN(HN-I) to SN-I.

WriteNoSnpPtl Write a partial cache line of data from an RN to Non-snoopable address region or a write for a partial cache line of data from Home to Slave.

- Data size is up to a cache line length.
- Byte enables must be asserted for the appropriate byte lanes within the specified data size and deasserted in the rest of the data transfer.
- Can have the exclusive attribute asserted. See Chapter 6 *Exclusive Accesses*.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F, HN-I).
  - ICN(HN-F) to SN-F.
  - ICN(HN-I) to SN-I.

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.

- Data size is a cache line length.
- All byte enables must be asserted.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F).

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.

- Data size is up to a cache line length.
- Byte enables must be asserted for the appropriate byte lanes within the specified data size and deasserted in the rest of the data transfer.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F).

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 read the addressed cache line. The expected Read request is ReadUnique.

- Data size is a cache line length.
- All byte enables must be asserted.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F).

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 read the addressed cache line. The expected Read request type is ReadUnique.

- Data size is up to a cache line length.
- Byte enables must be asserted for the appropriate byte lanes within the specified data size and deasserted in the remainder of the data transfer.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F).

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 byte enables with the data. CopyBack transactions do not require the snooping of other agents in the system.

WriteBackFull Write-back a full cache line of Dirty data to the next level cache or memory.

- Data size is a cache line length.
- All byte enables must be asserted except when the write data is CopyBackWrData_I.
- The cache line must not remain in the cache.
- Communicating node pairs:
  - RN-F to ICN(HN-F).

WriteBackPtl Write-back up to a cache line of Dirty data to the next level cache or memory.

- Data size is a cache line length.
- All appropriate byte enables, up to all 64, must be asserted.
- The cache line must not remain in the cache.
- Communicating node pairs:
  - RN-F to ICN(HN-F).

WriteCleanFull Write-back a full cache line of Dirty data to the next level cache or memory and retain a Clean copy in the cache.

- Data size is a cache line length.
- All byte enables must be asserted except when the write data is CopyBackWrData_I.
- The cache line is expected to be in Clean state at completion of the transaction.
- Communicating node pairs:
  - RN-F to ICN(HN-F).

WriteEvictFull Write-back of UniqueClean data to the next-level cache.

- Data size is a cache line length.
- All byte enables must be asserted except when the write data is CopyBackWrData_I.
- The cache line must not remain in the cache.
- The cache line must not propagate beyond its Snoop domain.
- Communicating node pairs:
  - RN-F to ICN(HN-F).

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.2.4 Atomic transactions

An Atomic transaction permits a Requester to send to the interconnect a transaction with a memory address and an operation to be performed on that memory location. 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 might rely on Exclusive reads and writes.

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 then 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 PoS or 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 so 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.

Atomic transaction types

This specification defines four Atomic transaction types:

- AtomicStore.
- AtomicLoad.
- AtomicSwap.
- AtomicCompare.

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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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, an HN or an SN, 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.
- Outbound data size is 1, 2, 4, or 8 byte.
- Only appropriate byte enables must be asserted.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F, HN-I).
  - ICN(HN-F) to SN-F.
  - ICN(HN-I) to SN-I.
- Number of operations supported is 8.

Table 4-1 shows the eight operations supported by the AtomicStore transaction.

Table 4-1 AtomicStore operations

STADD	• Update location with (TxnData + InitialData). • InitialData is not returned to the Requester.
STCLR	• Update location with (InitialData AND (NOT TxnData)). Bitwise. • InitialData is not returned to the Requester.
STEOR	• Update location with (InitialData XOR TxnData). Bitwise. • InitialData is not returned to the Requester.
STSET	• Update location with (InitialData OR TxnData). Bitwise. • InitialData is not returned to the Requester.
STSMAX	• Update location with TxnData if: — (((Signed INT) TxnData – (Signed INT) InitialData) > 0). • InitialData is not returned to the Requester.
STSMIN	• Update location with TxnData if: — (((Signed INT) TxnData – (Signed INT) InitialData) < 0). • InitialData is not returned to the Requester.
STUMAX	• Update location with TxnData if: — (((Unsigned INT) TxnData – (Unsigned INT) InitialData) > 0). • InitialData is not returned to the Requester.
STUMIN	• Update location with TxnData if: — (((Unsigned INT) TxnData – (Unsigned INT) InitialData) < 0). • InitialData is not returned to the Requester.

Each of the AtomicStore operations apply to 1, 2, 4, or 8 byte data sizes.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

AtomicLoad

- Sends a single data value with an address and the atomic operation to be performed.
- The target, an HN or an SN, 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.
- Data will not be cached at the Requester.
- Outbound data size is 1, 2, 4, or 8 byte.
- Only appropriate byte enables must be asserted.
- Inbound data size is the same as the outbound data size.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F, HN-I).
  - ICN(HN-F) to SN-F.
  - ICN(HN-I) to SN-I.
- Number of operations supported is 8.

Table 4-2 shows the eight operations supported by the AtomicLoad transaction.

Table 4-2 AtomicLoad operations

LDADD	• Update location with (TxnData + InitialData). • Return InitialData to the Requester.
LDCLR	• Update location with (InitialData AND (NOT TxnData)). Bitwise. • Return InitialData to the Requester.
LDEOR	• Update location with (InitialData XOR TxnData). Bitwise. • Return InitialData to the Requester.
LDSET	• Update location with (InitialData OR TxnData). Bitwise. • Return InitialData to the Requester.
LDSMAX	• Update location with TxnData if: — (((Signed INT) TxnData – (Signed INT) InitialData) > 0). • Return InitialData to the Requester.
LDSMIN	• Update location with TxnData if: — (((Signed INT) TxnData – (Signed INT) InitialData) < 0). • Return InitialData to the Requester.
LDUSMAX	• Update location with TxnData if: — (((Unsigned INT) TxnData – (Unsigned INT) InitialData) > 0). • Return InitialData to the Requester.
LDUMIN	• Update location with TxnData if: — (((Unsigned INT) TxnData – (Unsigned INT) InitialData) < 0). • Return InitialData to the Requester.

Each of the AtomicLoad operations apply to 1, 2, 4, or 8 byte data sizes.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

AtomicSwap

- Sends a single data value, the swap value, together with the address of the location to be operated on.
- The target, an HN or an SN, 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.
- Data will not be cached at the Requester.
- Outbound data size is 1, 2, 4, or 8 byte.
- Only appropriate byte enables must be asserted.
- Inbound data size is the same as the outbound data size.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F, HN-I).
  - ICN(HN-F) to SN-F.
  - ICN(HN-I) to SN-I.
- 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, an HN or an SN, 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.
- Data will not be cached at the Requester.
- Outbound data size is 2, 4, 8, 16, or 32 byte.
- Only appropriate byte enables must be asserted.
- Inbound data size is half of the outbound data size.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to ICN(HN-F, HN-I).
  - ICN(HN-F) to SN-F.
  - ICN(HN-I) to SN-I.
- Number of operations supported is 1.

The request/response rules for Atomic transactions are:

- Transaction ordering is supported for Atomic transactions:
  - In Atomic transactions to a Normal memory region, only Request Order is permitted to be asserted, Endpoint Order must not be asserted.
  - In Atomic transactions to a Device region, both Request Order and Endpoint Order are permitted to be asserted.
- For Atomic transactions with Request Order asserted, receiving of DBIDResp at the Requester is sufficient to provide the appropriate request ordering guarantees. See *Ordering requirements* on page 2-100.
- The Completer must wait for all snoop responses before sending the Comp or CompData response.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When the Slave Node supports the execution of atomic operations, the Home is permitted to forward Atomic transactions to the Slave Node. Home cannot use DMT for Non-store atomics that are forwarded to SN. The rules governing such forwarding are:

- The Atomic transaction must be sent to the Slave Node only after all the required Snoop transactions are completed and any Dirty cached data is written back to the Slave Node.
- The Slave Node can either send a separate Comp and DBIDResp or a combined CompDBIDResp as a response to the Atomic Store transaction. For AtomicLoad, AtomicSwap, and AtomicCompare transactions the Slave node must send DBIDResp and Comp with Data as CompData. CompData must use the value of ReturnNID and ReturnTxnID from the request as the value of TgtID and TxnID respectively.

  Note

  By separating the Comp and DBIDResp responses, the Slave Node has an opportunity to signal an error in the received data, or an error during execution of the atomic operation.

- Home must send Atomic transaction data after receiving DBIDResp without waiting for completion.
- Home is permitted to send the completion response to the Requester without waiting for the initiation or completion of the Atomic transaction at the Slave Node.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.2.5 Other transactions

This section describes the protocol transactions that carry out miscellaneous actions.

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.

- Communicating node pairs:
  - RN-F, RN-D to ICN(MN).

Prefetch transaction

The prefetch target transaction is used to speculatively read data from main memory.

PrefetchTgt Prefetch Target. A Request to a Snoopable memory address, sent from a Request Node directly to a Slave Node:

- The PrefetchTgt transaction does not include a response.
- The request can be used by the Slave Node to fetch the data from off-chip memory and buffer it in anticipation of a subsequent Read request to the same location.
- The request does not include a response, therefore the Requester can deallocate the request as soon as the request is sent.
- The following fields are inapplicable and can take any value:
  - TxnID.
  - Order.
  - Endian.
  - Size.
  - MemAttr.
  - SnpAttr.
  - Excl.
  - LikelyShared.

  Note

  Prior to CHI Issue C, the Transaction ID field is inapplicable and must be set to zero.
- The Receiver must not send any response, including RetryAck.
- 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 Slave Node resources waiting indefinitely for a future Read request to the same address.
- Communicating node pairs:
  - RN-F, RN-D, RN-I to SN-F.

The receiver must accept the request without any dependency on receiving of a subsequent Read request to the same address.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.3 Snoop request types

The ICN generates a snoop request either in response to a request from an RN or due to an internal 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.

This specification permits snoops to Non-snoopable address locations.

SnpOnceFwd, SnpOnce

Snoop request to obtain the latest copy of the cache line, preferably without changing the state of the cache line at the Snoopee:

- SnpOnceFwd is permitted to be sent only to one RN-F.
- See *Forwarding type snoops* on page 4-187 for the permitted responses to SnpOnceFwd.
- See *Snoop request transactions* on page 4-179 for the permitted responses to SnpOnce.
- Expected not to change cache state.

SnpStashUnique

Snoop request recommending that the Snoopee obtains a copy of the cache line in a Unique state:

- Permitted to be sent to only one RN-F.
- This specification recommends not sending the snoop for a StashOnceUnique request if the cache line is cached in Unique state at the Stash target.
- Permitted 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 if the value of the DoNotDataPull field is `0b0` in the Snoop request.
- If not using Data Pull, then this specification recommends, but it is not required, that the Snoopee uses ReadUnique to prefetch the cache line.
- 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:

- Permitted to be sent to only one RN-F.
- This specification recommends not sending 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 if the value of the DoNotDataPull field is `0b0` in the Snoop request.
- If not using Data Pull, then this specification recommends, but it is not required, that the Snoopee uses ReadShared, or ReadSharedNotDirty to prefetch the cache line.
- Must not change the cache line state at the Snoopee.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SnpCleanFwd, SnpClean

Snoop request to obtain a copy of the cache line in Clean state while leaving any cache copy in Shared state:

- SnpCleanFwd is permitted to be only sent to one RN-F.
- See *Forwarding type snoops* on page 4-187 for permitted responses to SnpCleanFwd.
- See *Snoop request transactions* on page 4-179 for permitted responses to SnpClean.
- 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:

- SnpNotSharedDirtyFwd is permitted to be sent only to one RN-F.
- See *Forwarding type snoops* on page 4-187 for permitted responses to SnoopNotSharedDirtyFwd.
- See *Snoop request transactions* on page 4-179 for permitted responses to SnpNotSharedDirty.

SnpSharedFwd, SnpShared

Snoop request to obtain a copy of the cache line in Shared state while leaving any cached copy in Shared state:

- SnpSharedFwd is permitted to be only sent to one RN-F.
- See *Forwarding type snoops* on page 4-187 for permitted responses to SnpSharedFwd.
- See *Snoop request transactions* on page 4-179 for permitted responses to SnpShared.
- 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:

- SnpUniqueFwd is permitted to be sent to only one RN-F.
- See *Forwarding type snoops* on page 4-187 for permitted responses to SnpUniqueFwd.
- See *Snoop request transactions* on page 4-179 for permitted responses to SnpUnique.
- Must change the cache line to Invalid state.

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:

- Permitted to be sent to only one RN-F.
- Snoop response can include data.
- See *Stash type snoops* on page 4-184 for responses to SnpUniqueStash.
- Permits the Snoop response to include a Data Pull if the value of the DoNotDataPull field in the Snoop request is `0b0`.
- If not using Data Pull, then this specification recommends, but it is not required, that the Snoopee uses ReadUnique to prefetch the cache line.

SnpCleanShared Snoop request to remove any Dirty copy of the cache line at the Snoopee:

- Snoop response can include data.
- See *Snoop request transactions* on page 4-179 for permitted SnpCleanShared responses.
- Must not leave the cache line in a Dirty state.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SnpCleanInvalid Snoop request to Invalidate the cache line at the Snoopee and obtain any Dirty copy. Might also be generated by the ICN without a corresponding request:

- Snoop response can include data.
- See *Snoop request transactions* on page 4-179 for permitted SnpCleanInvalid responses.
- 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:

- Permitted to be sent to only one RN-F.
- Snoopee must not return data with the Snoop response, Dirty data must be discarded.
- See *Stash type snoops* on page 4-184 for the permitted SnpMakeInvalidStash responses.
- Permits the Snoop response to include a Data Pull if the value of the DoNotDataPull field in the Snoop request is `0b0`.
- If not using Data Pull, then this specification recommends, but it is not required, that the Snoopee uses ReadUnique to prefetch the cache line.

SnpDVMOp Generated at the ICN, initiated by the DVMOp request:

- A single DVMOp request generates two snoop requests.
- Returns a single Snoop response for the two snoop requests.

  See *Non-sync type DVM transaction flow* on page 8-252.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.4 Request types and corresponding snoop requests

Table 4-3 shows the Request transactions and the corresponding Snoop transactions that are generated by the interconnect. A Requester can implement a subset of the Request transactions but must be able to respond to all Snoop transactions.

Table 4-3 Request types and the corresponding snoop requests

Request type	Request	Snoop		
		Expected	Alternative snoop	Snoop to non-target
Read	ReadNoSnp	n/a	SnpOnceFwd	n/a
	ReadNoSnpSep	n/a	n/a	n/a
	ReadOnce	SnpOnceFwd	SnpOnce	n/a
	ReadOnceCleanInvalid	SnpUnique	SnpOnceFwd	n/a
	ReadOnceMakeInvalid	SnpUnique	SnpOnceFwd	n/a
	ReadClean	SnpCleanFwd	SnpClean	n/a
	ReadNotSharedDirty	SnpNotSharedDirtyFwd	SnpNotSharedDirty	n/a
	ReadShared	SnpSharedFwd	SnpShared	n/a
	ReadUnique	SnpUniqueFwd	SnpUnique	n/a
Dataless	CleanUnique	SnpCleanInvalid	n/a	n/a
	MakeUnique	SnpMakeInvalid	n/a	n/a
	Evict	n/a	n/a	n/a
	CleanShared	SnpCleanShared	n/a	n/a
	CleanSharedPersist	SnpCleanShared	n/a	n/a
	CleanInvalid	SnpCleanInvalid	n/a	n/a
	MakeInvalid	SnpMakeInvalid	n/a	n/a
Dataless-Stash	StashOnceUnique	SnpStashUnique	n/a	n/a
	StashOnceShared	SnpStashShared	n/a	n/a
Write	WriteNoSnp	n/a	n/a	n/a
	WriteUniqueFull	SnpMakeInvalid	n/a	n/a
	WriteUniquePtl	SnpCleanInvalid or SnpUnique	n/a	n/a
Write-Stash	WriteUniqueFullStash	SnpMakeInvalidStash	n/a	SnpMakeInvalid
	WriteUniquePtlStash	SnpUniqueStash	n/a	SnpUnique
Write - CopyBack	WriteBack	n/a	n/a	n/a
	WriteClean	n/a	n/a	n/a
	WriteEvictFull	n/a	n/a	n/a

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 4-3 Request types and the corresponding snoop requests (continued)

Request type	Request	Snoop		
		Expected	Alternative snoop	Snoop to non-target
Atomic	AtomicStore	SnpUnique	n/a	n/a
	AtomicLoad	SnpUnique	n/a	n/a
	AtomicSwap	SnpUnique	n/a	n/a
	AtomicCompare	SnpUnique	n/a	n/a
Others	DVMOp	SnpDVMOp	n/a	n/a
	PCrdReturn	n/a	n/a	n/a
	PrefetchTgt	n/a	n/a	n/a

The interconnect has the following behavior when generating a snoop request on receipt of a request from an RN:

- 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 ICN 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, then 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 SharedClean (SC) state, then the interconnect does not send a snoop request.
- It is permitted for the interconnect to generate a snoop request spontaneously without a corresponding request from an RN. 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.
- This specification permits 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 and if the cache line is dirty then 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 might result in an unnecessary data transfer. This example shows the disadvantage of using SnpUnique instead of SnpCleanInvalid in certain circumstances.
- This specification permits the interconnect to:
  - Use SnpNotSharedDirty or SnpShared or SnpClean for ReadNotSharedDirty, ReadShared and ReadClean transactions.
  - Use any non-Forwarding snoop types except SnpMakeInvalid for the ReadOnce, ReadOnceCleanInvalid and ReadOnceMakeInvalid transactions.
  - Use Forwarding snoop type SnpOnceFwd for the ReadOnce, ReadOnceCleanInvalid and ReadOnceMakeInvalid transactions.
  - Send SnpStashUnique or SnpMakeInvalidStash to the target RN for WriteUniqueFullStash and WriteUniquePtlStash if the target RN 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.5 Response types

Each request can generate one or more responses. Some responses can also include data. A Response is classified as follows:

- *Completion response*.
- *WriteData response* on page 4-165.
- *Snoop response* on page 4-167.
- *Miscellaneous response* on page 4-173.

4.5.1 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. The Requester might, however, still need to 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 *Ordering* on page 2-96 for details on the Ordering guarantees.

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.

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 zero or the same as in the corresponding DataSepResp.

Table 4-4 shows the permitted Read transaction completion, the encoding of the Resp field, and the meaning of the response. An SN can send DataSepResp only in response to ReadNoSnpSep, it must send CompData in response to ReadNoSnp.

Table 4-4 Permitted Read transaction completion and Resp field encodings

Response	Resp[2:0]	Description
CompData_I DataSepResp_I	0b000	Indicates that a coherent copy of the cache line cannot be kept.
RespSepData_I	0b000	Cache state in this response is not applicable. Cache state must be determined from DataSepResp response.
CompData_UC DataSepResp_UC RespSepData_UC	0b010	The final state of the cache line can be 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 will not be coherent. Responsibility for a Dirty cache line is not being passed.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 4-4 Permitted Read transaction completion and Resp field encodings (continued)

Response	Resp[2:0]	Description
CompData_SC DataSepResp_SC RespSepData_SC	0b001	The final state of the cache line can be SC or I. Responsibility for a Dirty cache line is not being passed.
CompData_UD_PD	0b110	The final state of the cache line can be UD or SD. Responsibility for a Dirty cache line is being passed.
CompData_SD_PD	0b111	The final state of the cache line must be SD. Responsibility for a Dirty cache line is being passed.

In a response with an error indication, the cache state is permitted to be any value, including reserved values. See *Errors and transaction structure* on page 9-274.

Dataless transaction completion

A Dataless transaction completion is sent on the CRSP channel and uses the Comp 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 is ignored and the cache state remains unchanged.

Note

Dataless transactions do not pass responsibility for a Dirty cache line.

Table 4-5 shows the permitted Dataless transaction completion, the encoding of the Resp field, and the meaning of the response.

Table 4-5 Permitted Dataless transaction completion and Resp field encodings

Response	Resp[2:0]	Description
Comp_I	0b000	The final state of the cache line must be I
Comp_UC	0b010	The final state of the cache line can be UC, UCE, SC or I
Comp_SC	0b001	The final state of the cache line can be SC or I

In a response with an error indication, the cache state is permitted to be any value, including reserved values. See *Errors and transaction structure* on page 9-274.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Write and Atomic transaction completion

A Write and AtomicStore Completion is sent on the CRSP channel and uses the Comp or CompDBID 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 a Write transaction completion. All cache state information and responsibility for a Dirty cache line are communicated with the WriteData, See *WriteData response*.

The permitted Write transaction completion responses are:

Comp Used when the Completion response is separate from the DBIDResp response.

CompDBIDResp Used when the Completion response is combined with the DBIDResp response.

All CopyBack requests must use the CompDBIDResp completion response.

Non-CopyBack writes and AtomicStore, can either send Comp and DBIDResp responses separately or can opportunistically combine the Comp and DBIDResp responses and send CompDBIDResp if both are ready to be sent to the Requester.

Miscellaneous transaction completion

A Comp response, with the Resp field set to zero, is always used for DVM transaction completion.

4.5.2 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 response sent from the Completer.

The WriteData response is sent on the WDAT channel and uses the following opcodes.

CopyBackWrData

- Used for WriteBack, WriteClean, and WriteEvictFull 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.

NonCopyBackWrData

- Used for WriteUnique and WriteNoSnp transactions.
- Also used for a DVMOp transaction.
- The cache state in the response must be I.

NCBWrDataCompAck

- Used for WriteUnique transactions.
- Combined NonCopyBackWrData and CompAck.
- The 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 4-6 shows the permitted WriteData responses, the Opcode and Resp field encodings, and the meaning of the response.

Table 4-6 Permitted WriteData responses and Opcode and Resp field encodings

Response	DAT Opcode	Resp[2:0]	Description
CopyBackWrData_I	0x2	0b000	Data corresponding to a CopyBack request. Cache line state when data was sent is I.
CopyBackWrData_UC	0x2	0b010	Data corresponding to a CopyBack request. Cache line state when data was sent is UC.
CopyBackWrData_SC	0x2	0b001	Data corresponding to a CopyBack request. Cache line state when data was sent is SC.
CopyBackWrData_UD_PD	0x2	0b110	Data corresponding to a CopyBack request. Cache line state when data was sent is UD or UDP. Responsibility for updating the memory is passed.
CopyBackWrData_SD_PD	0x2	0b111	Data corresponding to a CopyBack request. Cache line state when data was sent is SD. Responsibility for updating the memory is passed.
NonCopyBackWrData	0x3	0b000	Data corresponding to a Non-CopyBack Write request.
NCBWrDataCompAck	0xC	0b000	Data corresponding to a Non-CopyBack Write request and combined CompAck to indicate that the transaction has completed.

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. It can be determined if the cache line remains 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.

The cache line state associated with a WriteData completion can be any value when the WriteData RespErr field indicates there is a data error.

A Requester of any RN type can choose to cancel a WriteUniquePtl, WriteUniquePtlStash, or WriteNoSnpPtl after sending the Write request and before sending the Write data. The DAT channel message WriteDataCancel is used to inform the Home that the Write request is canceled.

The WriteDataCancel response rules are:

- Can be used in WriteUniquePtl, WriteUniquePtlStash and WriteNoSnpPtl transactions only.
- Must not be used in WriteNoSnp transactions with Device memory type.
- All data packets originally intended to be transferred must be sent.
- In WriteNoSnpPtl transactions RN must wait for DBIDResp and must not wait for Comp before sending either non-canceled or canceled Data.
- In WriteUniquePtl and WriteUniquePtlStash transactions, RN must wait for DBIDResp and must not wait for Comp before sending either non-canceled or canceled Data.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- WriteDataCancel message that is visible at external interfaces must have its BE and Data field values zeroed.

  External interfaces include:
  - External RN to ICN.
  - ICN to an external SN.

4.5.3 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.
- It can include a Data Pull request for stashing snoops.
- Snoop response without data is always used for the response to a SnpDVMOp transaction.

Snoop response without Data to Home and *Direct Cache Transfer* (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.
- It can include a Data Pull request for stashing snoops.

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.
- It can include a Data Pull request for stashing snoops.
- 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 ICN.

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.

These attributes convey sufficient information for the interconnect to determine the appropriate response to the initial Requester, and to determine if data must be written back to memory. It is also sufficient to support snoop filter or directory maintenance in the interconnect.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 write data 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 4-7 shows the permitted Non-forward type snoop responses without data, the RSP Opcode and Resp field encodings, and the meaning of the response.

Table 4-7 Permitted Non-forward type snoop responses without data

Response	RSP Opcode	Resp[2:0]	Description
SnpResp_I	0x1	0b000	Snoop response without data. Cache line state is I.
SnpResp_SC	0x1	0b001	Snoop response without data. Cache line state is SC, or I.
SnpResp_UC	0x1	0b010	Snoop response without data. Cache line state is UC, UCE, SC, or I.
SnpResp_UD	0x1	0b010	Snoop response without data. Cache line state is UD.
SnpResp_SD	0x1	0b011	Snoop response without data. Cache line state is SD.

Table 4-8 shows the permitted Forward type snoop responses without data, the RSP Opcode, Resp, and FwdState field encodings, and the meaning of the response.

Table 4-8 Permitted Forward type snoop responses without data

Response	RSP Opcode	Resp[2:0]	FwdState[2:0]	Description
SnpResp_I_Fwded_I	0x9	0b000	0b000	Snoop response without data. Cache line state is I. Copy of data forwarded to the Requester. Forward State is I.
SnpResp_I_Fwded_SC	0x9	0b000	0b001	Snoop response without data. Cache line state is I. Copy of data forwarded to the Requester. Forward State is SC.
SnpResp_I_Fwded_UC	0x9	0b000	0b010	Snoop response without data. Cache line state is I. Copy of data forwarded to the Requester. Forward State is UC.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 4-8 Permitted Forward type snoop responses without data (continued)

Response	RSP Opcode	Resp[2:0]	FwdState[2:0]	Description
SnpResp_I_Fwded_UD_PD	0x9	0b000	0b110	Snoop response without data. Cache line state is I. Copy of data forwarded to the Requester. Forward State is UD. Responsibility for updating the memory is passed.
SnpResp_I_Fwded_SD_PD	0x9	0b000	0b111	Snoop response without data. Cache line state is I. Copy of data forwarded to the Requester. Forward State is SD. Responsibility for updating the memory is passed.
SnpResp_SC_Fwded_I	0x9	0b001	0b000	Snoop response without data. Cache line state is SC. Copy of data forwarded to the Requester. Forward State is I.
SnpResp_SC_Fwded_SC	0x9	0b001	0b001	Snoop response without data. Cache line state is SC. Copy of data forwarded to the Requester. Forward State is SC.
SnpResp_SC_Fwded_SD_PD	0x9	0b001	0b111	Snoop response without data. Cache line state is SC. Copy of data forwarded to the Requester. Forward State is SD. Responsibility for updating the memory is passed.
SnpResp_UC_Fwded_I SnpResp_UD_Fwded_I	0x9	0b010	0b000	Snoop response without data. Cache line state is UC or UD. Copy of data forwarded to the Requester. Forward State is 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	Snoop response without data. Cache line state is SD. Copy of data forwarded to the Requester. Forward State is I.
SnpResp_SD_Fwded_SC	0x9	0b011	0b001	Snoop response without data. Cache line state is SD. Copy of data forwarded to the Requester. Forward State is SC.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 4-9 shows the permitted Non-forward type snoop responses with data, the DAT Opcode and Resp field encodings, and the meaning of the response.

Table 4-9 Permitted Non-forward type snoop responses with data

Response	DAT Opcode	Resp[2:0]	Description
SnpRespData_I	0x1	0b000	Snoop response with data. Cache line state is I.
SnpRespData_UC SnpRespData_UD	0x1	0b010	Snoop response with data. Cache line state is 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	Snoop response with data. Cache line state is SC.
SnpRespData_SD	0x1	0b011	Snoop response with data. Cache line state is SD.
SnpRespData_I_PD	0x1	0b100	Snoop response with data. Cache line state is I. Responsibility for updating the memory is passed.
SnpRespData_UC_PD	0x1	0b110	Snoop response with data. Cache line state is UC. Responsibility for updating the memory is passed.
SnpRespData_SC_PD	0x1	0b101	Snoop response with data. Cache line state is SC. Responsibility for updating the memory is passed.
SnpRespDataPtl_I_PD	0x5	0b100	Snoop response with partial data. Cache line state is I. Responsibility for updating the memory is passed.
SnpRespDataPtl_UD	0x5	0b010	Snoop response with partial data. Cache line state is UDP.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 4-10 shows the permitted Forward type snoop responses with data, the DAT Opcode, Resp, and FwdState field encodings, and the meaning of the response.

Table 4-10 Permitted Forward type snoop responses with data

Response	DAT Opcode	Resp[2:0]	FwdState[2:0]	Description
SnpRespData_I_Fwded_SC	0x6	0b000	0b001	Snoop response with data. Cache line state is I. Copy of data forwarded to the Requester. Forward State is SC.
SnpRespData_I_Fwded_SD_PD	0x6	0b000	0b111	Snoop response with data. Cache line state is I. Copy of data forwarded to the Requester. Forward State is SD. Responsibility for updating the memory is passed.
SnpRespData_SC_Fwded_SC	0x6	0b001	0b001	Snoop response with data. Cache line state is SC. Copy of data forwarded to the Requester. Forward State is SC.
SnpRespData_SC_Fwded_SD_PD	0x6	0b001	0b111	Snoop response with data. Cache line state is SC. Copy of data forwarded to the Requester. Forward State is SD. Responsibility for updating the memory is passed.
SnpRespData_SD_Fwded_SC	0x6	0b011	0b001	Snoop response with data. Cache line state is SD. Copy of data forwarded to the Requester. Forward State is SC.
SnpRespData_I_PD_Fwded_I	0x6	0b100	0b000	Snoop response with data. Cache line state is I. Responsibility for updating the memory is passed. Copy of data forwarded to the Requester. Forward State is I.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 4-10 Permitted Forward type snoop responses with data (continued)

Response	DAT Opcode	Resp[2:0]	FwdState[2:0]	Description
SnpRespData_I_PD_Fwded_SC	0x6	0b100	0b001	Snoop response with data. Cache line state is I. Responsibility for updating the memory is passed. Copy of data forwarded to the Requester. Forward State is SC.
SnpRespData_SC_PD_Fwded_I	0x6	0b101	0b000	Snoop response with data. Cache line state is SC. Responsibility for updating the memory is passed. Copy of data forwarded to the Requester. Forward State is I.
SnpRespData_SC_PD_Fwded_SC	0x6	0b101	0b001	Snoop response with data. Cache line state is SC. Responsibility for updating the memory is passed. Copy of data forwarded to the Requester. Forward State is SC.

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 error.

The cache line state associated with a Snoop response without data is a don't care and can take any value if the response has an error.

In responses to Stashing 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.
  - Snp_Resp_SC_Read.
  - SnpResp_SD_Read.
- For SnpStashShared:
  - SnpResp_I_Read.
  - SnpResp_UC_Read.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.4 Miscellaneous response

This section describes responses that cannot be classified as a Completion, WriteData or Snoop response.

For all responses in this section the Resp and RespErr fields have no meaning and must be set to zero.

The miscellaneous responses are:

CompAck

- Sent by the Requester on receipt of the Completion response.
- Used by Read, Dataless, and WriteUnique transactions.

See *Transaction structure* on page 2-39.

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.

See *Transaction Retry sequence* on page 2-66.

PCrdGrant

- Grants a Protocol Credit. A subsequent request, sent using the Protocol Credit, is guaranteed to be accepted by the target.

See *Transaction Retry sequence* on page 2-66.

ReadReceipt

- Sent for a request that requires Request Order in the interconnect with respect to other ordered requests from the same Requester.
- Sent by a Slave Node to indicate it has accepted a Read request and will not send a RetryAck response.
- Applies to ReadNoSnp, ReadNoSnpSep, and ReadOnce* request transactions.

See *ReadNoSnp, ReadOnce, ReadOnceCleanInvalid, ReadOnceMakeInvalid* on page 2-44.

DBIDResp

- Response sent as part of a write and an Atomic transaction to signal to the Requester that resources are available to accept the WriteData response.

See *Transaction structure* on page 2-39.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.6 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 4-11. 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 4-11 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.

The Notes column in Table 4-11 indicates how a silent cache transition can be made non-silent or visible at the interface.

Table 4-11 Legal silent cache state transitions

RN-F action	RN-F Cache state		Notes
	Present	Next	
Cache eviction	UC	I	Can use Evict or WriteEvictFull transaction
	UCE	I	Can use Evict transaction
	SC	I	Can use Evict transaction
Local sharing	UC	SC	-
	UD	SD	-
Store	UC	UD	Full or partial cache line store
	UCE	UDP	Partial cache line store
	UCE	UD	Full cache line store
	UDP	UD	Store that fills the cache line
Cache Invalidate	UD	I	Can use Evict transaction
	UDP	I	Can use Evict transaction

A cache state change from UC to UCE is not permitted.

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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.7 Cache state transitions

This section specifies the cache state transitions and completion responses for the following request transactions:

- *Read request transactions*.
- *Dataless request transactions* on page 4-177.
- *Write request transactions* on page 4-178.
- *Atomic transactions* on page 4-179.
- *Other request transactions* on page 4-179.
- *Snoop request transactions* on page 4-179.
- *Stash type snoops* on page 4-184.
- *Forwarding type snoops* on page 4-187.

4.7.1 Read request transactions

Table 4-12 shows the cache state transitions at the Requester, and the completion responses, for Read request transactions.

The cache state in the Data response to the Requester from the Slave Node is UC, that is, CompData_UC irrespective of the original request type. The Requester must ignore the cache state in the CompData 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 Slave to Home, the cache state in the response can be either I or UC, but it is expected that typically a slave design can be simplified by always using UC. Home then sends CompData to the Requester with the appropriate cache state value.

Table 4-12 Cache state transitions at the Requester for Read request transactions

Request type	Cache state				
	Initial		Final	Comp response	Separate responses[a]
	Expected	Others Permitted			
ReadNoSnp	I	-	I	CompData_UC, CompData_I	RespSepData + DataSepResp_UC
ReadOnce	I, UCE[b]	-	I	CompData_UC, CompData_I	RespSepData + DataSepResp_UC
ReadOnceCleanInvalid	I, UCE[b]	-	I	CompData_UC, CompData_I	RespSepData + DataSepResp_UC
ReadOnceMakeInvalid	I, UCE[b]	-	I	CompData_UC, CompData_I	RespSepData + DataSepResp_UC
ReadClean	I, UCE[b]	-	SC	CompData_SC	RespSepData + DataSepResp_SC
			UC	CompData_UC	RespSepData + DataSepResp_UC

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 4-12 Cache state transitions at the Requester for Read request transactions (continued)

Request type	Cache state				
	Initial		Final	Comp response	Separate responses[a]
	Expected	Others Permitted			
ReadNotSharedDirty	I, UCE[b]	-	SC	CompData_SC	RespSepData + DataSepResp_SC
			UC	CompData_UC	RespSepData + DataSepResp_UC
			UD	CompData_UD_PD	-
ReadShared	I, UCE[b]	-	SC	CompData_SC	RespSepData + DataSepResp_SC
			UC	CompData_UC	RespSepData + DataSepResp_UC
			SD	CompData_SD_PD	-
			UD	CompData_UD_PD	-
ReadUnique	I, SC	UC, UCE	UC	CompData_UC	RespSepData + DataSepResp_UC
			UD	CompData_UD_PD	-
	SD	UD, UDP[c]	UD	CompData_UC, CompData_UD_PD	RespSepData + DataSepResp_UC

a. Separate Comp and Data response is not permitted prior to CHI Issue C.

b. For ReadOnce*, ReadClean, ReadNotSharedDirty and ReadShared transactions the Requester with an initial state of UCE must not upgrade the cache line to UDP nor UD while the request is outstanding.

c. 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.

Note

- The Other Permitted initial cache states in Table 4-12 on page 4-175 are the cache states that are permitted while the transaction is in progress.
- For any of the transactions in Table 4-12 on page 4-175, it is legal to use the transaction if the cache line can silently transition to any Expected or Other Permitted state. This silent transition must occur before the transaction is issued.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.7.2 Dataless request transactions

Table 4-13 shows the cache state transitions at the Requester, and the completion responses, for Dataless request transactions.

Table 4-13 Cache state transitions at the Requester for Dataless request transactions

Request type	Cache state			Comp Response
	Initial		Final	
	Expected	Others permitted		
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
StashOnceShared	I	-	I	Comp
CleanShared CleanSharedPersist	I, SC, UC	-	No Change	Comp_UC
				Comp_SC
				Comp_I
CleanInvalid	I	-	I	Comp_I
MakeInvalid	I	-	I	Comp_I

Before a CleanInvalid, 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 4-13 shows I state as the only initial state.

Note

- The Other Permitted initial cache states in Table 4-13 are the cache states that are permitted while the transaction is in progress.
- For any of the transactions in Table 4-13, it is legal to use the transaction if the cache line can silently transition to any Expected or Other Permitted state. This silent transition must occur before the transaction is issued.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.7.3 Write request transactions

Table 4-14 shows the cache state transitions at the Requester, the Write data response, and the combined or separate Completion and DBID response for Write and WriteBack request transactions.

Table 4-14 Requester cache state transitions for Write request transactions

Request Type	Cache state at Requester			WriteData response	Comp response
	Initial	Before WriteData response[a]	Final		
WriteNoSnpPtl	I	-	I	NCBWrData or WriteDataCancel	DBIDResp + Comp or CompDBIDResp
WriteNoSnpFull	I	-	I	NCBWrData	DBIDResp + Comp or CompDBIDResp
WriteUniquePtl WriteUniquePtlStash	I	I	I	NCBWrData or WriteDataCancel or NCBWrDataCompAck[b]	DBIDResp + Comp or CompDBIDResp
WriteUniqueFull WriteUniqueFullStash	I	I	I	NCBWrData or NCBWrDataCompAck[b]	DBIDResp + Comp or CompDBIDResp
WriteBackFull	UD	UD	I	CBWrData_UD_PD	CompDBIDResp
		UC	I	CBWrData_UC	CompDBIDResp
	UD, SD	SD	I	CBWrData_SD_PD	CompDBIDResp
		SC	I	CBWrData_SC	CompDBIDResp
		I	I	CBWrData_I	CompDBIDResp
WriteBackPtl	UDP	UDP	I	CBWrData_UD_PD	CompDBIDResp
		I	I	CBWrData_I	CompDBIDResp
WriteCleanFull	UD	UD	UC	CBWrData_UD_PD	CompDBIDResp
		UC	UC	CBWrData_UC	CompDBIDResp
	UD, SD	SD	SC	CBWrData_SD_PD	CompDBIDResp
		SC	SC	CBWrData_SC	CompDBIDResp
		I	I	CBWrData_I	CompDBIDResp
WriteEvictFull	UC	UC	I	CBWrData_UC	CompDBIDResp
		SC	I	CBWrData_SC	CompDBIDResp
		I	I	CBWrData_I	CompDBIDResp

a. A snoop might be received while a write is pending and results in a cache line state change before the WriteData response.

b. NCBWrDataCompAck is not permitted prior to CHI Issue C.

Note

After completion of a WriteClean transaction, it is possible for the cache line in a Unique state to immediately transition to a Dirty state.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.7.4 Atomic transactions

Table 4-15 shows the cache state transitions at the Requester, and the completion and response for Atomic transactions.

Table 4-15 Requester cache state transitions for Atomic request transactions

Atomic request	Cache state			WriteData response	Comp response
	Initial		Final		
	Expected	Others permitted			
AtomicStore	I, SC UCE, SD	UC, UD, UDP	I	NCBWrData	DBIDResp + Comp_I or CompDBIDResp
AtomicLoad	I, SC UCE, SD	UC, UD, UDP	I	NCBWrData	DBIDResp + CompData_I
AtomicSwap	I, SC UCE, SD	UC, UD, UDP	I	NCBWrData	DBIDResp + CompData_I
AtomicCompare	I, SC UCE, SD	UC, UD, UDP	I	NCBWrData	DBIDResp + CompData_I

4.7.5 Other request transactions

DVMOp and PrefetchTgt requests do not have any cache state transitions associated with them.

4.7.6 Snoop request transactions

For snoop Non-forward requests, the response must be either a SnpResp or SnpRespData. In the case of multiple final cache state options, the response that is used is IMPLEMENTATION DEFINED.

A Request Node does not have to respond to a snoop with data when in UC state. Except for SnpOnce, the receiver of the snoop response can not differentiate between:

- A Request Node not responding with data from an UC state.
- A Request Node silently transitioning to SC or I state, prior to receiving the snoop request, and therefore not including data with the snoop response.

Table 4-16 on page 4-180 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 4-16 Cache state transitions, RetToSrc value, and valid completion responses

Snoop request type	Initial cache state	Final cache state		RetToSrc[a]	Snoop response
		Expected	Others permitted		
SnpOnce	I	I	-	X	SnpResp_I
	UC	UC	I, SC	0	SnpResp_UC
					SnpRespData_UC
				1	SnpResp_UC
					SnpRespData_UC
		SC	I	0	SnpResp_SC
					SnpRespData_SC
				1	SnpResp_SC
					SnpRespData_SC
		I	-	0	SnpResp_I
					SnpRespData_I
				1	SnpResp_I
					SnpRespData_I
	UCE	UCE	I	X	SnpResp_UC
		I	-	X	SnpResp_I
	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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 4-17 shows for SnpClean, SnpShared, and SnoopNotSharedDirty, 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.

Table 4-17 Cache state transitions, RetToSrc value, and valid completion responses

Snoop request type	Initial cache state	Final cache state		RetToSrc[a]	Snoop response
		Expected	Others permitted		
SnpClean, SnpShared, SnpNotSharedDirty	I	I	-	X	SnpResp_I
	UC	SC	I	0	SnpResp_SC
					SnpRespData_SC
				1	SnpResp_SC
					SnpRespData_SC
		I	-	0	SnpResp_I
					SnpRespData_I
				1	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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 4-18 shows for SnpUnique, 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.

Table 4-18 Cache state transitions, RetToSrc value, and valid completion responses

Snoop request type	Initial cache state	Final cache state		RetToSrc[a]	Snoop response
		Expected	Others permitted		
SnpUnique	I	I	-	X	SnpResp_I
	UC	I	-	0	SnpResp_I
					SnpRespData_I
				1	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.

Table 4-19 on page 4-183 shows for SnpCleanShared, SnpCleanInvalid, and SnpMakeInvalid 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 4-19 Cache state transitions, RetToSrc value, and valid completion responses

Snoop request type	Initial cache state	Final cache state		RetToSrc	Snoop response
		Expected	Others permitted		
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
	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

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.7.7 Stash type snoops

The following sub-sections show the permitted responses for the Stash type snoops.

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.

A Snoop response can include Data Pull only if the DoNotDataPull in the Snoop request is deasserted.

Table 4-20 shows the Snoopee cache state transitions and required Snoop responses. The Snoop responses do not include the Data Pull options. Data Pull is permitted with any Snoop response.

Table 4-20 Snoop response to SnpUniqueStash and SnpMakeInvalidStash

Snoop request type	Cache state			RetToSrc	Snoop response
	Initial	Final			
		Expected	Others permitted		
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
					SnpRespData_I
	SD	I	-	0	SnpRespData_I_PD
SnpMakeInvalidStash	Any	I	-	0	SnpResp_I

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 Data Pull only if the cache state in the response is precise and DoNotDataPull in the corresponding Snoop request is deasserted.

The inclusion of Data Pull 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 *Read transactions* on page 4-143.

Table 4-21 shows the Snoopee cache state transitions, the required Snoop responses, and Data Pull options for SnpStashUnique.

Table 4-21 Snoop response to SnpStashUnique

Snoop request type	Cache state			RetToSrc	Snoop response
	Initial	Final			
		Expected	Others permitted		
SnpStashUnique	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

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 4-22 shows the Snoopee cache state transitions, the required Snoop responses, and Data Pull options for SnpStashShared.

Table 4-22 Snoop response to SnpStashShared

Snoop request type	Cache state			RetToSrc	Snoop response
	Initial	Final			
		Expected	Others permitted		
SnpStashShared	I	I	-	0	SnpResp_I_Read
					SnpResp_I
	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_I
	SD	SD	-	0	SnpResp_SD
					SnpResp_I

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.7.8 Forwarding type snoops

Forwarding (Fwd) type snoops are used by Home to support DCT. The rules, common to all Fwd type snoops at the Snoopee are:

- Must forward a copy to the Requester if the cache line is in one of the following states:
  - UD.
  - UC.
  - SD.
  - SC.
- Not permitted to convert to the corresponding Non-fwd type snoop.
- 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 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.
- 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 Fwd type snoop see the following individual sub-sections.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 zero.
- Snoopee can ignore the DoNotGoToSD value in the snoop.

Table 4-23 shows the Snoopee cache state transition and required Snoop responses for SnpOnceFwd.

Table 4-23 Snoop response to SnpOnceFwd

Snoopee cache state			RetToSrc	Response to	
Initial	Final			Requester	Home
	Expected	Other permitted			
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
UD	UD	-	0	CompData_I	SnpResp_UD_Fwded_I
	SD	-	0	CompData_I	SnpResp_SD_Fwded_I
	SC	I	0	CompData_I	SnpRespData_SC_PD_Fwded_I
	I	-	0	CompData_I	SnpRespData_I_PD_Fwded_I
UDP	UDP	-	0	No Fwd	SnpRespDataPtl_UD
	I	-	0	No Fwd	SnpRespDataPtl_I_PD
SC	SC	I	0	CompData_I	SnpResp_SC_Fwded_I
	I	-	0	CompData_I	SnpResp_I_Fwded_I
SD	SD	-	0	CompData_I	SnpResp_SD_Fwded_I
	SC	I	0	CompData_I	SnpRespData_SC_PD_Fwded_I
	I	-	0	CompData_I	SnpRespData_I_PD_Fwded_I

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SnpCleanFwd

The rules, in addition to the common rules, to be followed by a Snoopee that receives a SnpCleanFwd 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 *Shared clean state return* on page 4-194.

Table 4-24 shows the Snoopee cache state transitions and required Snoop responses for SnpCleanFwd.

Table 4-24 Snoop response to SnpCleanFwd

Snoopee cache state			RetToSrc	Response to	
Initial	Final			Requester	Home
	Expected	Other permitted			
I	I	-	X[a]	No Fwd	SnpResp_I
UC	SC	I	0	CompData_SC	SnpResp_SC_Fwded_SC
			1	CompData_SC	SnpRespData_SC_Fwded_SC
	I	-	0	CompData_SC	SnpResp_I_Fwded_SC
			1	CompData_SC	SnpRespData_I_Fwded_SC
UCE	I	-	X[a]	No Fwd	SnpResp_I
UD	SD[b]	-	0	CompData_SC	SnpResp_SD_Fwded_SC
			1	CompData_SC	SnpRespData_SD_Fwded_SC
	SC	I	X[a]	CompData_SC	SnpRespData_SC_PD_Fwded_SC
	I	-	X[a]	CompData_SC	SnpRespData_I_PD_Fwded_SC
UDP	I	-	X[a]	No Fwd	SnpRespDataPtl_I_PD
SC	SC	I	0	CompData_SC	SnpResp_SC_Fwded_SC
			1	CompData_SC	SnpRespData_SC_Fwded_SC
	I	-	0	CompData_SC	SnpResp_I_Fwded_SC
			1	CompData_SC	SnpRespData_I_Fwded_SC
SD	SD[b]	-	0	CompData_SC	SnpResp_SD_Fwded_SC
			1	CompData_SC	SnpRespData_SD_Fwded_SC
	SC	I	X[a]	CompData_SC	SnpRespData_SC_PD_Fwded_SC
	I	-	X[a]	CompData_SC	SnpRespData_I_PD_Fwded_SC

a. The protocol requirements apply for both states of RetToSrc.
b. This state transition is not permitted if DoNotGoToSD is asserted.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SnpNotSharedDirtyFwd

The rules, in addition to the common rules, to be followed by a Snoopee that receives a SnpNotSharedDirtyFwd are:

- Snoopee must forward the cache line in SC state.
- Snoopee must transition to SD, SC or I state.
- For behavior related to the RetToSrc bit see *Shared clean state return* on page 4-194.

Table 4-25 shows the Snoopee cache state transitions and required Snoop responses for SnpNotSharedDirtyFwd.

Table 4-25 Snoop response to SnpNotSharedDirtyFwd

Snoopee cache state			RetToSrc	Response to	
Initial	Final			Requester	Home
	Expected	Other permitted			
I	I	-	X[a]	No Fwd	SnpResp_I
UC	SC	I	0	CompData_SC	SnpResp_SC_Fwded_SC
			1	CompData_SC	SnpRespData_SC_Fwded_SC
	I	-	0	CompData_SC	SnpResp_I_Fwded_SC
			1	CompData_SC	SnpRespData_I_Fwded_SC
UCE	I	-	X[a]	No Fwd	SnpResp_I
UD	SD[b]	-	0	CompData_SC	SnpResp_SD_Fwded_SC
			1	CompData_SC	SnpRespData_SD_Fwded_SC
	SC	I	X[a]	CompData_SC	SnpRespData_SC_PD_Fwded_SC
	I	-	X[a]	CompData_SC	SnpRespData_I_PD_Fwded_SC
UDP	I	-	X[a]	No Fwd	SnpRespDataPtl_I_PD
SC	SC	I	0	CompData_SC	SnpResp_SC_Fwded_SC
			1	CompData_SC	SnpRespData_SC_Fwded_SC
	I	-	0	CompData_SC	SnpResp_I_Fwded_SC
			1	CompData_SC	SnpRespData_I_Fwded_SC
SD	SD[b]	-	0	CompData_SC	SnpResp_SD_Fwded_SC
			1	CompData_SC	SnpRespData_SD_Fwded_SC
	SC	I	X[a]	CompData_SC	SnpRespData_SC_PD_Fwded_SC
	I	-	X[a]	CompData_SC	SnpRespData_I_PD_Fwded_SC

a. The protocol requirements apply for both states of RetToSrc.
b. This state transition is not permitted if DoNotGoToSD is asserted.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 *Shared clean state return* on page 4-194.

Table 4-26 shows the Snoopee cache state transition and required Snoop responses for SnpSharedFwd.

Table 4-26 Snoop response to SnpSharedFwd

Snoopee cache state			RetToSrc	Response to	
Initial	Final			Requester	Home
	Expected	Other permitted			
I	I	-	X[a]	No Fwd	SnpResp_I
UC	SC	I	0	CompData_SC	SnpResp_SC_Fwded_SC
			1	CompData_SC	SnpRespData_SC_Fwded_SC
	I	-	0	CompData_SC	SnpResp_I_Fwded_SC
			1	CompData_SC	SnpRespData_I_Fwded_SC
UCE	I	-	X[a]	No Fwd	SnpResp_I
UD	SD[b]	-	0	CompData_SC	SnpResp_SD_Fwded_SC
			1	CompData_SC	SnpRespData_SD_Fwded_SC
	SC	I	0	CompData_SD_PD	SnpResp_SC_Fwded_SD_PD
			1	CompData_SD_PD	SnpRespData_SC_Fwded_SD_PD
			X[a]	CompData_SC	SnpRespData_SC_PD_Fwded_SC
	I	-	0	CompData_SD_PD	SnpResp_I_Fwded_SD_PD
			1	CompData_SD_PD	SnpRespData_I_Fwded_SD_PD
			X[a]	CompData_SC	SnpRespData_I_PD_Fwded_SC
UDP	I	-	X[a]	No Fwd	SnpRespDataPtl_I_PD
SC	SC	I	0	CompData_SC	SnpResp_SC_Fwded_SC
			1	CompData_SC	SnpRespData_SC_Fwded_SC
	I	-	0	CompData_SC	SnpResp_I_Fwded_SC
			1	CompData_SC	SnpRespData_I_Fwded_SC

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 4-26 Snoop response to SnpSharedFwd (continued)

Snoopee cache state			RetToSrc	Response to	
Initial	Final			Requester	Home
	Expected	Other permitted			
SD	SD[b]	-	0	CompData_SC	SnpResp_SD_Fwded_SC
			1	CompData_SC	SnpRespData_SD_Fwded_SC
	SC	I	0	CompData_SD_PD	SnpResp_SC_Fwded_SD_PD
			1	CompData_SD_PD	SnpRespData_SC_Fwded_SD_PD
			X[a]	CompData_SC	SnpRespData_SC_PD_Fwded_SC
	I	-	0	CompData_SD_PD	SnpResp_I_Fwded_SD_PD
			1	CompData_SD_PD	SnpRespData_I_Fwded_SD_PD
			X[a]	CompData_SC	SnpRespData_I_PD_Fwded_SC

a. The protocol requirements apply for both states of RetToSrc.
b. This state transition is not permitted if DoNotGoToSD is asserted.

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 zero.

Table 4-27 shows the Snoopee cache state transitions and required Snoop responses for SnpUniqueFwd.

Table 4-27 Snoop response to SnpUniqueFwd

Snoopee cache state			RetToSrc	Response to	
Initial	Final			Requester	Home
	Expected	Other permitted			
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

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 4-27 Snoop response to SnpUniqueFwd (continued)

Snoopee cache state			RetToSrc	Response to	
Initial	Final			Requester	Home
	Expected	Other permitted			
UDP	I	-	0	No Fwd	SnpRespDataPtl_I_PD
SC	I	-	0	CompData_UC	SnpResp_I_Fwded_UC
SD	I	-	0	CompData_UD_PD	SnpResp_I_Fwded_UD_PD

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.8 Shared clean state return

A RetToSrc field is included in the Snoop request to instruct the Snoopee to return a copy of the cache line to Home.

The rules for returning a copy of the cache line to Home are as follows:

If the RetToSrc field value is set to 1:

- For a Forwarding snoop:
  - Must return a copy to Home if the cache line is Dirty or Clean.
- For Non-forwarding snoops SnpOnce, SnpClean, SnpNotSharedDirty, SnpShared, and SnpUnique:
  - Return a copy to Home from SC state.
  - Must return a copy to Home from UD, UDP, and SD state.
  - Optionally can return a copy to Home from UC state.

If the RetToSrc field value is set to 0:

- For a Forwarding snoop:
  - Must not return a copy to Home except when the responsibility for updating memory is being passed to Home, or the Snoopee has the cache line in UDP state and does not relinquish the state.
  - Passing of Dirty to Home is required when neither the Snoopee nor the Requester retain a Dirty copy.
- For Non-forwarding snoops SnpOnce, SnpClean, SnpNotSharedDirty, SnpShared, and SnpUnique:
  - Must not return a copy to Home from SC state.
  - Must return a copy to Home from UD, UDP, and SD state.
  - Optionally can return a copy to Home from UC state.

The RetToSrc field must be set to 0 in the following Non-forwarding snoops because these snoops never return data from Clean cache state:

- SnpStashUnique.
- SnpStashShared.
- SnpUniqueStash.
- SnpCleanShared.
- SnpCleanInvalid.
- SnpMakeInvalid.
- SnpMakeInvalidStash.

The RetToSrc field must be set to 0 in the following Forwarding snoops:

- SnpOnceFwd.
- SnpUniqueFwd.

Home must only set RetToSrc on the Snoop request to a single Request Node.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4.9 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 *Ordering* on page 2-96.

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. It is the responsibility of the interconnect, that is, ICN(HN-F, HN-I and MN), 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.

4.9.1 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 Data 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:

- 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 then the following additional requirements apply:

- Request transaction flow must be completed after receiving the CompDBIDResp.
- The cache state in the WriteData 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.
- An RN is permitted to not send valid CopyBack Data, if the cache line state after the Snoop response is sent is I or SC. The cache state in the WriteData response, after CopyBack Data is taken away by the snoop, must be I and all byte enables must be deasserted and the corresponding data must be set to zero.
- If data is included with WriteData it must be the same data that as sent with the Snoop response or more up to date data.

  Note

  More recent data than that sent with the snoop response can only be provided if the snoop was a SnpOnce, SnpOnceFwd, or SnpCleanShared and the Snoop response indicates that the cache line can be further modified.

For non-ordered transactions, the RN can send CompAck without waiting for DataSepResp. For ordered transactions, the RN can send CompAck as soon as the first packet of DataSepResp is received. In both cases, an RN must not respond to a Snoop request before receiving all data packets.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The RN-F might receive multiple snoop requests before it receives a response for a pending CopyBack request for the same cache line, in which case the data response carries the cache line state after 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.

4.9.2 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, might 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, then 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, it must send all packets of the message without dependence on completion of any other Request or Response message.

Similarly, a Home, when it accepts part of the data message, must accept the remaining packets of that message 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 it receives:

- A CompAck for any Read and Dataless requests except for ReadOnce* and ReadNoSnp.
- A WriteData response for CopyBack and Atomic requests.
- For WriteUnique, a WriteData response and, if applicable, CompAck.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 5
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:

- *Read transaction flows* on page 5-198.
- *Dataless transaction flows* on page 5-209.
- *Write transaction flows* on page 5-212.
- *Atomic transaction flows* on page 5-215.
- *Stash transaction flows* on page 5-222
- *Hazard handling examples* on page 5-225.

See *Time-Space diagrams* on page xiii for details of the conventions used to illustrate protocol flow in this specification.

In the transaction flow diagrams that follow:

- There are multiple coherent RNs, an HN-F and a SN-F.
- If the HN-F receives multiple data responses, that is, one response from a snooped RN-F and another from a SN-F, then the data being forwarded to the Requester is highlighted in bold.
- There is no ICN cache at the HN-F, this results in all requests to the HN-F initiating a request to the SN-F.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.1 Read transaction flows

This section gives examples of the interconnect protocol flow for Read transactions.

5.1.1 Read transactions with DMT and without snoops

For Read transactions without snoops, this specification recommends the use of *Direct Memory Transfer* (DMT).

Figure 5-1 shows an example DMT transaction flow using the ReadShared transaction.

In this example a response from SN-F to HN-F is not required because CompAck from the Requester is used to deallocate the request at Home.

The steps in the ReadShared transaction flow are:

1. RN-F sends a Read request to HN-F.
2. HN-F sends a Read request to SN-F.
   - The ID field values in the Read request are based on where the Data response is to be sent. Data can be sent to the Requester or to the HN-F. See Figure 2-24 on page 2-78 that shows an example of how the ID field values are derived.
3. SN-F sends a Data response directly to RN-F.
4. RN-F sends CompAck to HN-F as the Request is ReadShared and requires CompAck to complete the transaction.

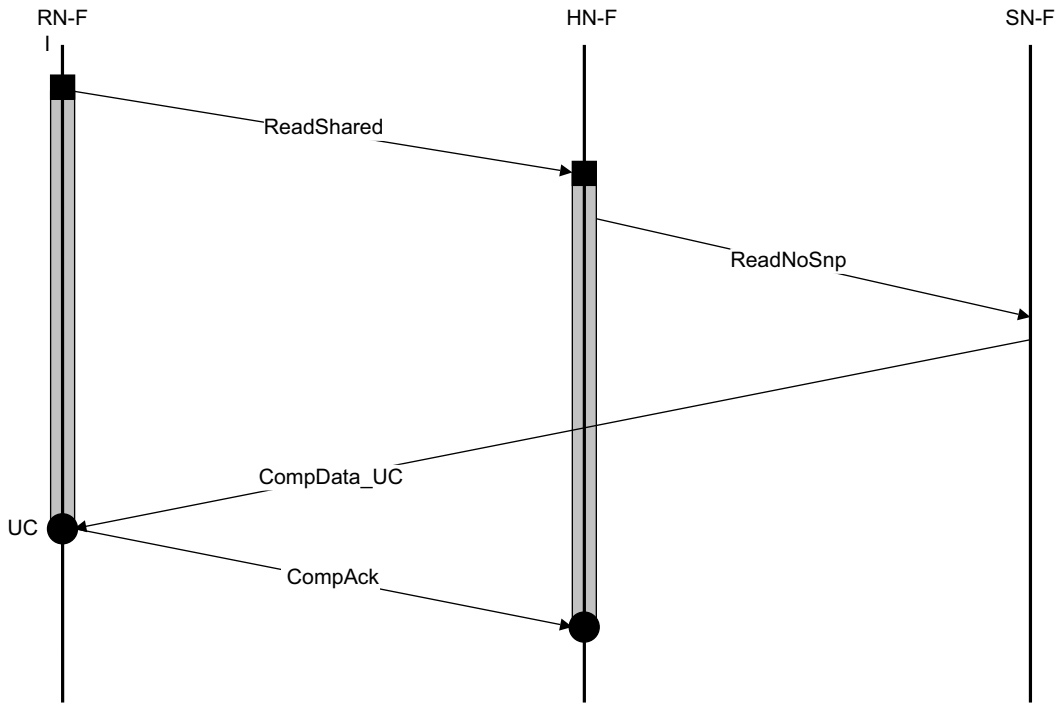


Figure 5-1 DMT Read transaction example without snoops

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.1.2 Read transaction with DMT and with snoops

For Read transactions with snoops and data from memory this specification recommends the use of DMT.

Figure 5-2 shows an example DMT transaction flow using the ReadShared transaction.

In this example a response from SN-F to HN-F is not required because CompAck from the Requester is used to deallocate the request at Home.

The steps in the ReadShared transaction flow are:

1. RN-F0 sends a Read request to HN-F.
2. HN-F sends a Snoop request to RN-F1.
3. HN-F sends a Read request to SN-F after receiving the Snoop response from RN-F1, which guarantees that RN-F1 has not responded with data.
   - The ID field values in the Read request are based on where the Data response is to be sent. Data can be sent to the Requester or to the HN-F. See Figure 2-24 on page 2-78 that shows an example of how the ID field values are derived.
4. SN-F sends a Data response directly to RN-F0.
5. RN-F0 sends CompAck to HN-F as the Request is ReadShared and requires CompAck to complete the transaction.

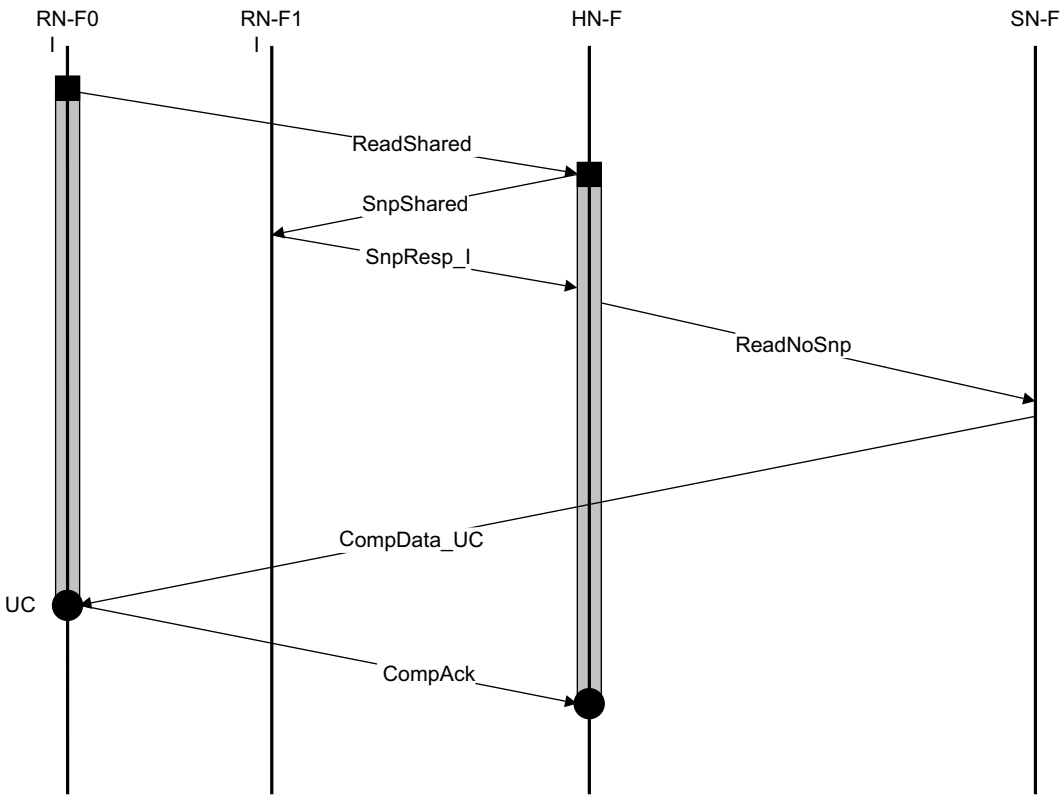


Figure 5-2 DMT Read transaction example with snoops and data from memory

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.1.3 Read transaction with DCT

For Read transactions with snoops and data from cache memory. this specification recommends use of *Direct Cache Transfer* (DCT).

DCT from cache line in UC state

Figure 5-3 on page 5-201 shows an example flow for a DCT transaction. The Requester is RN-F0 and the forwarding cache is located at RN-F1.

The steps in the DCT transaction flow are:

1. RN-F0 sends a ReadShared request to HN-F.
2. HN-F sends a SnpSharedFwd, a forward Snoop request to RN-F1.
3. RN-F1 cache line state transitions from UC to SC.
4. RN-F1 forwards CompData_SC response to RN-F0.
5. RN-F1 also sends a SnpResp_SC_Fwded_SC Snoop response to HN-F that indicates:
   - The data was forwarded to the Requester.
   - The final state of the cache line in the snooped cache is SC.
   - The state in which the cache line can be cached at the Requester is SC.
6. After receiving the CompData response RN-F0 sends a CompAck response to HN-F to conclude the transaction.

Note

Steps 4 and 5 in the DCT transaction flow can occur in any order as CompData and SnpResp are sent on different channels.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

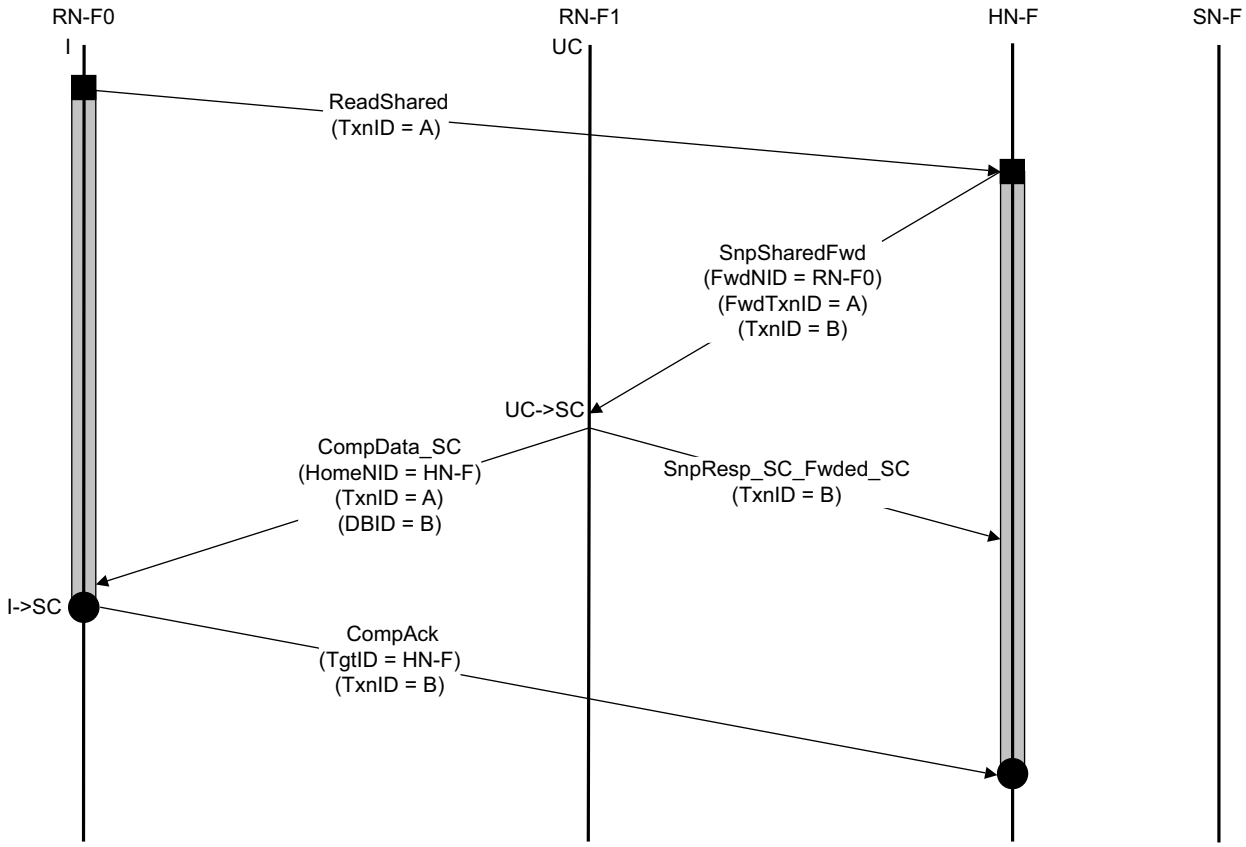


Figure 5-3 Direct Cache Transfer from cache line in UC state

Double data return in a DCT transaction

Figure 5-4 on page 5-202 shows an example DCT transaction flow that sends the data to HN-F as well as forwarding the data to the RN-F0.

The steps in the DCT transaction flow 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 cache line state transitions from UD to SC.
4. RN-F1 sends CompData_SC response to RN-F0.
5. 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:
   - The data was forwarded to the Requester.
   - The final state of the cache line in the snooped cache is SC.
   - The state in which the cache line can be cached at the Requester is SC.
6. The RN-F0 sends CompAck after it receives the Data response to conclude the transaction.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

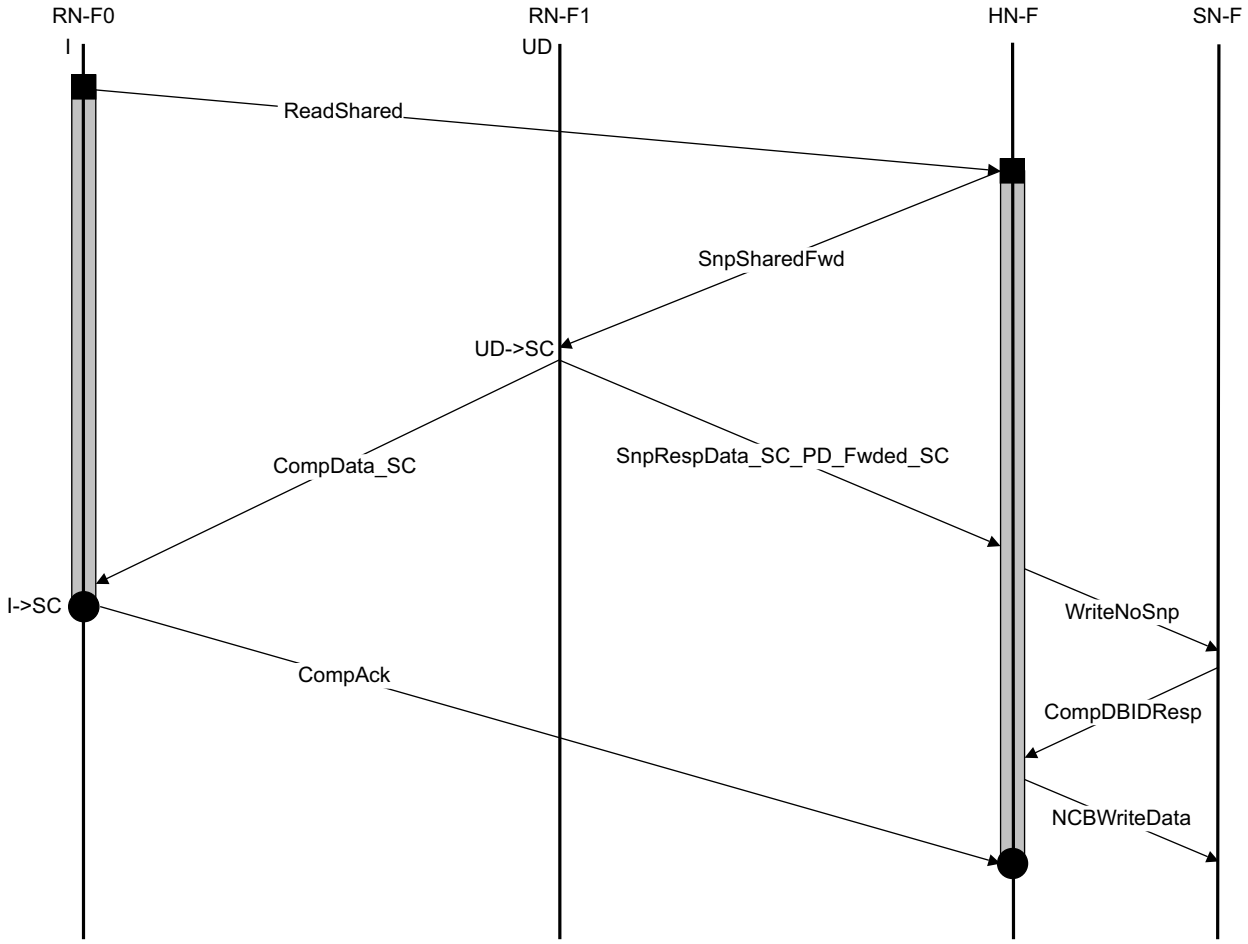


Figure 5-4 Double data return in a DCT transaction

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.1.4 Read transaction with neither DMT nor DCT

Figure 5-5 shows an example of the flow without DMT using the ReadNoSnp transaction. In this example, 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. The steps in the ReadNoSnp transaction flow are:

1. RN-F0 issues a ReadNoSnp transaction.
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 data response to HN-F.
5. HN-F in turn returns the data to RN-F0. If ExpCompAck was not asserted in the ReadNoSnp request then HN-F deallocates the request.
6. If ExpCompAck was asserted in the ReadNoSnp request, RN-F0 sends a CompAck response to HN-F.
7. RN-F0 deallocates the request.
8. HN-F receives the CompAck response and deallocates the request.

Figure 5-5 shows the transaction flow, the copy of data being transferred is marked in bold.

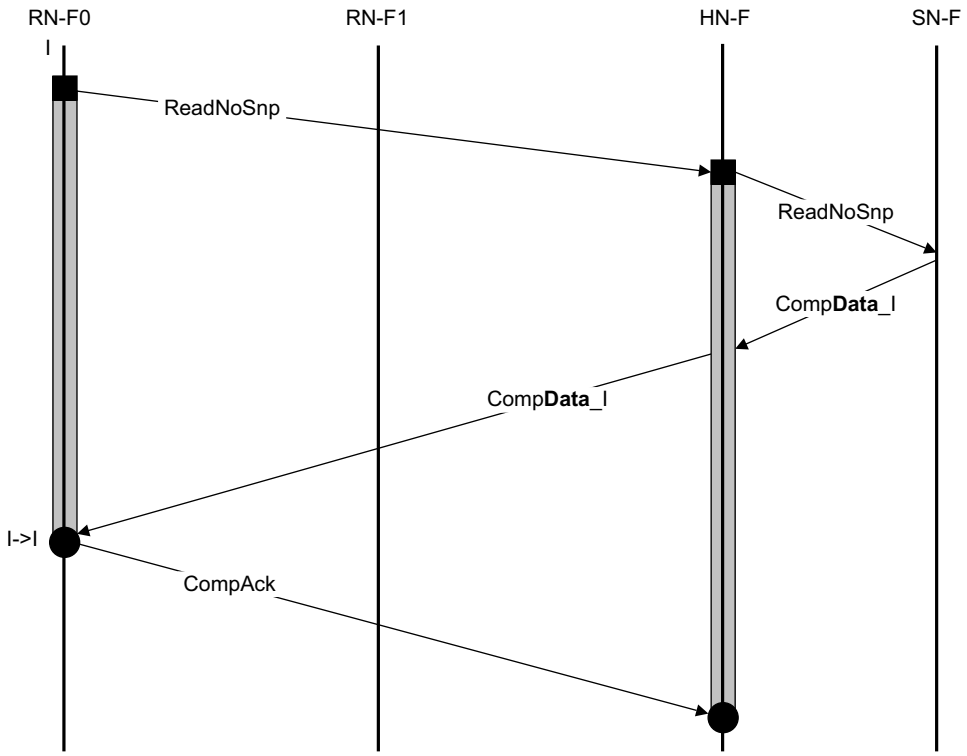


Figure 5-5 ReadNoSnp transaction flow

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.1.5 Read transaction with snoop response with partial data and no memory update

An example of this type of flow is a ReadUnique transaction.

RN-F1 has the cache line in UDP state. RN-F1 responds to the snoop with a snoop response with partial cache line data and passes responsibility for updating memory.

HN-F waits for the data response from memory, merges the partial snoop response data with the data response from memory, and sends the resultant data to the Requester.

HN-F does not update memory because responsibility for updating memory is passed on to the Requester.

Figure 5-6 shows the transaction flow, the copy of data being transferred is marked in bold.

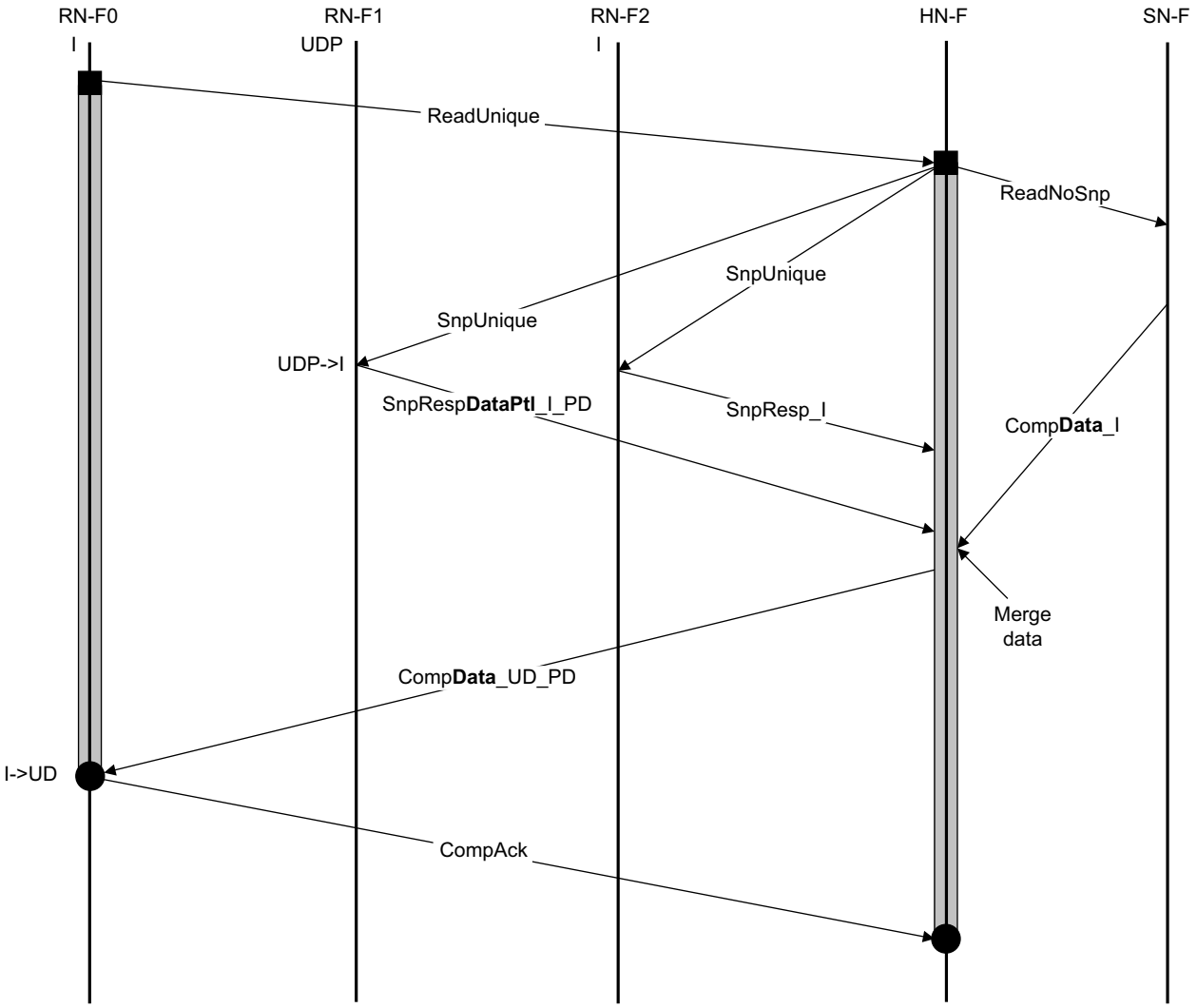


Figure 5-6 ReadUnique with partial data snoop response

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.1.6 Read transaction with snoop response with partial data and memory update.

An example of this type of flow is a ReadClean transaction.

RN-F1 has the cache line in UDP state. RN-F1 responds to the snoop with a snoop response with partial cache line data and passes responsibility for updating memory.

HN-F waits for the data response from memory, merges the partial snoop response data with the data response from memory, and sends the resultant data to the Requester.

HN-F updates memory as the responsibility for updating memory is not passed on to the Requester.

Figure 5-7 shows the transaction flow, the copy of data being transferred is marked in bold.

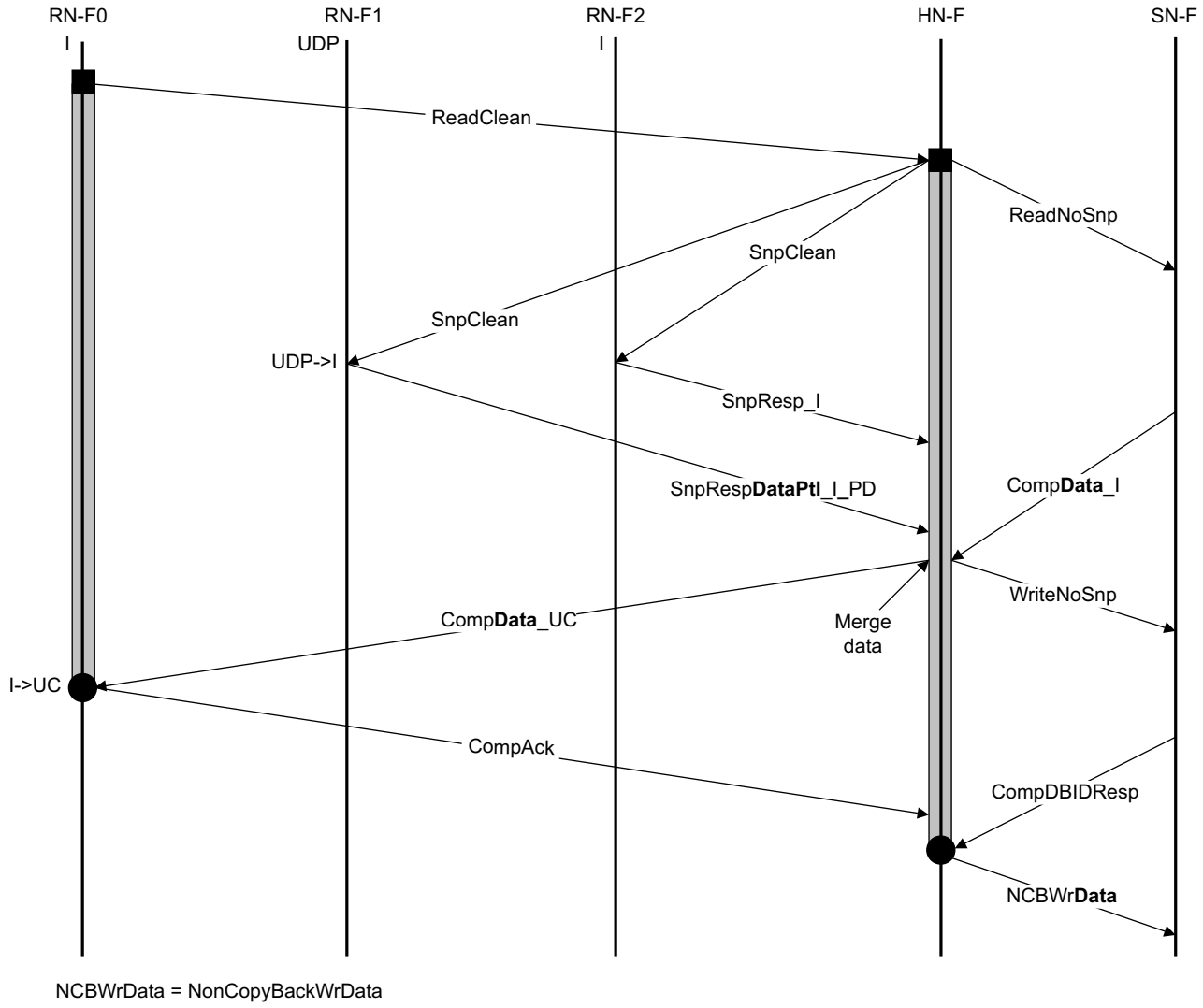


Figure 5-7 ReadClean with partial data snoop response

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.1.7 ReadOnce* and ReadNoSnp with early Home deallocation

Figure 5-8 shows the optimized flow for an unordered ReadOnce request.

The steps in the optimized ReadOnce transaction flow 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.

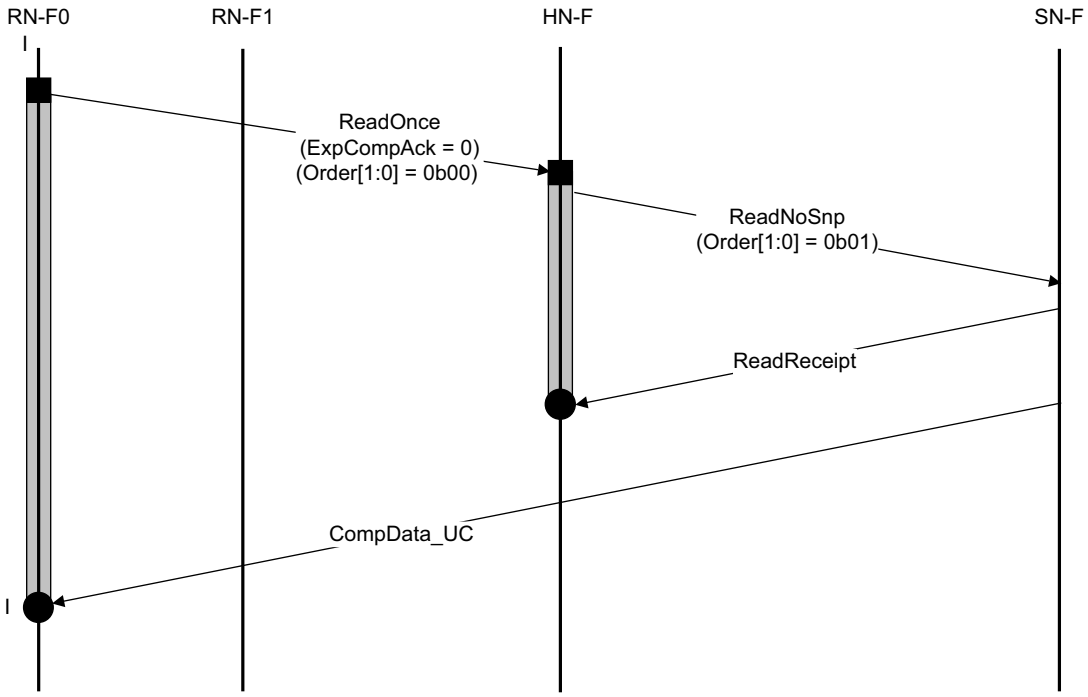


Figure 5-8 DMT optimization for unordered ReadOnce

Note

Use of a ReadNoSnp transaction from Home to Slave, in the case where CompAck is not required, avoids the need to send a RespSepData response from Home to Requester.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.1.8 ReadNoSnp transaction with DMT and separate Non-data and Data-only

Figure 5-9 shows an example DMT transaction flow with separate Non-data and Data-only.

In this example there is no ordering requirement and RN-F can send CompAck to HN-F to deallocate the request at Home without waiting for DataSepResp.

The steps in the ReadNoSnp transaction are:

1. RN-F sends a ReadNoSnp request to HN-F.
2. HN-F sends a ReadNoSnpSep request to SN-F.
   - This tells SN-F that a data only response is required.
3. HN-F sends a RespSepData response to RN-F.
   - This tells RN-F that the request has been allocated at HN-F and that there will be a separate Data response.
4. SN-F sends a ReadReceipt to HN-F.
   - This tells HN-F that the request to the Slave is completed and ensures that the SN will not respond with RetryAck for that request. As the request is a ReadNoSnp with Order field set to zero, the request at HN-F can be deallocated on receipt of ReadReceipt.
5. RN-F sends CompAck after receiving RespSepData.
   - For ReadNoSnp, CompAck is not functionally required, HN-F that deallocates the request after receiving the ReadReceipt from SN-F can just drop the CompAck.
6. SN-F, as Completer, sends DataSepResp to RN-F returning the read data.

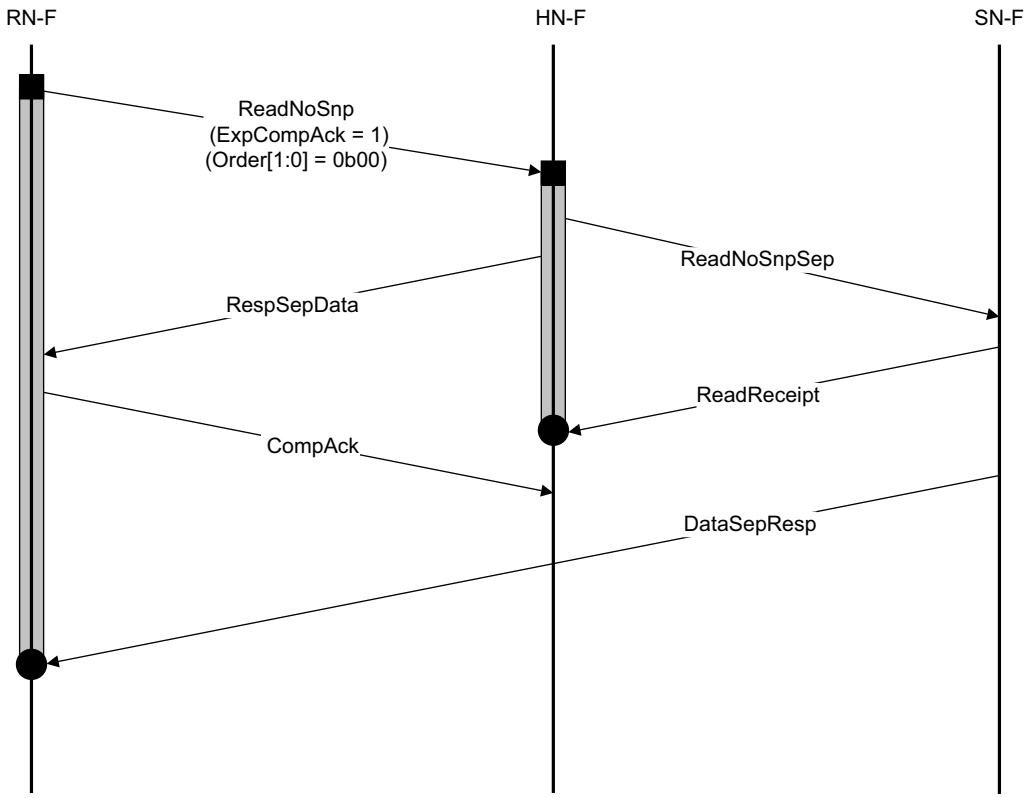


Figure 5-9 DMT Read transaction example with separate Non-data and Data-only

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.1.9 ReadNoSnp transaction with DMT with ordering and separate Non-data and Data-only

Figure 5-10 shows an example DMT transaction flow with ordering and separate Non-data and Data-only.

This example, which has ReadNoSnp with non-zero Order field 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 it receives CompAck.

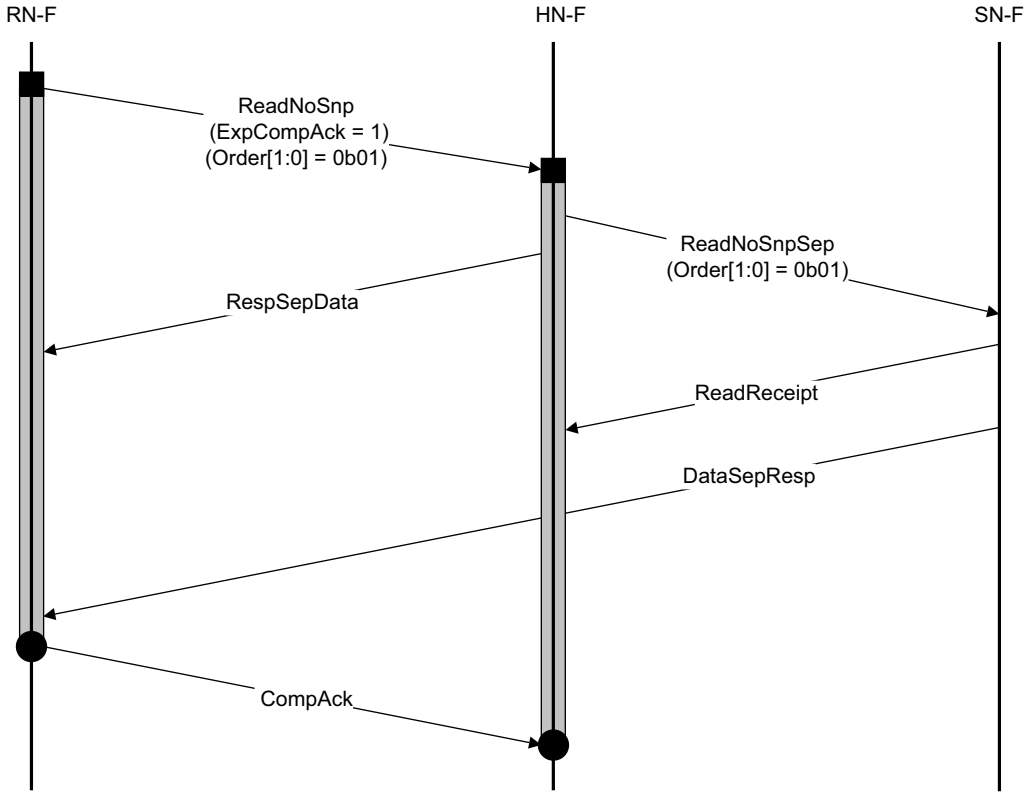


Figure 5-10 DMT Read transaction example with ordering and separate Non-data and Data-only

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.2 Dataless transaction flows

This section gives examples of the interconnect protocol flow for Dataless transactions.

5.2.1 Dataless transaction without memory update

An example of this type of flow is a MakeUnique transaction.

RN-F1 has the cache line in UC state. RN-F1 responds to the snoop with a snoop response without data and changes the cache line state to I.

HN-F waits for all snoop responses and then sends a Comp_UC response to the Requester.

HN-F does not send a read request to SN-F because the request is a Dataless transaction.

Figure 5-11 shows the transaction flow.

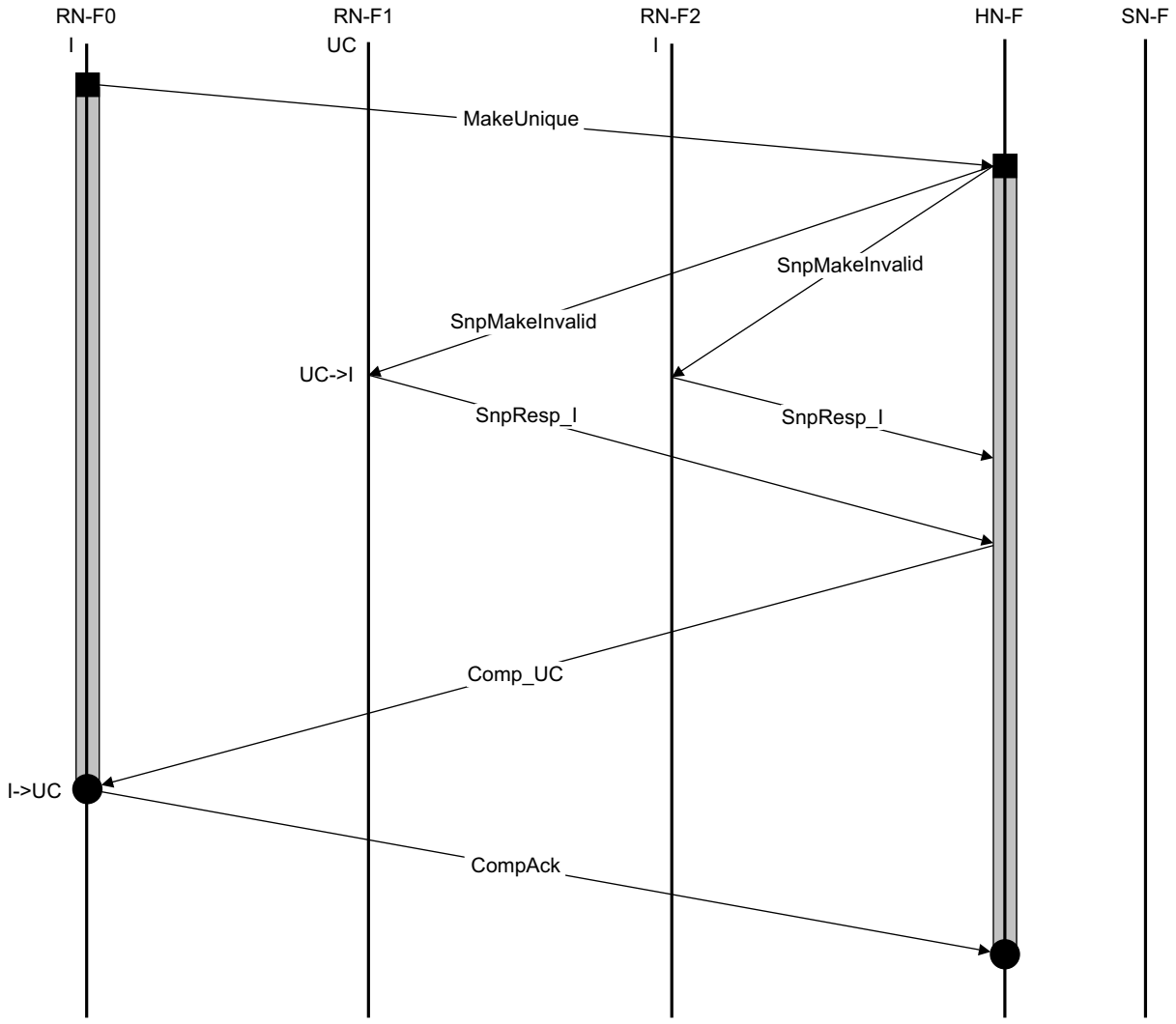


Figure 5-11 MakeUnique without memory update

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.2.2 Dataless transaction with memory update

An example of this type of flow is a CleanUnique transaction.

RN-F1 has the cache line in SD state and responds to the snoop with a snoop response with data and PD asserted.

HN-F waits for all snoop responses and then sends a Comp_UC response to the Requester.

HN-F sends a write request to update memory with the data received from RN-F1.

Figure 5-12 shows the transaction flow.

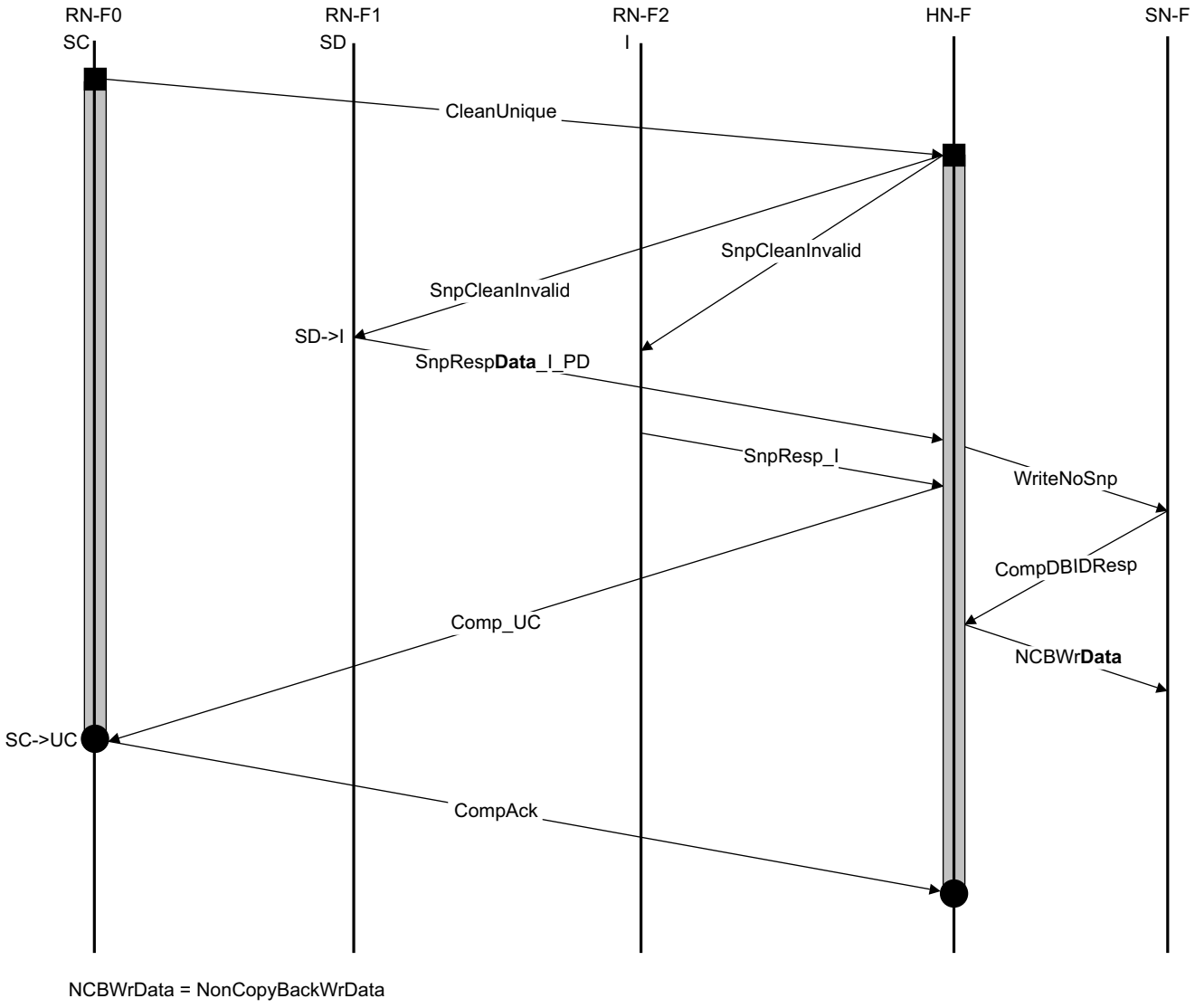


Figure 5-12 CleanUnique with memory update

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.2.3 Evict transaction

Figure 5-13 shows the Evict transaction flow.

RN-F0 moves the cache line to I state and issues an Evict transaction.

HN-F receives and allocates the request.

Note

The Evict request is a hint. A Comp response can be given by HN-F without updating the Snoop Filter or Snoop Directory.

HN-F returns the Comp response and deallocates the request.

RN-F0 deallocates the request.

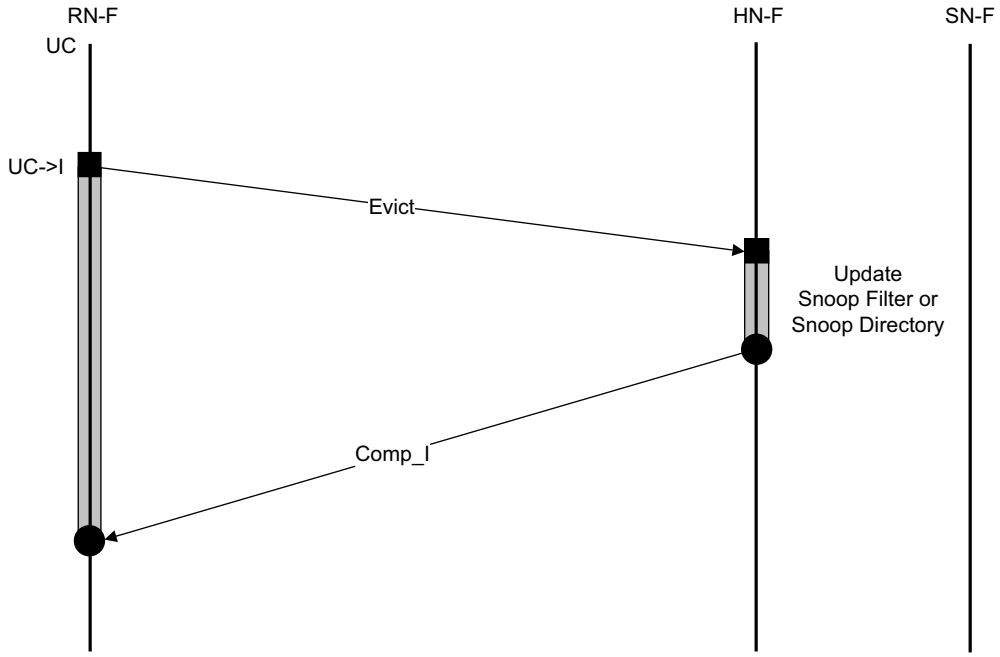


Figure 5-13 Evict transaction flow

Note

The cache state at the Requester must change to Invalid before the Evict message is sent.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.3 Write transaction flows

This section gives examples of the interconnect protocol flow for Write transactions.

5.3.1 Write transaction with no snoop and separate responses

An example of this type of flow is a WriteNoSnp transaction. The steps in the WriteNoSnp transaction flow are:

1. RN-F0 issues a WriteNoSnp transaction.
2. HN-F receives and allocates the request.
3. HN-F sends DBIDResp without Comp.
4. RN-F0 responds with data.
5. HN-F sends a Comp after it receives CompDBIDResp from 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 it receives the WriteNoSnp request from RN-F0.

6. RN-F0 waits for Comp from HN-F and deallocates its request.

Figure 5-14 shows the flow, the copy of data being transferred is marked in bold.

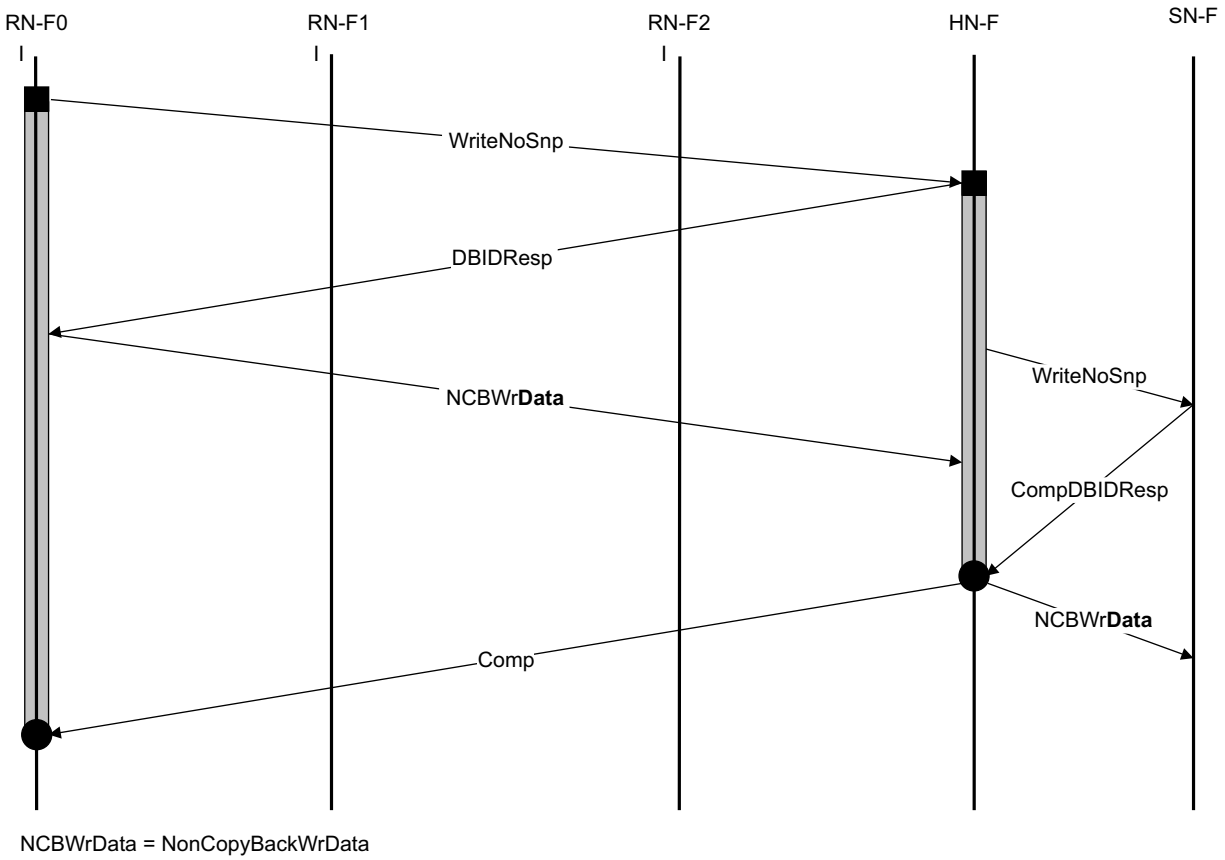


Figure 5-14 WriteNoSnp with separate responses from HN to RN

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.3.2 Write transaction with snoop and separate responses

An example of this type of flow is a WriteUniquePtl transaction.

The Comp_I response from HN-F must be sent when the coherency activity is complete at HN-F.

Figure 5-15 shows the transaction flow, the copy of data being transferred is marked in bold.

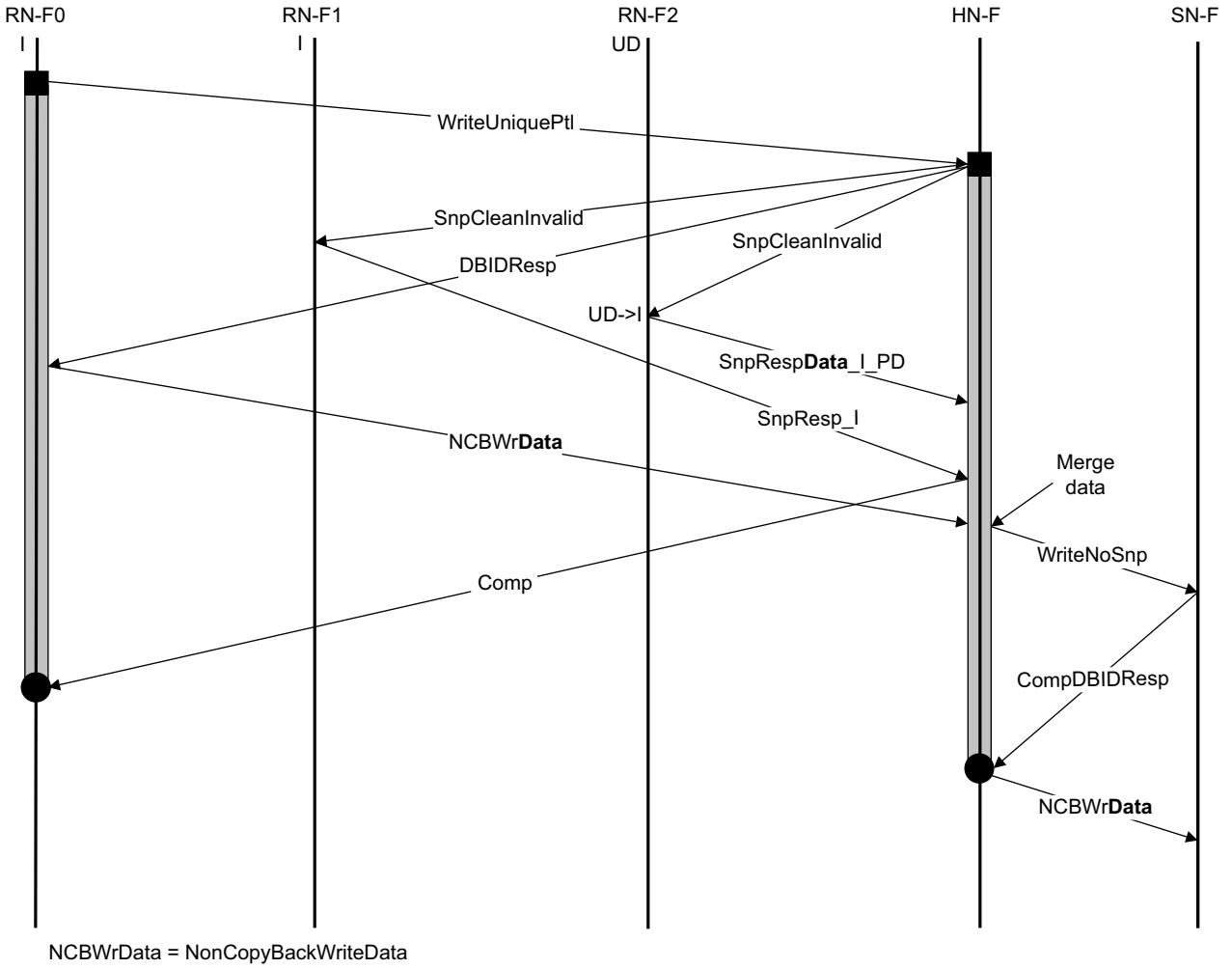


Figure 5-15 WriteUniquePtl with snoop

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.3.3 CopyBack write transaction to memory

An example of this type of flow is a WriteBackFull transaction.

The data received from RN-F0 is written to SN-F by HN-F using a WriteNoSnp transaction.

Figure 5-16 shows the transaction flow, the copy of data being transferred is marked in bold.

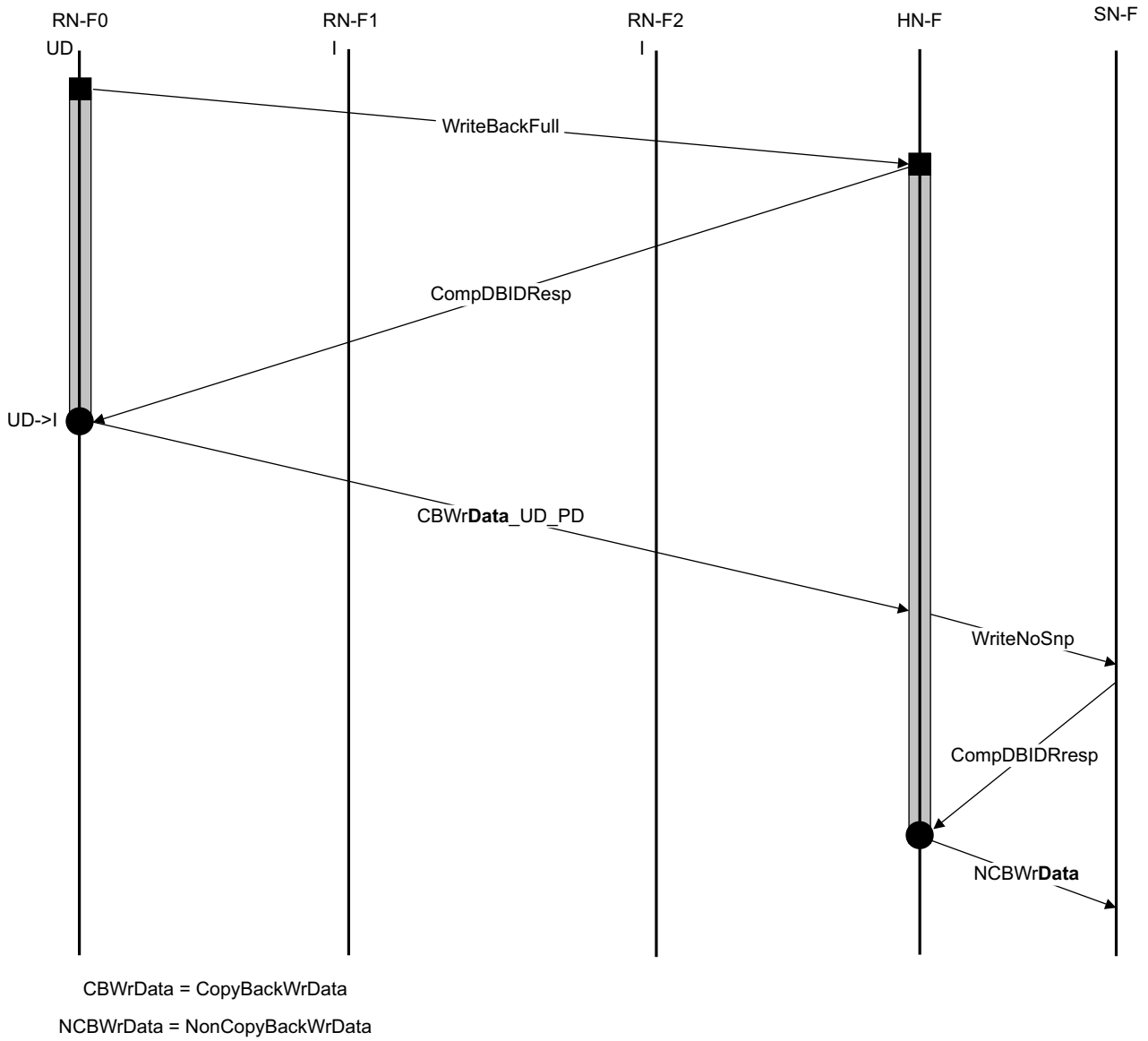


Figure 5-16 WriteBackFull returning Data Buffer Identifier

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.4 Atomic transaction flows

This section shows flows for different Atomic transaction types. It contains the following sub-sections:

- *Atomic transactions with data return*.
- *Atomic transaction without data return* on page 5-218.
- *Atomic operation executed at the SN* on page 5-220.

5.4.1 Atomic transactions with data return

This flow is applicable to:

- AtomicLoad.
- AtomicCompare.
- AtomicSwap.

Atomic transaction with snoops and data return

Figure 5-17 on page 5-216 shows the atomic operation executed at HN-F.

The steps in this transaction flow 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 to send a speculative ReadNoSnp to SN-F.
3. RN-F2 has the cache line in UD state and responds by sending data and invaliding its own cached copy.
   - The response is SnpRespData_I_PD.
   - This data is marked as (Old_Data) in Figure 5-17 on page 5-216 to distinguish it from both 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 old 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 NonCopyBackWrData_I response from the Requester.
   - This data is marked as (New_Data) in Figure 5-17 on page 5-216 to distinguish it from the data sent by RN-F2 in response to the Snoop request from HN-F,
6. Once HN-F receives the NonCopyBackWrData_I response from the Requester, and the Snoop response with data from RN-F2, it executes the atomic operation.
7. The resulting value after atomic operation execution is written to SN-F.
8. In this example, the read data received due to the speculative read is discarded by HN-F.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

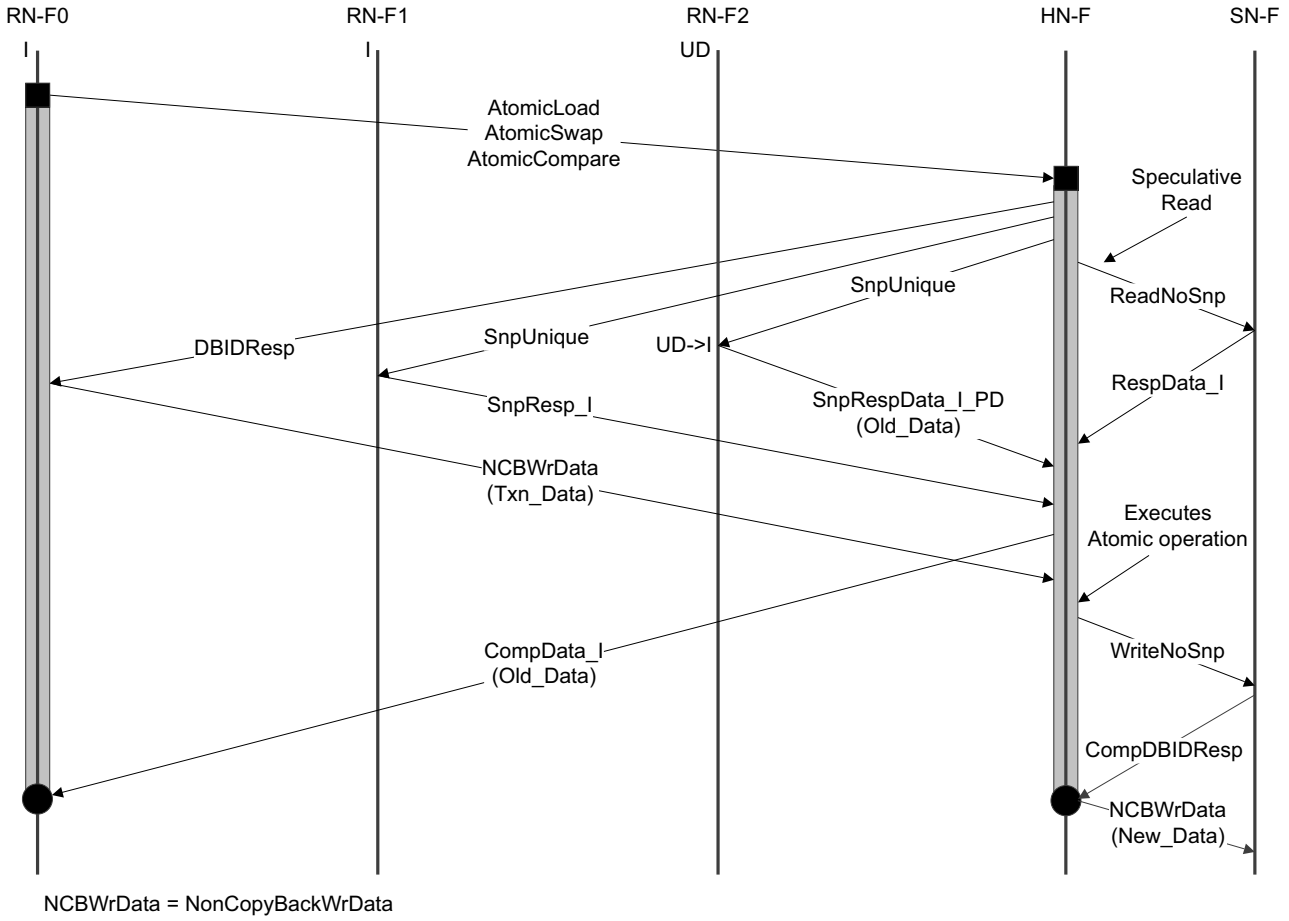


Figure 5-17 AtomicLoad, AtomicSwap, or AtomicCompare executed at HN-F

Note

In Figure 5-17, the CompData_I response from HN-F can be sent as soon as all Snoop responses are received.

Alternatively, to aid error reporting, CompData_I can be delayed until NCBWrData is received from the Requester and the atomic operation is executed.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Atomic transaction without snoops and with data return

Figure 5-18 shows the atomic operation executed at HN.

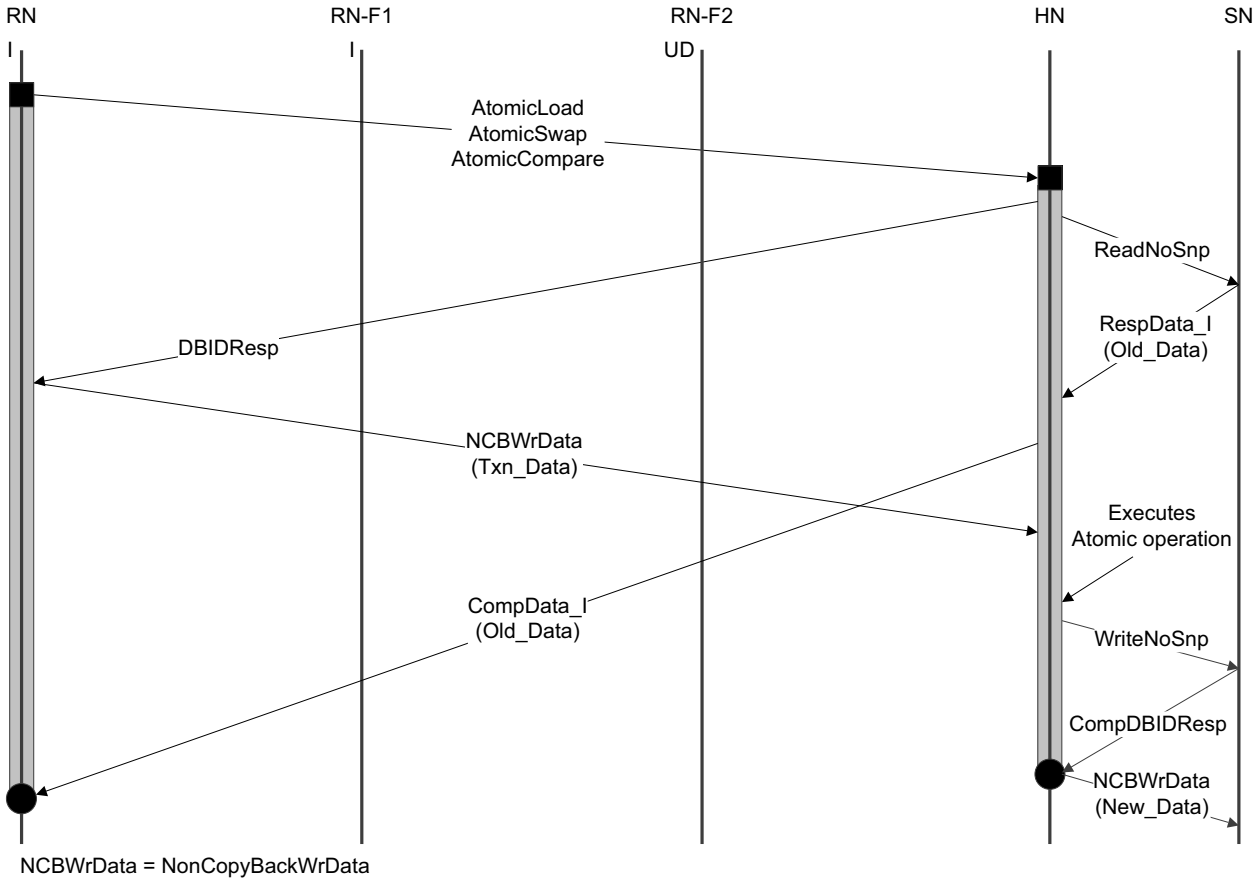


Figure 5-18 AtomicLoad, AtomicSwap, or AtomicCompare executed at HN

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.4.2 Atomic transaction without data return

This flow is applicable to AtomicStore transactions.

Atomic transaction with snoops and without data return

Figure 5-19 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.

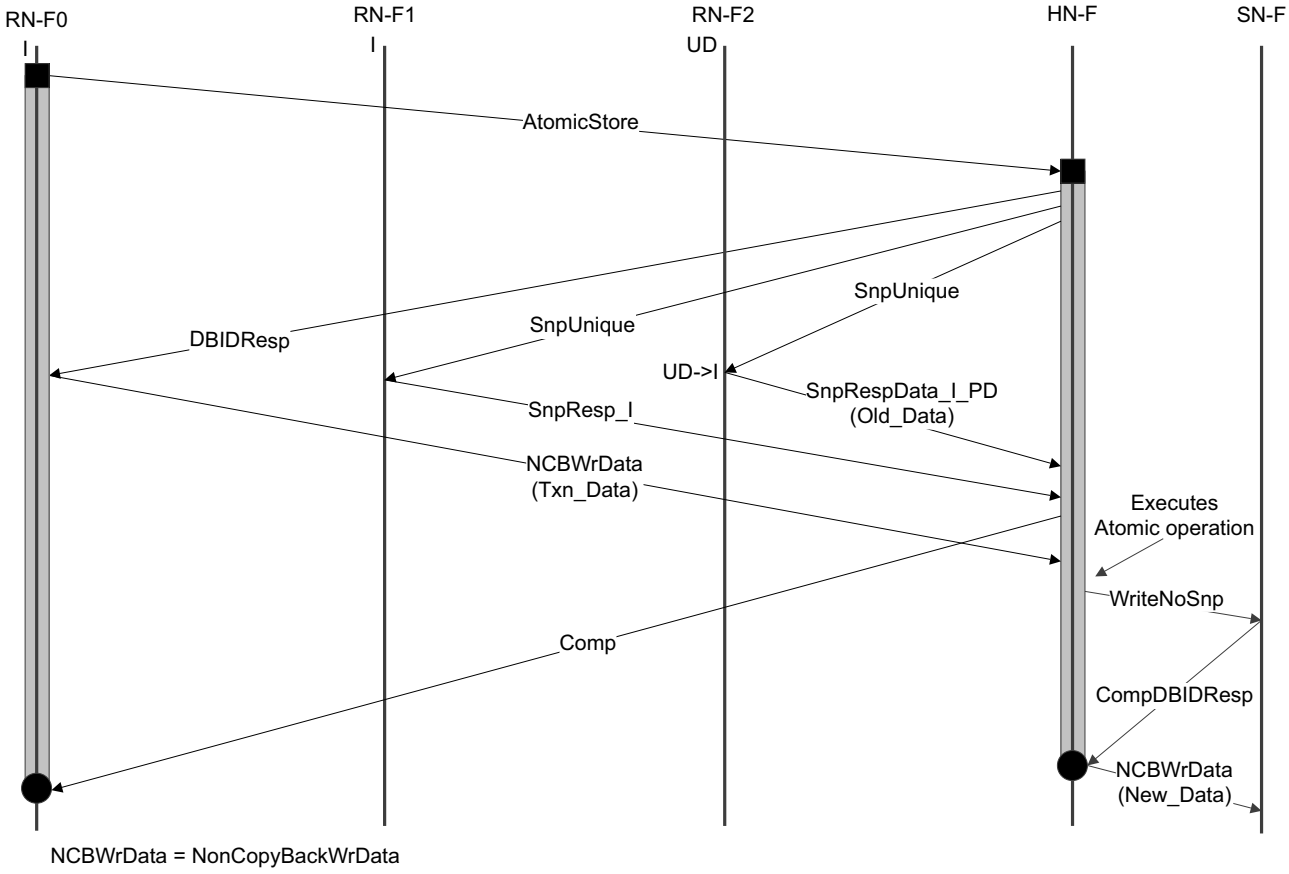


Figure 5-19 AtomicStore executed at HN-F

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Atomic transaction without snoops and without data return

Figure 5-20 shows the atomic operation executed at HN. The flow is similar to the Atomic transaction without snoop and with data return except that the Comp response to RN does not include data.

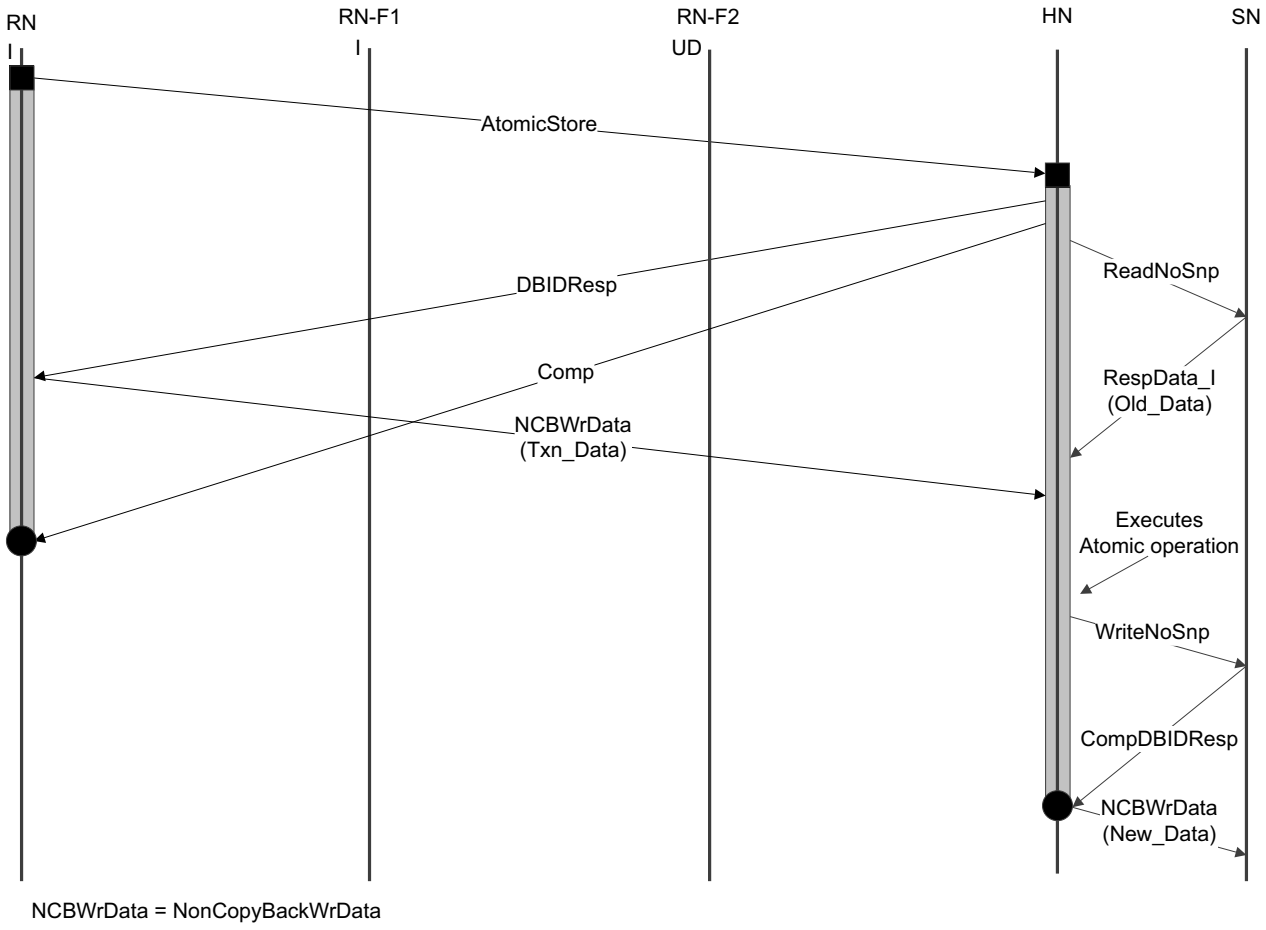


Figure 5-20 AtomicStore executed at HN

Note

- In Figure 5-20, the read from SN is required to obtain the Old_Data and is not speculative.
- The Comp response from HN can be combined with the DBIDResp response.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.4.3 Atomic operation executed at the SN

Figure 5-21 on page 5-221 shows an example Atomic transaction flow where the SN-F is executing the atomic operation.

The steps in this transaction flow 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 invaliding its own cached copy.
   - The response is SnpRespData_I_PD.
   - This data is marked as (Old_Data) in Figure 5-21 on page 5-221 to distinguish it from both 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 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 NonCopyBackWrData 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 as soon as all the Snoop responses are received.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

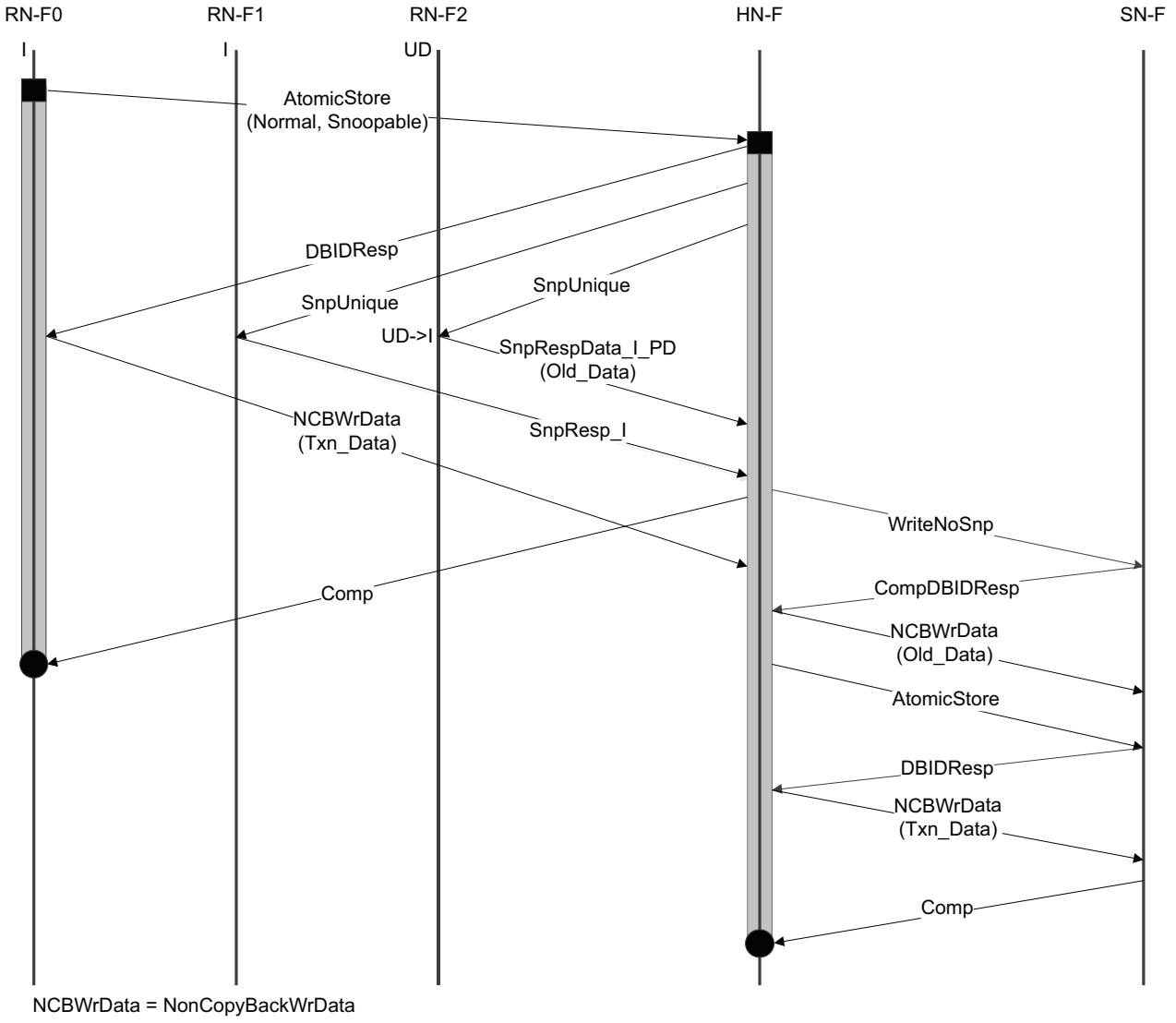


Figure 5-21 AtomicStore executed at SN-F

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.5 Stash transaction flows

This section shows example interconnect protocol flows for the two Stash transaction types:

- *Write with Stash hint*.
- *Independent Stash request* on page 5-223.

5.5.1 Write with Stash hint

Figure 5-22 on page 5-223 shows an example WriteUniqueStash with Data Pull transaction flow.

1. RN sends a WriteUniqueFullStash request to HN-F with the Stash target identified as RN-F1. Typically, the reqesting RN 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 RN.
5. RN-F1 sends CompAck to HN-F to complete the Read transaction.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

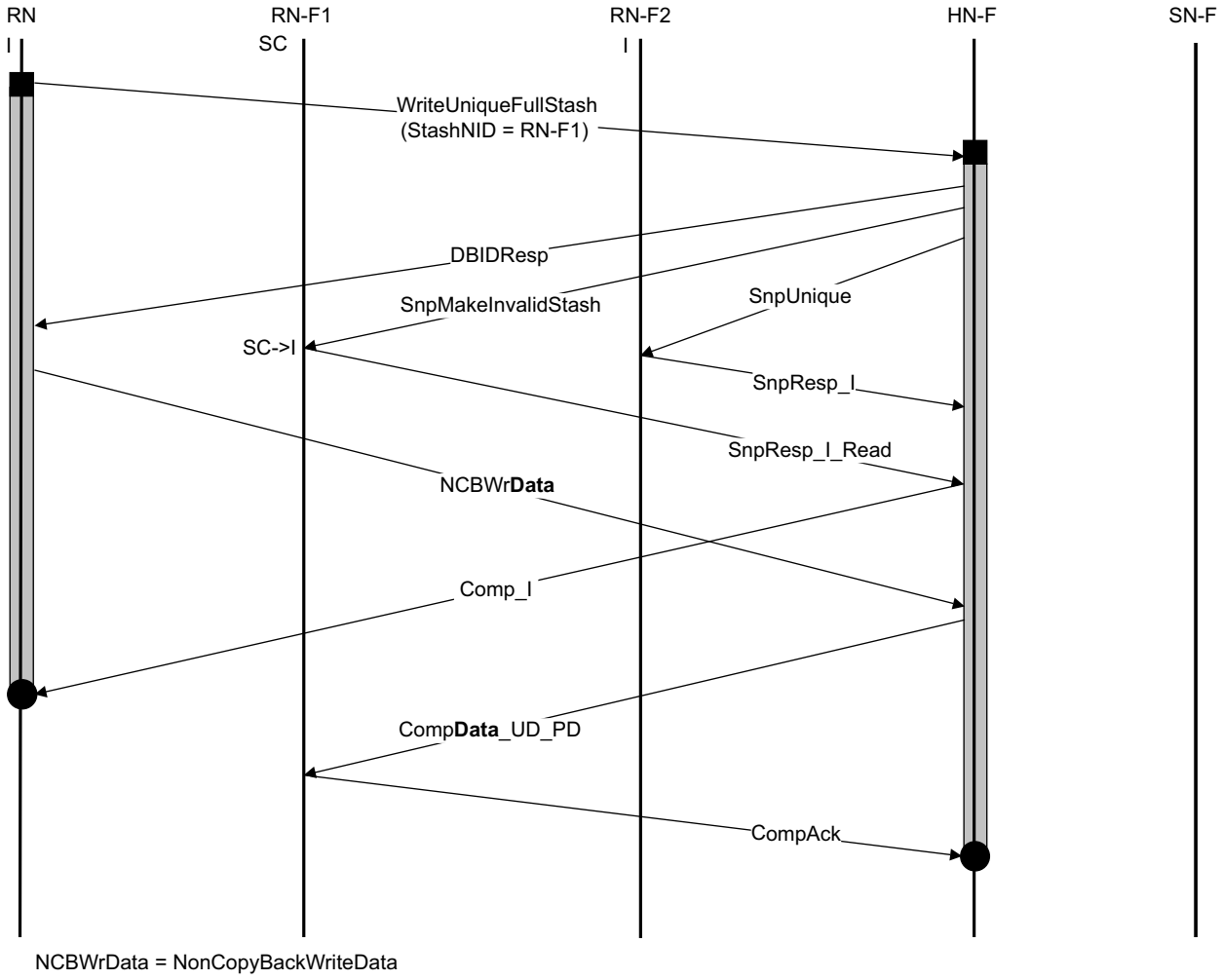


Figure 5-22 WriteUniqueStash with Data Pull

5.5.2 Independent Stash request

Figure 5-23 on page 5-224 shows an example StashOnce with Data Pull transaction flow.

1. RN sends a StashOnceShared request to HN-F with the Stash target identified as RN-F1.
2. HN-F can immediately send Comp to RN to acknowledge the Stash request.
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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

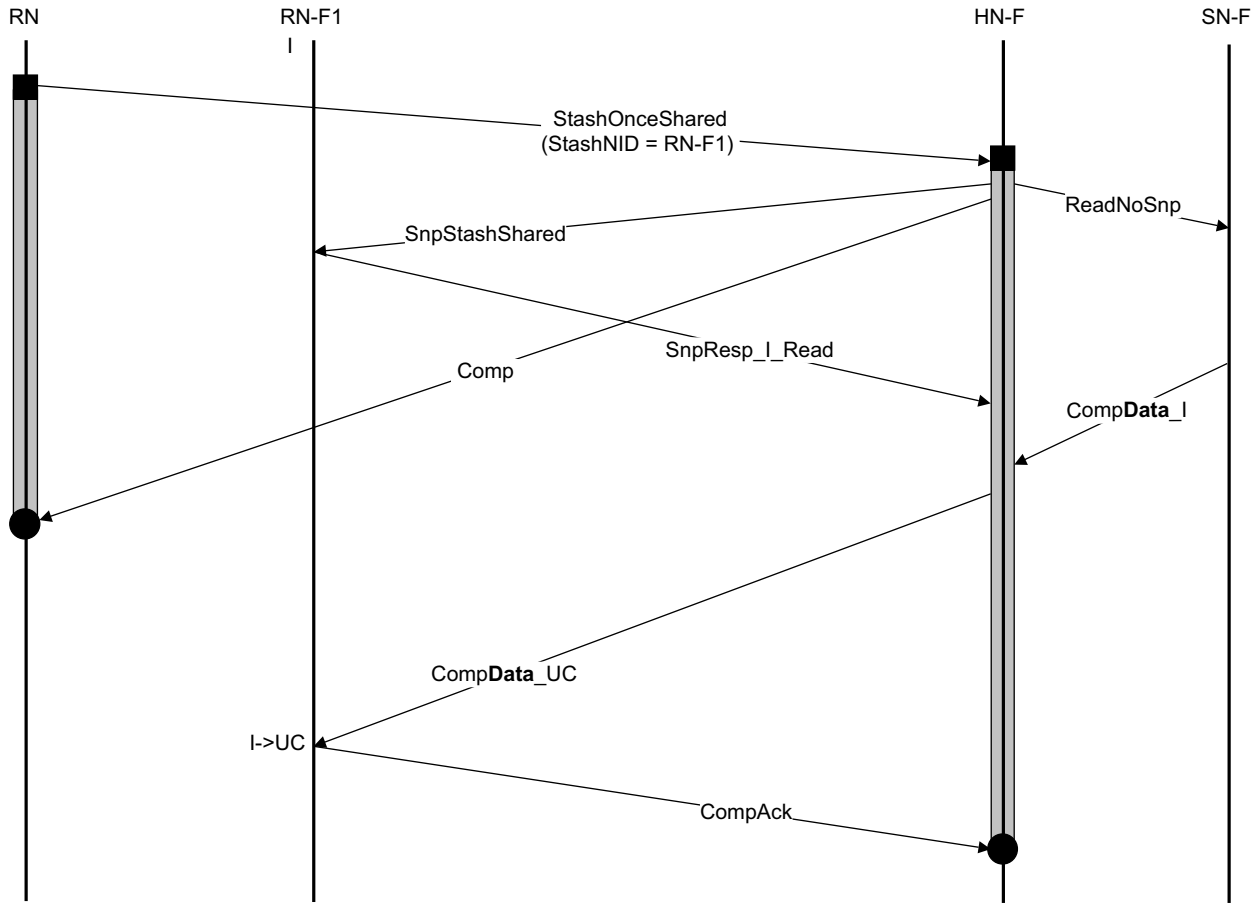


Figure 5-23 StashOnceShared with Data Pull

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.6 Hazard handling examples

This section shows how CopyBack-Snoop request hazard conditions are handled at the Requester and how various request to request, and request to snoop request hazard conditions are handled at the HN-F. It contains the following subsections:

- *CopyBack-Snoop hazard at RN-F*.
- *Request hazard at HN-F* on page 5-228.
- *Read - CopyBack or Dataless - CopyBack hazard at HN-F* on page 5-230.
- *Request-CompAck to HN-F race hazard* on page 5-231.

5.6.1 CopyBack-Snoop hazard at RN-F

Figure 5-24 on page 5-226 shows a Snoop request to an RN-F hazarding a pending CopyBack request at time C. 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 CopyBackWrData_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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

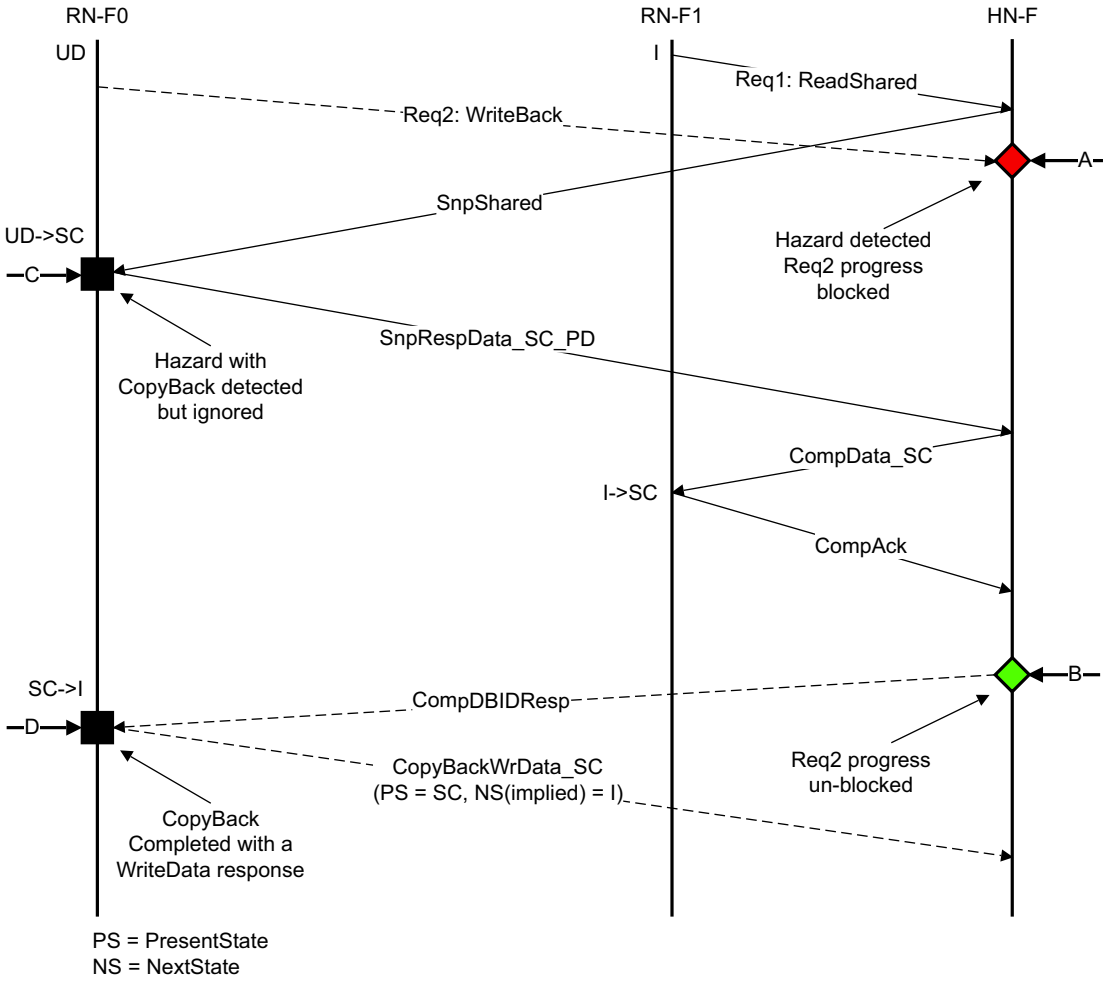


Figure 5-24 CopyBack-Snoop hazard at RN-F example

Note

- The response to a snoop request that hazards with an outstanding Evict must be SnpResp_I.
- During the period between receiving a snoop request and sending a snoop response, including data if applicable, while a CopyBack request to the same address is pending, the only response that can be received for the CopyBack request is a RetryAck.

Figure 5-25 on page 5-227 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

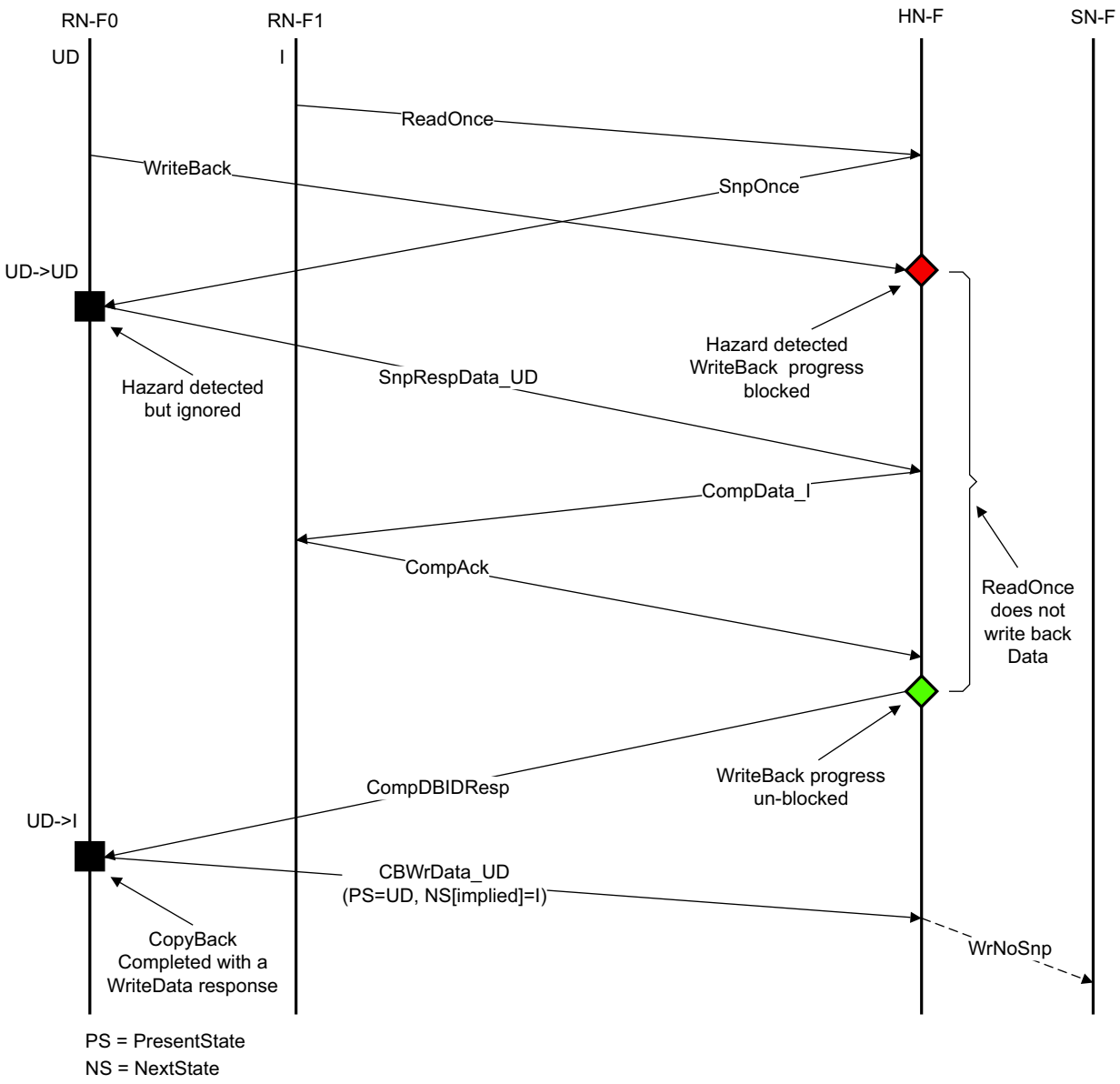


Figure 5-25 CopyBack-Snoop hazard with no cache state change example

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.6.2 Request hazard at HN-F

If more than one request to the same cache line is ready to be processed at the HN-F, then the HN-F can select the next request in any order, except for when the two requests have an ordering requirement and are from the same source, then the order of processing must match the order of arrival.

Figure 5-26 on page 5-229 shows an example where a ReadShared and a ReadUnique, for the same cache line, arrive at the HN-F at approximately the same time. The steps required to resolve this hazard 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.

With the exception of ReadNoSnp, the flows will be similar if the two transactions, that Figure 5-26 on page 5-229 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

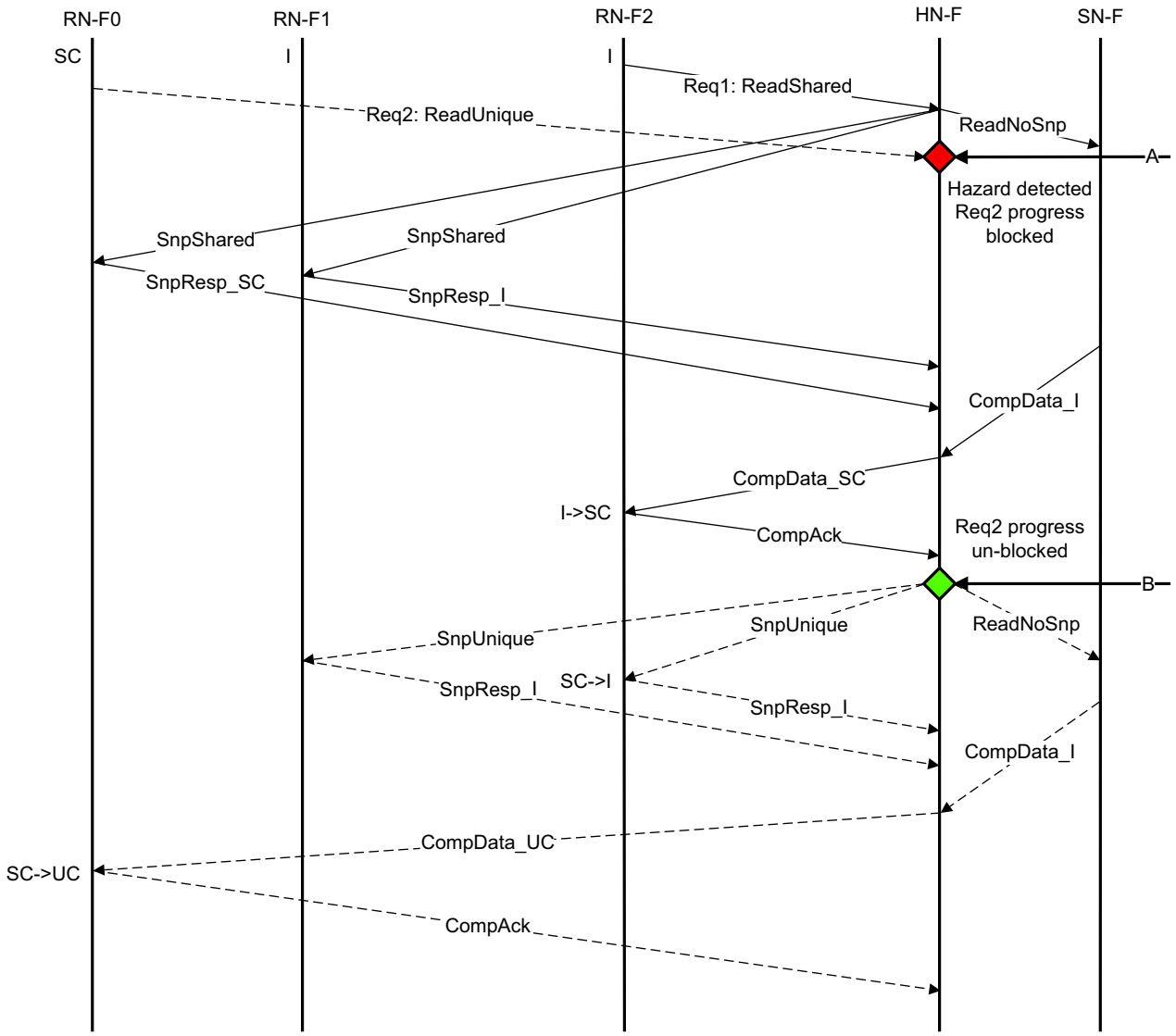


Figure 5-26 Read-Read request hazard example

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

5.6.3 Read - CopyBack or Dataless - CopyBack hazard at HN-F

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 *Request hazard at HN-F* on page 5-228. See also *CopyBack-Snoop hazard at RN-F* on page 5-225.

Figure 5-27 on page 5-231 shows the case where a ReadShared and a WriteBack, for the same cache line, arrive at the HN-F at approximately the same time. The steps required to resolve this hazard 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 needs to 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, then the ReadShared request will be 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 required.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

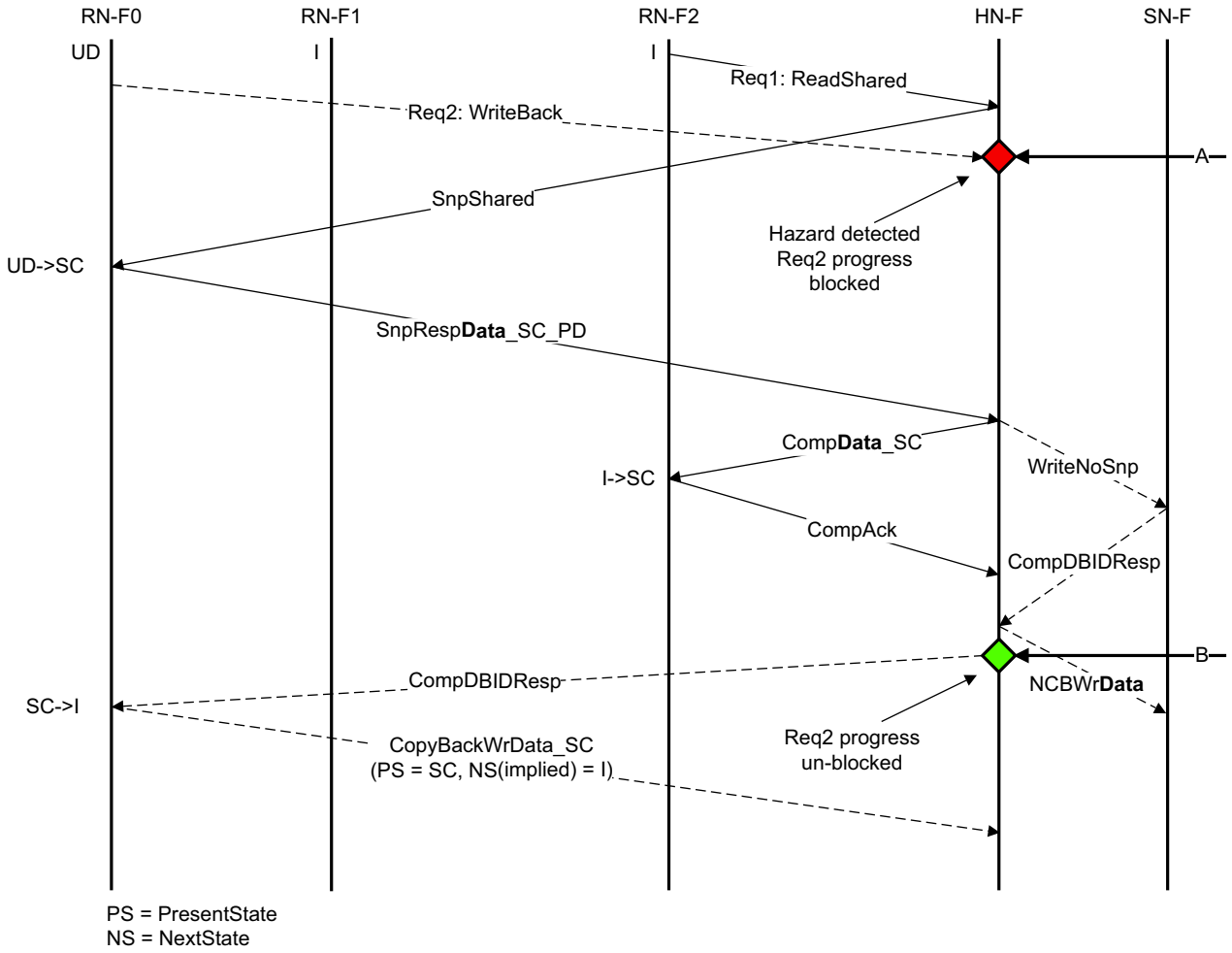


Figure 5-27 Read - CopyBack or Dataless - Copyback hazard example

5.6.4 Request-CompAck to HN-F race hazard

After completion, a request might 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, upon arrival at HN-F, the CompAck response deallocates the previous request from the HN-F and unblocks the processing of the new request.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 6
Exclusive Accesses

This chapter describes the mechanisms that the architecture includes to support Exclusive accesses. It contains the following sections:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

6.1 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.

Two different forms of Exclusive access are supported:

- Exclusive accesses to a Snoopable memory location.
- Exclusive accesses to a Non-snoopable memory location.

If the location is updated since the Exclusive Load, by a different LP, then 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.

Note

- The term Exclusive Load is used to describe the action of an LP executing an appropriate program instruction such as LDREX. This action requires:
  - Obtaining the data from the location to which it wants to perform an exclusive sequence.
  - Indicating that it is starting an exclusive sequence.
- The term Exclusive Load transaction is used to describe 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.
- The term Exclusive Store is used to describe the action of an LP executing an appropriate program instruction such as STREX. This action requires:
  - Determining if the exclusive sequence has passed or failed.
  - If appropriate, updating the data at the location.

  An Exclusive Store 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.
- The term Exclusive Store transaction is used to describe a transaction issued on the interface that might 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

6.2 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, is dependent on the memory attributes of the address.

6.2.1 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, which is indicated by an invalidating snoop request to the same address.
- There is a store to the location by the same LP. Resetting of the monitor if the store from the same LP is non-exclusive 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 that an Exclusive Store transaction from an LP is only successful if that LP could not have received a snoop transaction, relating to an Exclusive Store to the same address from another LP, after it issued its own Exclusive Store transaction.

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 it then performs an Exclusive Store transaction before a successful Exclusive Store transaction from another LP is scheduled, then the Exclusive Store transaction must be successful.

The monitor must support the parallel monitoring of all exclusive-capable LPs 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, it has not been reset by an Exclusive Store transaction from another LP, then the Exclusive Store transaction is successful and is permitted to proceed. In such a case, registered attempts of all other LPs must be reset. This specification recommends, but does not require, 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, it has been reset by an Exclusive Store from another LP, then the Exclusive Store transaction is failed and is not permitted to proceed. The monitor must register that the LP is attempting an exclusive sequence.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 will all be 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 this specification does not require it. At that point, all other LPs must then 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.

6.2.2 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 another LP's Exclusive Store transaction 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. It is reset by a successful Exclusive Store transaction from another LP to a matching address.

A monitor that includes 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, additional address monitors can be used on a first-come first-served basis, or by allocation to particular LPs. Alternatively, a more complex algorithm might 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 a large number of Exclusive access transactions. This specification recommends that Secure Exclusive accesses are permitted to make forward progress independently of the progress of Non-secure accesses.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

6.2.3 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 so it can observe all transactions to the monitored location.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

6.3 Exclusive transactions

The following transaction types support Exclusive accesses through an Excl bit:

- Exclusive Load transaction to a Snoopable location:
  - ReadClean.
  - ReadNotSharedDirty.
  - ReadShared.
- Exclusive Store transaction to a Snoopable location:
  - CleanUnique.
- 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 *Logical Processor Identifier* on page 2-95.

Exclusive reads must not use Direct data transfer flow.

6.3.1 Responses to exclusive requests

Transaction responses to exclusive requests are similar to the normal responses to reads and writes except that:

- Exclusive reads must not use separate Comp and Data response.
- Exclusive reads that do not fail must not use either DMT nor DCT.

However, the response must also indicate if the exclusive request has passed or failed. The RespErr field in the response is used for this purpose. See *RespErr* on page 12-331. 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.

Whether or not an Exclusive Store transaction to a location that does not support Exclusive accesses will update that location is IMPLEMENTATION DEFINED.

This specification recommends that an Exclusive Store transaction is not performed to a location that does not support Exclusive accesses.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 6-1 shows the Snoopable attributes of the request, the relevant monitor type and possible reasons for fail conditions and response requirements.

Table 6-1 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
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	

6.3.2 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.
- This specification recommends that progress on Secure Exclusive requests is independent of progress on Non-secure Exclusive requests.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

6.3.3 Exclusive accesses to Snoopable locations

This section describes the behavior of an LP when performing Exclusive accesses to a Snoopable address location.

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 might already hold the cache line in its local cache:

- If the LP holds the cache line in a Unique state, then it is permitted, but not recommended by this specification, that it performs an Exclusive Load transaction.
- If the LP holds the cache line in a Shared state, then it is permitted, but not required by this specification, that it performs an Exclusive Load transaction.
- If the LP does not hold a copy of the cache line, this specification recommends that the LP uses an Exclusive Load transaction to obtain the cache line, but is permitted to use ReadClean or ReadShared or ReadNotSharedDirty without the Excl attribute asserted.

Snoopable Exclusive Load to Snoopable Exclusive Store

After the execution of an Exclusive Load an LP will typically calculate a new value to store to the location before it attempts 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 might have updated the location.

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 then the Exclusive Store has passed and it can update the location without issuing a transaction.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- If the cache line is held in a Shared state and the LP exclusive monitor is set then the LP must issue an Exclusive Store transaction. A CleanUnique 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 transaction is in progress.

  The transaction will receive a Normal Okay or an Exclusive Okay response.

  If the transaction receives an Exclusive Okay response, then 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 then the Exclusive Store has passed and the update is performed.

  — If the LP exclusive monitor is not set, it indicates that an update to the cache line has occurred between the point that the Exclusive Store transaction was issued and the point that it completed. 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, then the Exclusive Store must fail and the exclusive sequence must be restarted.

  If the Exclusive Store transaction receives a Normal Okay response then 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 LPs 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, then 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, then the LP can repeat the Exclusive Store transaction.

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, that is, 1, 2, 4, 8, 16, 32, 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, 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 SN does not support Exclusive accesses, as indicated by an Exclusive Fail on the Exclusive ReadNoSnp, then the write will update the location if the write is given an Exclusive Fail response.

If the SN does support Exclusive accesses, as indicated by an Exclusive Pass on the Exclusive ReadNoSnp, then the write will not update the location if the write is given an Exclusive Fail response.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 7
Cache Stashing

This chapter describes the cache stashing mechanism whereby data that is written from an RN can be installed in a peer cache. It contains the following sections:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

7.1 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, therefore improving the system performance.

Cache stashing is permitted to Snoopable memory only.

This specification supports two main forms of cache stashing transaction:

Write with stash hint

WriteUniqueStash. 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 will affect the Snoop transactions that are used. See *Write with Stash hint* on page 7-246.

Independent stash request

StashOnce. 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 StashOnceShared transaction respectively, which corresponds to whether the next expected use of the cache line is for storing or for reading respectively. See *Independent Stash request* on page 7-247.

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 a logical processor cache within the peer node, if the peer node has multiple logical processors. The logical processor is identified by the LPID in the target cache field. See *Stash target identifiers* on page 7-249.

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 *Stash target not specified* on page 7-249.

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.

7.1.1 Snoop requests and Data Pull

The following Snoop requests are used to notify a peer cache that it is the target of a Stash request:

- SnpUniqueStash.
- SnpMakeInvalidStash.
- SnpStashUnique.
- SnpStashShared.

Table 7-1 shows the Snoop requests associated with each of the Stash requests.

Table 7-1 Stash request and the corresponding Snoop request

Stash request	Snoop request
WriteUniquePtlStash	SnpUniqueStash
WriteUniqueFullStash	SnpMakeInvalidStash
StashOnceUnique	SnpStashUnique
StashOnceShared	SnpStashShared

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

A Snoopee that receives a Stash type 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 Data Pull. Data Pull can only be used if the DoNotDataPull field in the Snoop request is deasserted. Table 7-2 shows the type of Read request that is implied by a Data Pull in the response to each Stash type Snoop request.

Table 7-2 Read request and corresponding Data Pull response

Snoop request	Implied Read request
SnpUniqueStash	ReadUnique
SnpMakeInvalidStash	ReadUnique
SnpStashUnique	ReadUnique
SnpStashShared	ReadNotSharedDirty

- Provides a Snoop response without the use of Data Pull but sends a separate Read request to obtain a copy of the cache line. If this approach is used, there is no mechanism to associate the Read request with the stash operation. It cannot be determined if the Read request is directly as a result of the stash operation or if it is unrelated.
- Provides a Snoop response with neither a Data Pull nor a follow up request, ignoring the cache stash hint.

The value of the DataPull field in the SnpResp and SnpRespData responses indicates if Data Pull is requested. See *DataPull* on page 12-326 for legal values for DataPull.

The use of Data Pull to complete a Snoop request with Stash is optional and can be controlled by both sides of the interface:

- If Home is not able to support the Data Pull transaction flow then it must assert the DoNotDataPull field within the Snoop request.
- If the Snoopee is not able to support the Data Pull transaction flow then it is permitted to either ignore the stash operation or to issue an independent Read request.

Note

Using the Data Pull transaction flow removes the need to send an additional Request packet after the Snoop response and it can also improve the Home efficiency by ensuring a closer coupling between the original stash operation and the movement of the data to the targeted cache.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

7.2 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.
- 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 *Stash target not specified* on page 7-249.
- Permitted to use DMT to get data from SN-F to the Stash target in response to a Data Pull 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 Data Pull request.

The Stash target responsibilities:

- This specification recommends, but does not require, that the Read request is combined with the Snoop response if the DoNoDataPull bit in the Snoop request is not set.
- 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:
  - DoNotDataPull bit is set.
  - Snoop has an address hazard with an outstanding request.
- When requesting Data Pull:
  - The Stash target must guarantee the Read data is accepted without any structural or protocol dependencies that might 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.
- Permitted not to request Data Pull but to send a separate Read request. In this case this specification recommends, but does not require, that the Stash target uses ReadUnique for the read.
- Permitted to ignore the Stash hint and handle the snoop as SnpUnique.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

7.3 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 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 either StashOnceUnique or StashOnceShared to Home, based on whether the stashed cache line is to be modified.
- The StashOnce request provides a Stash target when the data is to be stashed in a peer cache.
- The StashOnce request does not provide a Stash target when the data is to be allocated to the next level cache.

Home Node responsibilities:

- Permitted to send a RetryAck response to a StashOnce request and follow the Retry transaction flow.
- Send a SnpStashUnique to the target RN-F for StashOnceUnique.
- Send a SnpStashShared to the target RN-F for StashOnceShared.
- Permitted to not send a Snoop request in response to a StashOnce request.
- Must send a Comp response, even if it abandons the Stash request.
- Fetches the addressed cache line from memory into the shared system cache when a StashOnce request without a Stash target is received.
- Permitted to send Comp after receiving the StashOnce request, and before sending any SnpStash or receiving the Snoop response.
- Send Comp_[X], where [X] is not I state, if the request hit the cache line at Home.

  The [X] state is permitted only when it matches the cache state of the cache line at Home.
- 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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.
- This specification recommends, but does not require, that the Stash target includes a Data Pull request in the Snoop response if the DoNoDataPull bit in the Snoop request is not set.
- Must not request Data Pull if:
  - DoNotDataPull bit is set.
  - 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.
- When requesting Data Pull:
  - The Stash target must guarantee the Read data is accepted without any structural or protocol dependencies that might 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 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 or an independent CleanUnique 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 till it completes the local cache lookup 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 StashOnceShared or StashOnceUnique transactions, 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 StashOnceUnique 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

7.4 Stash target identifiers

For all Stash requests, both options of specified and non-specified Stash target are supported.

7.4.1 Stash target specified

If the Stash target is available in the Stash request then Home sends the snoop with a stash hint to the specified target. The specified target can be an RN or a logical processor within an RN.

7.4.2 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 an RN, then Home can treat that RN as the Stash target.
- If the cache line is not cached in a Unique state then Home must only send SnpUnique as required, and must not send SnpUniqueStash to any RN.
- For WriteUniquePtlStash, if the cache line is not in any cache then this specification recommends Home to prefetch and allocate 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 then Home is permitted to allocate the cache line in the shared system cache.

The Home Node that receives a StashOnceUnique or StashOnceShared request without a Stash target does the following:

- If the cache line is not cached in any peer cache then this specification recommends that the cache line is allocated in the shared system cache.
- If the cache line is cached in a peer cache then it is IMPLEMENTATION DEFINED if a snoop is sent to transfer a copy of the cache line and allocate it in the shared system cache. For StashOnceUnique, it is also IMPLEMENTATION DEFINED if all cached copies are invalidated before allocating the cache line in the shared system cache.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

7.5 Stash messages

Stash messages are classified as:

- Write requests:
  - WriteUniqueFullStash.
  - WriteUniquePtlStash.

  See *Write transactions* on page 4-150.
- Dataless requests:
  - StashOnceUnique.
  - StashOnceShared.

  See *Dataless transactions* on page 4-147.
- Snoop requests:
  - SnpUniqueStash.
  - SnpMakeInvalidStash.
  - SnpStashUnique.
  - SnpStashShared.

  See *Snoop request types* on page 4-158.

7.5.1 Supporting REQ packet fields

The fields defined in the REQ packet to support Stash requests are:

- StashNID, StashLPID.
- StashNIDValid, StashLPIDValid.

See *Protocol flit fields* on page 12-314.

7.5.2 Supporting SNP packet fields

The fields defined in the SNP packet to support Stash requests are:

- StashLPID.
- StashLPIDValid.
- DoNotDataPull.

See *Protocol flit fields* on page 12-314.

7.5.3 Supporting RSP packet field

The field defined in the RSP packet to support Stash requests is:

- DataPull.

See *Protocol flit fields* on page 12-314.

7.5.4 Supporting DAT packet fields

The field defined in the DAT packet to support Stash requests is:

- DataPull.

See *Protocol flit fields* on page 12-314.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 8
DVM Operations

This chapter describes *Distributed Virtual Memory* (DVM) operations that the protocol uses to manage virtual memory. It contains the following sections:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

8.1 DVM transaction flow

All DVM transactions have similar requirements and are mapped to a single flow. The following sections show the Non-sync and Sync type DVM transaction requirements:

- *Non-sync type DVM transaction flow*.
- *Sync type DVM transaction flow* on page 8-254.
- *Flow control* on page 8-255.
- *DVMOp field value restrictions* on page 8-257.
- *Field value requirements* on page 8-260.

8.1.1 Non-sync type DVM transaction flow

Figure 8-1 shows the steps in a Non-sync type DVM transaction.

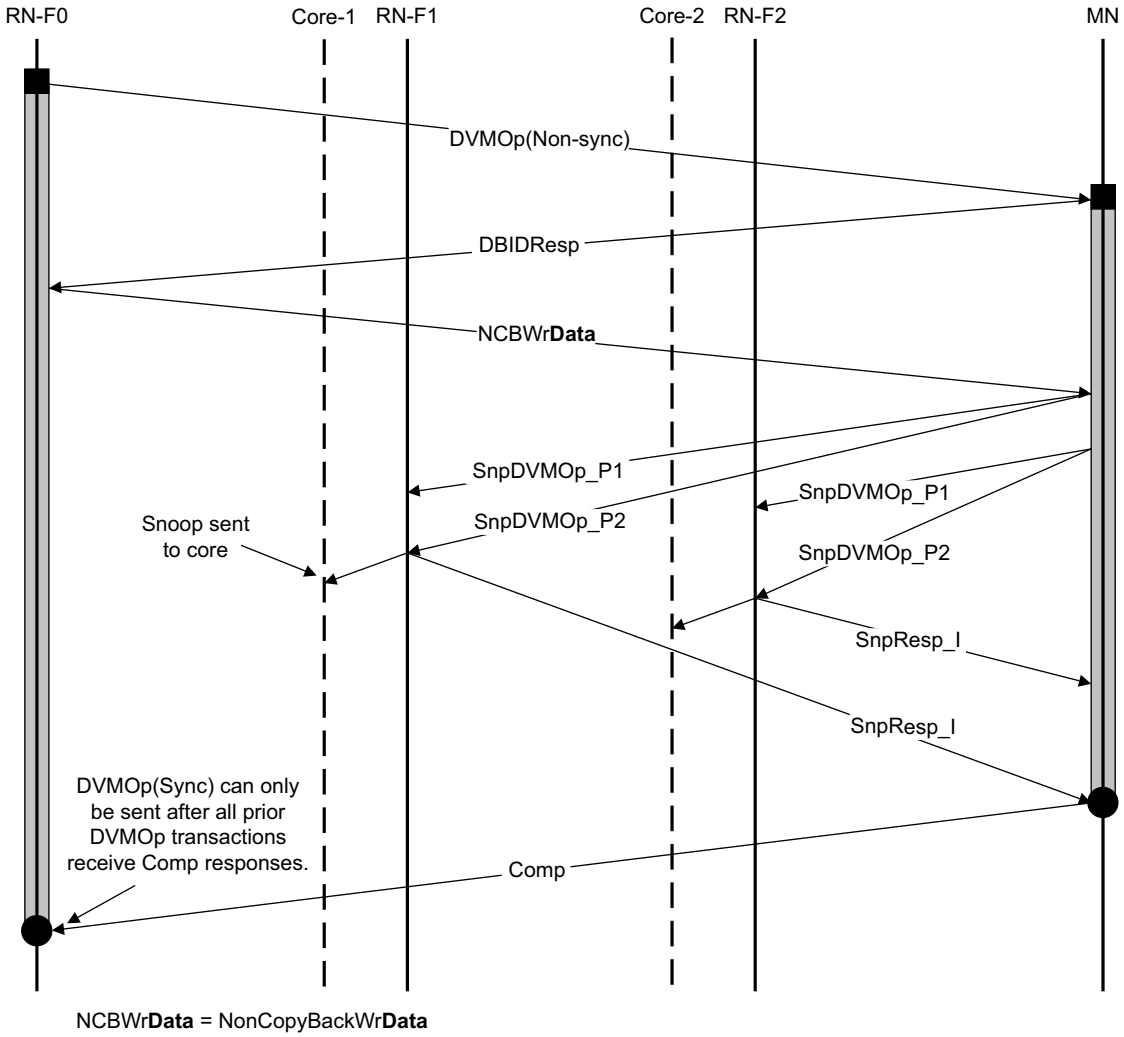


Figure 8-1 Non-sync type DVM transaction flow

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The required steps that Figure 8-1 on page 8-252 shows are:

1. RN-F0 sends a DVMOp(Non-sync) to the MN using the appropriate write semantics for the DVMOp type.
2. The MN 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 all the RN-F and RN-D nodes in the system. 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 respectively.

   Note

   - Both parts of the message must carry the same *Transaction ID* (TxnID).
   - RN must have resources available to accept the SnpDVMOp. See *Flow control* on page 8-255.

5. After completing the required actions, each recipient of the SnpDVMOp sends a single SnpResp response to the MN.

   Note

   Sending of a SnpResp implies that the target RN has forwarded the SnpDVMOp to the required RN structures and has freed up the resources needed to accept another DVM operation. It does not imply that the requested DVM operation has completed. See *Sync type DVM transaction flow* on page 8-254.

6. After receiving all the SnpResp responses, the MN sends a Comp response to the requesting node.

   Note

   DBIDResp and Comp responses cannot be opportunistically combined for DVMOps.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

8.1.2 Sync type DVM transaction flow

Figure 8-2 shows the flow in a Sync type DVM transaction.

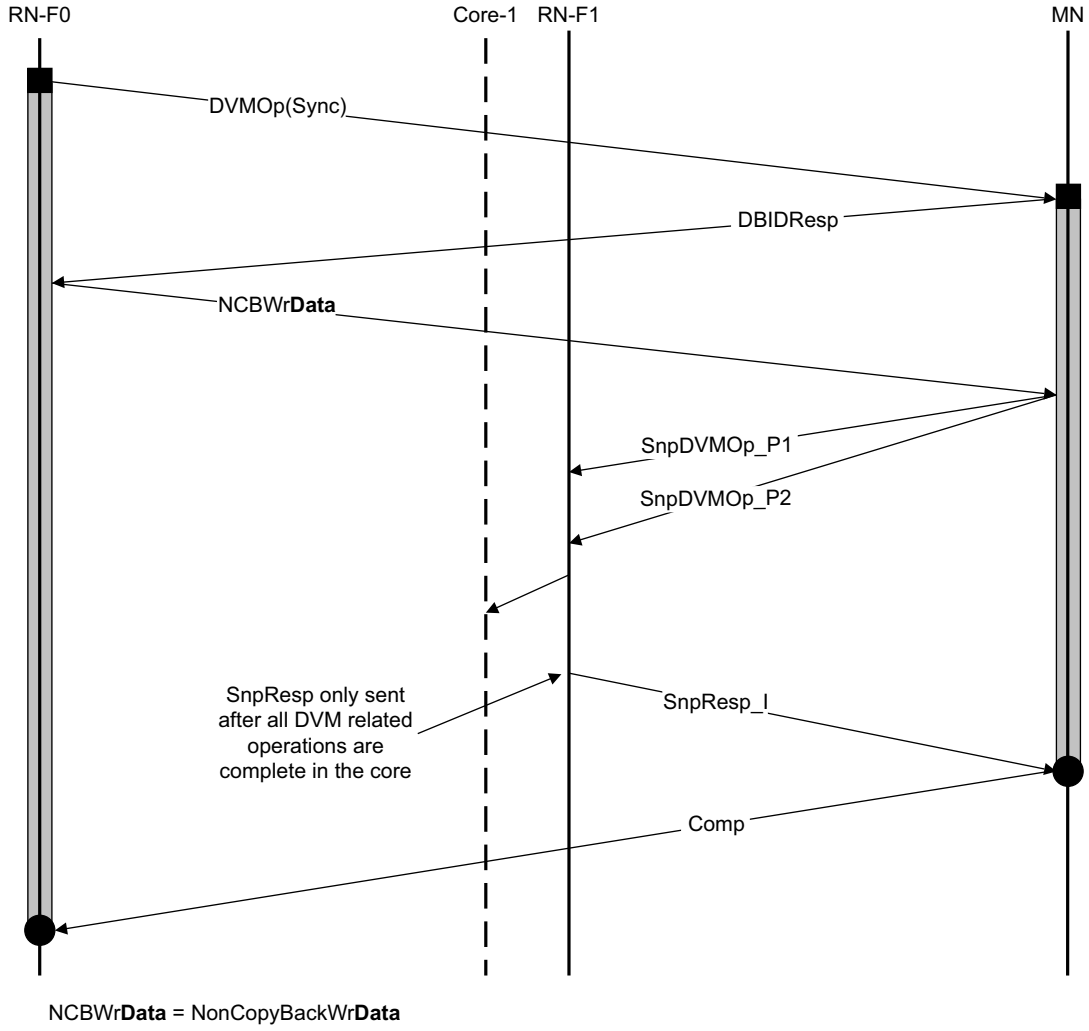


Figure 8-2 Sync type DVM transaction flow

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The required steps that Figure 8-2 on page 8-254 shows are:

1. RN-F0 sends a DVMOp(Sync) to the MN.

   Note

   All previous DVMOp requests whose completion needs to be guaranteed by the DVMOp(Sync) must have received a Comp response before the RN can send a DVMOp(Sync).

2. The MN 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 MN 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 respectively.
5. After completing the DVM Sync operation, RN-F1 sends a SnpResp response to the MN.

   Note

   Sending of a SnpResp implies that all DVM related operations have completed at the RN structures and the target RN has freed up the resources needed to accept another SnpDVMOp.

6. After receiving the SnpResp, the MN sends a Comp response to RN-F0.

8.1.3 Flow control

DVMOp request flow control:

- A DVMOp can receive a RetryAck response from an MN.
- A DVMOp that receives a RetryAck response must wait for a PCrdGrant response from the MN 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 RN can send the DVMOp(Sync).
- The interconnect must guarantee forward progress on DVMOp(Non-Sync), which implies that there should be at least one tracker entry in MN reserved for DVMOp(Non-Sync).
- It is permitted to overlap a DVMOp(Non-sync) and a DVMOp(Sync), from the same RN, if the DVMOp(Sync) is not required to guarantee completion of the DMVOp(Non-sync).

SnpDVMOp flow control:

- 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.
- Multiple SnpDVMOp(Non-sync) transactions can be outstanding from an MN.
- Only one SnpDVMOp(Sync) transaction can be outstanding from an MN to an RN.
- To prevent deadlocks, due to the two part SnpDVMOp requests that uses the snoop channel, a SnpDVMOp transaction must only be sent when the receiving RN has pre-allocated resources to accept both parts of the SnpDVMOp transaction.
- An RN must provide a response to a SnpDVMOp transaction only after it has received both SnpDVMOp request packets corresponding to that transaction.
- An RN must provide a response to a SnpDVMOp only when it can accept a further SnpDVMOp from an MN.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- 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 RNs that do not specify a number.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

8.1.4 DVMOp field value restrictions

Table 8-1 shows the Request, Data, Snoop, and Response message field value restrictions during a DVMOp transaction.

Table 8-1 DVMOp transaction field value restrictions

Message type	Field	Restriction
Request	QoS	None. Can be any value.
	TgtID	Expected to be node ID of MN. 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	Must be `0b0000000`…
	StashNIDValid Endian	Must be zero.
	ReturnTxnID StashLPIDValid StashLPID	Must be `0b00000000`.
	Opcode	Must be DVMOp.
	Size	Must be 8-byte.
	Addr	See *DVM Operations* on page 8-264.
	NS	Must be zero.
	LikelyShared	Must be zero.
	AllowRetry	Can be any value, because a DVMOp can be given a Retry.
	Order	Must be `0b00`.
	PCrdType	Must be `0b0000` if AllowRetry is asserted, otherwise the credit type value.
	Allocate	Must be zero.
	Cacheable	Must be zero.
	Device	Must be zero.
	EWA	Must be zero.
	SnpAttr	Must be zero.
	LPID	None. Don’t Care.
	Excl SnoopMe	Must be zero.
	ExpCompAck	Must be zero.
	TraceTag	None.
	RSVDC	None. Don’t Care.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 8-1 DVMOp transaction field value restrictions (continued)

Message type	Field	Restriction
Data	QoS	None. Can be 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	Must be `0b0000000`…
	Opcode	Must be NonCopyBackWriteData.
	RespErr	Must be `0b00` or `0b10`.
	Resp	Must be `0b000`.
	FwdState DataPull DataSource	Must be `0b000`.
	DBID	None. Don’t Care.
	CCID	Must be `0b00`.
	DataID	Must be `0b00`.
	TraceTag	None.
	BE	Only BE[7:0] must be asserted.
	Data	Unused bits must be zero for Data[63:0] and Data[n:64] = Don’t Care.
	DataCheck	Must be the appropriate value for the Data field.
	Poison	None. Can take any value.
SnpDVMOp	QoS	None. Can be any value.
	SrcID	Must be node ID of MN.
	TxnID	An ID generated by MN.
	FwdNID	Must be `0b0000000`…
	VMIDExt	Must be used for VMID Extension.
	Opcode	Must be SnpDVMOp.
	Addr	See *DVM Operations* on page 8-264.
	NS	Must be zero.
	DoNotGoToSD DoNotDataPull	Must be zero.
	RetToSrc	Must be zero.
	TraceTag	None.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 8-1 DVMOp transaction field value restrictions (continued)

Message type		Field	Restriction
Response	RetryAck	QoS	None. Can be any value.
		TgtID	Must be ID of the original Requester.
		SrcID	Must be ID of the MN that is handling DVMs.
		TxnID	Must match TxnID of the original request.
		Opcode	Must be RetryAck.
		RespErr	Must be `0b00`.
		Resp	Must be `0b000`.
		FwdState DataPull	Must be `0b000`.
		DBID	None. Don’t Care.
		PCrdType	None. Can be any value.
		TraceTag	None.
	DBIDResp	QoS	None. Can be any value.
		TgtID	Must be ID of the original Requester.
		SrcID	Must be ID of the MN that is handling DVMs.
		TxnID	Must match TxnID of the original request.
		Opcode	Must be DBIDResp.
		RespErr	Must be `0b00`.
		Resp	Must be `0b000`.
		FwdState DataPull	Must be `0b000`.
		DBID	Generated by the MN that is handling DVMOps.
		PCrdType	Must be `0b0000`.
		TraceTag	None.
	SnpResp	QoS	None. Can be any value.
		TgtID	Must be ID of the MN that is handling DVMOps.
		SrcID	Must be ID of the node that is responding to the snoop.
		TxnID	Must match the TxnID of the SnpDVMOp snoop request.
		Opcode	Must be SnpResp.
		RespErr	Must be `0b00` or `0b11`.
		Resp	Must be `0b000`.
		FwdState DataPull	Must be `0b000`.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 8-1 DVMOp transaction field value restrictions (continued)

Message type		Field	Restriction
Response	SnpResp	DBID	None. Don't Care.
		TraceTag	None.
		PCrdType	Must be `0b0000`.
	Comp	QoS	None. Can be any value.
		TgtID	Must be ID of the original Requester
		SrcID	Must be ID of the MN that is handling DVMs.
		TxnID	Must match TxnID of the original request.
		Opcode	Must be Comp.
		RespErr	Must be `0b00` or `0b11`.
		Resp	Must be `0b000`.
		FwdState DataPull	Must be `0b000`.
		DBID	Generated by the MN that is handling DVMs.
		PCrdType	Must be `0b0000`.
		TraceTag	None.

8.1.5 Field value requirements

Both SnpDVMOp request packets, corresponding to a single DVMOp, must have the same value in the following fields:

- TxnID
- Opcode
- SrcID
- TgtID

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

8.2 DVM Operation types

The following DVM Operations are supported:

- TLB Invalidate.
- Branch Predictor Invalidate.
- Instruction Cache Invalidate:
  - Physical address invalidate.
  - Virtual address invalidate.
- Synchronization.

8.2.1 DVMOp payload

The payload of a DVM operation from the RN to the MN is distributed within:

- The address field in the DVM request from the RN.
- The lower 8 bytes of write data in the NonCopyBackWrData packet.

The payload of a DVM operation from the MN to the RN is distributed over two SnpDVMOp request packets using the address fields.

The various fields in the payload and their encodings are shown in Table 8-2.

Table 8-2 DVMOp fields and encodings

Field	Bits	Function
VA Valid	1	`0b1` indicates that the specified address is valid
VMID Valid	1	`0b1` indicates that the *Virtual Machine IDentifier* (VMID) or *Virtual Index* (VI) is valid
ASID Valid	1	`0b1` indicates that the *Address Space IDentifier* (ASID) or VI is valid
Security	2	Indicates that the transaction applies to: `0b00` Secure and Non-secure `0b01` Reserved `0b10` Secure `0b11` Non-secure
Exception Level	2	Indicates that the transaction applies to: `0b00` Hypervisor and all Guest OS `0b01` EL3[a] `0b10` Guest OS `0b11` Hypervisor
DVMOp type	3	Indicates the DVM operation type as: `0b000` TLB Invalidate `0b001` Branch Predictor Invalidate `0b010` Physical Instruction Cache Invalidate `0b011` Virtual Instruction Cache Invalidate `0b100` Synchronization `0b101-0b111` Reserved
VMID	8	Virtual Machine ID VMID[7:0] or Virtual Index VA[27:20]
ASID	16	Address Space ID or Virtual Index VA[19:12][b]

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 8-2 DVMOp fields and encodings (continued)

Field	Bits	Function
S2-S1 Staged Invalidation	2	Indicates Stage 2 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` Reserved.
Leaf Entry Invalidation	1	`0b1` indicates that only leaf level translation invalidation is required
VA or PA	49 to 53 44 to 52	Virtual address Physical address
VMIDExt	8	Virtual Machine ID VMID[15:8]

a. DVMv8 only.
b. When used as Virtual Index, the upper 8-bits of ASID are Don't Care.

8.2.2 DVMOp and SnpDVMOp packet

Table 8-3 on page 8-263 shows the distribution of the payload in the DVMOp request from the RN, using 8-byte write semantics, and the distribution of the payload in the SnpDVMOp requests from the MN.

In the DVMOp, the combination of the address field in the request and the 8-byte write data transports the complete payload. Addr[3] is not used in the request and must be set to zero.

In the two SnpDVMOp requests the combination of the two address fields transports the complete payload. Addr[3] 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.

Note

In Table 8-3 on page 8-263, the number given shows which Address or Data bit is replaced by the DVMOp field.

For example, the VA Valid field is placed in the same position that Addr[4] normally occupies. In a Request packet, this would be the fifth bit position in the Addr field, but in a Snoop packet it would be the second bit position because the Snoop packet does not include the three least significant address bits.

Also, PA[6] is placed in the same position that Data[4] normally occupies in a write data packet, and in the same position that Addr[4] normally occupies in a Snoop packet. PA[6] is provided in the Part 2 Snoop packet, while VA Valid is provided in the Part 1 Snoop packet.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 8-3 DVMOp and SnpDVMOp request payloads using a 49-bit VA and 44-bit PA

Field	Bits	Request Addr	Data Data	Snoop Addr Part 1	Snoop Addr Part 2	Notes
Part Num	1	[3]	-	[3]	[3]	Must be `0b0` for the Request and Snoop Part 1. Must be `0b1` for Snoop Part 2.
VA Valid	1	[4]	-	[4]	-	-
VMID Valid	1	[5]	-	[5]	-	-
ASID Valid	1	[6]	-	[6]	-	-
Security	2	[8:7]	-	[8:7]	-	-
Exception Level	2	[10:9]	-	[10:9]	-	-
DVMOp type	3	[13:11]	-	[13:11]	-	-
VMID[7:0]	8	[21:14]	-	[21:14]	-	For VMID[15:8], see below.
ASID	16	[37:22]	-	[37:22]	-	-
S2-S1 Staged Invalidation	2	[39:38]	-	[39:38]	-	-
Leaf Entry Invalidation	1	[40]	-	[40]	-	-
PA[(MPA-1):6]	(MPA-6)	-	[(MPA-3):4]	-	[(MPA-3):4]	-
VA bits when REQ Addr width = 44 bits						
VA[45:6] VA[48:46]	40 3	- -	[43:4] [46:44]	- [43:41]	[43:4] -	Either VA or PA are used not both.
Additional VA bits where REQ Addr = 45 bits						
VA[49] VA[50]	1 1	- -	[47] [48]	- [44]	[44] -	Either VA or PA are used not both.
Additional VA bits where REQ Addr = 46 to 52 bits						
VA[51] VA[52]	1 1	- -	[49] [50]	- [45]	[45] -	Either VA or PA are used not both.
VMID[15:8]	8	-	[63:56]	VMIDExt	-	VMIDExt is a separate field outside of the Addr field. See Table 8-2 on page 8-261.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

8.3 DVM Operations

This section describes the supported DVM Operations:

- *TLB Invalidate* on page 8-265.
- *Branch Predictor Invalidate* on page 8-267.
- *Physical Instruction Cache Invalidate* on page 8-268.
- *Virtual Instruction Cache Invalidate* on page 8-269.
- *Synchronization* on page 8-270.

Table 8-4 shows the values for the Part Num field in all supported DVM Operations.

Table 8-4 Part Num field values

Addr		Value			Status
Bit	Field	Request	Snoop		
			Part 1	Part 2	
[3]	Part Num	0b0	0b0	0b1	Not utilized in the Request

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

8.3.1 TLB Invalidate

Table 8-5 shows the supported TLB Invalidate operations.

Table 8-5 TLB Invalidate operations

Addr								Operation
[13:11] DVMOp type	[10:9] Exception Level	[8:7] Secure	[6] ASID valid	[5] VMID valid	[40] LEAF	[39:38] S2-S1	[4] VA valid	
0b000 TLBI	0b10 All Guest OS	0b10 Secure	0b0 Ignore	0b0 Ignore	0b0 Ignore	0b00[a]	0b0 Ignore	Secure TLB Invalidate all
			0b0 Ignore	0b0 Ignore	0b0 Ignore	0b00[a]	0b1 Match	Secure TLB Invalidate by VA
			0b0 Ignore	0b0 Ignore	0b1 Leaf	0b00[a]	0b1 Match	Secure TLB Invalidate by VA Leaf Entry only
			0b1 Match	0b0 Ignore	0b0 Ignore	0b00[a]	0b0 Ignore	Secure TLB Invalidate by ASID
			0b1 Match	0b0 Ignore	0b0 Ignore	0b00[a]	0b1 Match	Secure TLB Invalidate by ASID and VA
			0b1 Match	0b0 Ignore	0b1 Leaf	0b00[a]	0b1 Match	Secure TLB Invalidate by ASID and VA Leaf Entry only
	0b10 All Guest OS	0b11 Non-secure	0b0 Ignore	0b0 Ignore	0b0 Ignore	0b00[a]	0b0 Ignore	All Guest OS TLB Invalidate all
			0b0 Ignore	0b1 Match	0b0 Ignore	0b01 S1	0b0 Ignore	Guest OS TLB Invalidate all Stage 1 invalidation only
			0b0 Ignore	0b1 Match	0b0 Ignore	0b00[a]	0b0 Ignore	Guest OS TLB Invalidate all ARMv7 must carry out Stage 1 and 2 invalidation
			0b0 Ignore	0b1 Match	0b0 Ignore	0b00[a]	0b1 Match	Guest OS TLB Invalidate by VA
			0b0 Ignore	0b1 Match	0b1 Leaf	0b00[a]	0b1 Match	Guest OS TLB Invalidate by VA Leaf Entry only
			0b1 Match	0b1 Match	0b0 Ignore	0b00[a]	0b0 Ignore	Guest OS TLB Invalidate by ASID
			0b1 Match	0b1 Match	0b0 Ignore	0b00[a]	0b1 Match	Guest OS TLB Invalidate by ASID and VA
			0b1 Match	0b1 Match	0b1 Leaf	0b00[a]	0b1 Match	Guest OS TLB Invalidate by ASID and VA Leaf Entry only
			0b0 Ignore	0b1 Match	0b0 Ignore	0b10 S2	0b1 IPA[b]	Guest OS TLB Invalidate by IPA
			0b0 Ignore	0b1 Match	0b1 Leaf	0b10 S2	0b1 IPA[b]	Guest OS TLB Invalidate by IPA Leaf Entry only

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 8-5 TLB Invalidate operations (continued)

Addr								Operation
[13:11] DVMOp type	[10:9] Exception Level	[8:7] Secure	[6] ASID valid	[5] VMID valid	[40] LEAF	[39:38] S2-S1	[4] VA valid	
0b000 TLBI	0b11 Hypervisor	0b11 Non-secure	0b0 Ignore	0b0 Ignore	0b0 Ignore	0b00[a]	0b0 Ignore	Hypervisor TLB Invalidate all
			0b0 Ignore	0b0 Ignore	0b0 Ignore	0b00[a]	0b1 Match	Hypervisor TLB Invalidate by VA
			0b0 Ignore	0b0 Ignore	0b1 Leaf	0b00[a]	0b1 Match	Hypervisor TLB Invalidate by VA Leaf Entry only
			0b1 Match	0b0 Ignore	0b0 Ignore	0b00[a]	0b0 Ignore	Hypervisor TLB Invalidate by ASID
			0b1 Match	0b0 Ignore	0b0 Ignore	0b00[a]	0b1 Match	Hypervisor TLB Invalidate by ASID and VA
			0b1 Match	0b0 Ignore	0b1 Leaf	0b00[a]	0b1 Match	Hypervisor TLB Invalidate by ASID and VA Leaf Entry only
	0b01 EL3	0b10 Secure	0b0 Ignore	0b0 Ignore	0b0 Ignore	0b00[a]	0b1 Match	EL3 TLB Invalidate by VA
			0b0 Ignore	0b0 Ignore	0b1 Leaf	0b00[a]	0b1 Match	EL3 TLB Invalidate by VA Leaf Entry only
			0b0 Ignore	0b0 Ignore	0b0 Ignore	0b00[a]	0b0 Ignore	EL3 TLB Invalidate All

a. All DVMv7 transactions must use 0b00.

b. IPA is the Intermediate Physical Address. The IPA is sent using the same format as the Virtual Address (VA).

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

8.3.2 Branch Predictor Invalidate

This section shows the Branch Predictor Invalidate operations.

Table 8-6 shows the fixed value fields in the Branch Predictor Invalidate operation.

Table 8-6 Branch Predictor Invalidate operation fixed values

Addr Bits	Field	Value	Status
[3]	Part Num	-	See Table 8-4 on page 8-264
[5]	VMID Valid	0b0	VMID field not valid
[6]	ASID Valid	0b0	ASID field not valid
[8:7]	Secure	0b00	Applies to both secure and Non-secure
[10:9]	Exception Level	0b00	Applies to all Guest OS and Hypervisor
[21:14]	VMID VMIDExt	0xXX	VMID not specified
[37:22]	ASID	0xXXXX	ASID not specified
[39:38]	S2, S1 Staged Invalidation	0b00	Reserved, set to Zero
[40]	Leaf Entry Invalidation	0b0	Reserved, set to Zero

Note

The use of Branch Predictor Invalidate with a 16-bit ASID is not supported.

Table 8-7 shows the operations supported by Branch Predictor Invalidate.

Table 8-7 Branch Predictor Invalidate operations

Addr		Operation
[13:11] DVMOp type	[4] VA valid	
0b001	0b0 Ignore	Branch Predictor Invalidate all
	0b1 Match	Branch Predictor Invalidate by VA

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

8.3.3 Physical Instruction Cache Invalidate

This section shows the Physical Instruction Cache Invalidate operations.

Table 8-8 shows the fixed value fields in the Physical Instruction Cache Invalidate operation.

Table 8-8 Physical Instruction Cache Invalidate operation fixed values

Addr Bits	Field	Value	Status
[3]	Part Num	-	See Table 8-4 on page 8-264
[10:9]	Exception Level	0b00	Applies to all Guest OS and Hypervisor
[39:38]	S2, S1 Staged Invalidation	0b00	Reserved, set to zero
[40]	Leaf Entry Invalidation	0b0	Reserved, set to zero

Table 8-9 shows the operations supported by Physical Instruction Cache Invalidate.

Note

When Virtual Index is 0b11, then VA[19:12] and VA[27:20], at Addr[29:22] and Addr[21:14] respectively, are used as part of the Physical Address. Addr[37:30] are not used, and are Don't Care values.

Table 8-9 Physical Instruction Cache Invalidate operations

[13:11] DVMOp type	[8:7] Secure	[6:5] Virtual Index	[4] VA	Operation
0b010 PICI	0b10 Secure	0b00	0b0 Ignore	Secure Physical Address Cache Invalidate all
		0b00	0b1 Match	Secure Physical Address Cache Invalidate by PA without Virtual Index
		0b11	0b1 Match	Secure Physical Address Cache Invalidate by PA with Virtual Index
	0b11 Non-secure	0b00	0b0 Ignore	Non-secure Physical Address Cache Invalidate all
		0b00	0b1 Match	Non-secure Physical Address Cache Invalidate by PA without Virtual Index
		0b11	0b1 Match	Non-secure Physical Address Cache Invalidate by PA with Virtual Index

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

8.3.4 Virtual Instruction Cache Invalidate

This section shows the Virtual Instruction Cache Invalidate operations.

Table 8-10 shows the fixed value fields in the Virtual Instruction Cache Invalidate operation.

Table 8-10 Virtual Instruction Cache Invalidate operation fixed values

Addr Bits	Field	Value	Status
[3]	Part Num	-	See Table 8-4 on page 8-264
[39:38]	S2, S1 Staged Invalidation	0b00	Reserved. set to zero
[40]	Leaf Entry Invalidation	0b0	Reserved, set to zero

Table 8-11 shows the operations supported by Virtual Instruction Cache Invalidate.

Table 8-11 Virtual Instruction Cache Invalidate operations

Addr						Operation
[13:11] DVMOp type	[10:9] Exception Level	[8:7] Secure	[6] ASID valid	[5] VMID valid	[4] VA valid	
0b011 VICI	0b00 Hypervisor and all Guest OS	0b00 Secure and Non-secure	0b0 Ignore	0b0 Ignore	0b0 Ignore	Invalidate all. Applies to Secure and Non-secure. Applies to Hypervisor and all Guest OS.
		0b11 Non-secure	0b0 Ignore	0b0 Ignore	0b0 Ignore	Invalidate all. Applies to Non-secure. Applies to Hypervisor and all Guest OS.
	0b10 Guest OS	0b10 Secure	0b1 Match	0b0 Ignore	0b1 Match	Secure Invalidate by ASID and VA.
		0b11 Non-secure	0b0 Ignore	0b1 Match	0b0 Ignore	Guest OS, Invalidate all.
			0b1 Match	0b1 Match	0b1 Match	Guest OS, Invalidate by ASID and VA.
	0b11 Hypervisor	0b11 Non-secure	0b0 Ignore	0b0 Ignore	0b1 Match	Hypervisor, Invalidate by VA.
			0b1 Match	0b0 Ignore	0b1 Match	Hypervisor, Invalidate by ASID and VA

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

8.3.5 Synchronization

This section shows the DVMSync Operation.

Table 8-12 shows the fixed value fields in the Sync operation.

Table 8-12 Sync operation fixed values

Addr Bits	Field	Value	Status
[3]	Part Num	-	See Table 8-4 on page 8-264
[4]	VA Valid	0b0	Not applicable
[5]	VMID Valid	0b0	Ignore VMID
[6]	ASID Valid	0b0	Ignore ASID
[8:7]	Secure	0b00	Applies to both Secure and Non-secure
[10:9]	Exception Level	0b00	Applies to all Guest OS and Hypervisor
[13:11]	DVMOp type	0b100	Synchronization message
[21:14]	VMID VMIDExt	0xXX	VMID not specified
[37:22]	ASID	0xXXXX	ASID not specified
[39:38]	S2, S1 Staged Invalidation	0b00	Set to zero
[40]	Leaf Entry Invalidation	0b0	Set to zero

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 9
Error Handling

This chapter describes the error handling requirements. It contains the following sections:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

9.1 Error types

This specification supports two types of error reporting at sub packet level, and two types of error reporting at packet level.

The packet level error reporting types are:

Data Error 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 ECC or a parity check.

Data Error reporting is supported by the RespErr, Poison, and DataCheck fields in the DAT packet.

Non-data Error 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

9.2 Error response fields

The RespErr field is used to indicate error conditions. The RespErr field is included in both response and data packets.

Table 9-1 shows the encoding of the RespErr field. See *Responses to exclusive requests* on page 6-238 for more details on the Exclusive Okay response.

Table 9-1 Error response field encodings

RespErr[1:0]	Name	Description
0b00	OK	Okay. Indicates that a Non-exclusive access has been successful. Also used to indicate an Exclusive access failure.
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.

The mixing of OK, DERR, and NDERR responses within a single transaction is permitted.

The mixing of EXOK, DERR and NDERR responses within a single transaction is permitted.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

9.3 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 then 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 *Response types* on page 4-163 for more details on the legal Resp field values. If a response to a transaction does not have a legal cache state, then the RespErr field must indicate a Non-data Error for all data packets. A Snoop response with error that does not have a legal cache state must not include data with the response.

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.

The DataSource field value is inapplicable and can take any value in Read data packets with Non-data error response.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

9.4 Error response use by transaction type

This section defines the permitted 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:

- *Read Transactions*.
- *Dataless transactions* on page 9-276.
- *Write transactions* on page 9-277.
- *Atomic transactions* on page 9-277.
- *Other transactions* on page 9-279.
- *Cache Stashing transactions* on page 9-279.
- *Snoop transactions* on page 9-279.

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.

9.4.1 Read Transactions

Read transactions can contain multiple CompData data packets. Each data packet can use a different error type, as indicated by Table 9-2, with the restriction that a single transaction cannot mix OK and EXOK responses.

Table 9-2 Read transaction's Data and Response packets legal RespErr field values

Read transaction	Associated Data and Response packets					
	Read Receipt	CompData				CompAck
		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	-	Y	N	Y	Y	OK

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 9-3 Read transaction's Data-only and Non-data packets legal RespErr field values

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
ReadClean ReadNotSharedDirty ReadShared	Y	N	Y	Y	Y	N	N	Y
ReadUnique	Y	N	Y	Y	Y	N	N	Y

9.4.2 Dataless transactions

A Data Error can be reported for a Dataless transaction when the processing of the transaction by another component encounters a data corruption error. This data error can be indicated back to the originating component, even though a transfer of data does not occur.

Table 9-4 shows the Dataless transaction packets legal RespErr field values.

Table 9-4 Dataless transaction packets legal RespErr field values

Dataless transaction	Associated Response packets				
	Comp				CompAck
	OK	EXOK	DERR	NDERR	
CleanUnique	Y	Y	Y	Y	OK
MakeUnique	Y	N	Y	Y	OK
CleanShared CleanSharedPersist CleanInvalid MakeInvalid	Y	N	Y	Y	-
Evict	Y	N	N	Y	-
StashOnceUnique StashOnceShared	Y	N	Y	Y	-

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

9.4.3 Write transactions

A Write transaction can include either a Non-data Error or a Data Error. Errors can be signalled in both directions, from the Requester to the Completer, and from the Completer back to the Requester.

For a Write transaction an error can be signalled from the Completer back to the Requester using either the combined CompDBIDResp or using the Comp response. It is permitted for the Completer to signal an error even before it has observed the WriteData for the transaction and this can occur when the processing of the transaction, such as the cache lookup, encounters a data corruption error.

Table 9-5 shows the Write transaction response packets legal RespErr field values.

Table 9-5 Write transaction response packets legal RespErr field values

Write transaction	Associated Response packets									
	DBIDResp	Comp				CompDBIDResp				CompAck
		OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR	
WriteNoSnp	OK	Y	Y	Y	Y	Y	Y	Y	Y	-
WriteUnique	OK	Y	N	Y	Y	Y	N	Y	Y	OK
WriteBack WriteClean WriteEvictFull	-	-	-	-	-	Y	N	Y	Y	-

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 9-6 shows the Write transaction data packets legal RespErr field values.

Table 9-6 Write transaction data packets legal RespErr field values

Write transaction	Associated data packets											
	WriteData				WriteDataCancel				NCBWrDataCompAck			
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
WriteNoSnp	Y	N	Y	N	Y	N	Y	N	-	-	-	-
WriteUnique	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
WriteBack WriteClean WriteEvictFull	Y	N	Y	N	-	-	-	-	-	-	-	-

9.4.4 Atomic transactions

It is permitted for a Completer to give a Comp response before it has received 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.

A Data Error or Non-data Error can be signaled at the following points within a transaction:

- With the DBIDResp response.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- 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 Data Error or Non-data Error, then 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 Non-data Error 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, might be able to provide such information, but this is not a requirement of this specification.

The permitted RespErr values in Atomic transactions are an amalgamation of those permitted in Read and Write transactions.

A Data Error can vary between data packets.

Table 9-7 shows the Atomic transaction response packets legal RespErr field values

Table 9-7 Atomic transaction response packets legal RespErr field values

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 9-8 shows the Atomic transaction data packets legal RespErr field values.

Table 9-8 Atomic transaction data packets legal RespErr field values

Atomic transaction	Associated response packets							
	WriteData				CompData			
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
AtomicStore	Y	N	Y	N	-	-	-	-
AtomicLoad AtomicSwap AtomicCompare	Y	N	Y	N	Y	N	Y	Y

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

9.4.5 Other transactions

This section describes the error handling requirements for the DVMOp and PrefetchTgt transactions.

DVMOp

A DVMOp transaction can include a Non-data Error in the Comp response. The interconnect must 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 *Interoperability of Poison and DataCheck* on page 9-284.

Table 9-9 shows the DVM transaction packets legal RespErr field values.

Table 9-9 DVM transaction packets legal RespErr field values

DVM transaction	Associated response and data packets								
	DBIDResp	NCBWrData				Comp			
		OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
DVMOp	OK	Y	N	Y	N	Y	N	Y	Y

PrefetchTgt

A PrefetchTgt transaction request to a non-supporting address must be discarded.

Note

A component is permitted to record and report such an error.

9.4.6 Cache Stashing transactions

If the specified stash target does not support receiving Stash type snoops then 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, it must complete the transaction in a protocol-compliant manner without signaling an error.

9.4.7 Snoop transactions

A snoop transaction response that includes data can indicate a Data Error. A Snoop transaction response that includes data can mix Okay and Data Error responses for different packets within the transaction. A snoop transaction response that does not include data can indicate a Non-data Error.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 9-10 shows the Snoop request response packets legal RespErr field values.

Table 9-10 Snoop request response packets legal RespErr field values

Snoop Transaction	Associated Data and Response packets							
	SnpResp				SnpRespData			
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
SnpOnce SnpClean SnpNotSharedDirty SnpShared SnpUnique SnpUniqueStash SnpCleanShared SnpCleanInvalid	Y	N	N	Y	Y	N	Y	N
SnpStashUnique SnpStashShared SnpMakeInvalid SnpMakeInvalidStash SnpDVMOp	Y	N	N	Y	-	-	-	-

A DERR in response to the Data Pull request is not expected to be transferred to the Comp response to the Stash request.

A fowarding Snoop transaction can include an error indication similar to those in a snoop as well as in a completion with data from the Snoopee to the Requester. When simultaneously forwarding data to the Requester and returning Data to Home, it is permitted for only one response to include an indication of a Data Error if the other response does not encounter the error.

The Non-data Error in SnpRespFwded is permitted to include the case where the error is detected after the data is forwarded to the Requester but before the response is sent to Home. FwdState in the SnpRespFwded response with Non-data Error must be the RESP state in the CompData to the Requester.

Table 9-11 shows the forward Snoop response packets legal RespErr field values.

Table 9-11 Forward Snoop response packets legal RespErr field values

Snoop transaction	Associated Response packets							
	SnpResp				SnpRespFwded			
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
SnpOnceFwd SnpCleanFwd SnpNotSharedDirtyFwd SnpSharedFwd SnpUniqueFwd	Y	N	N	Y	Y	N	Y	Y

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 9-12 shows the forward snoop Data response packets legal RespErr field values.

Table 9-12 Forward snoop Data response packets legal RespErr field values

Snoop transaction	Associated Data packets							
	SnpRespData SnpRespDataFwded				CompData			
	OK	EXOK	DERR	NDERR	OK	EXOK	DERR	NDERR
SnpOnceFwd SnpCleanFwd SnpNotSharedDirtyFwd SnpSharedFwd SnpUniqueFwd	Y	N	Y	N	Y	N	Y	N

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

9.5 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 master.
  - Is permitted 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 to over poison the data.

Poison must be accurate if there are any valid bytes in the 64-bit chunk, which is Poison granularity, Otherwise, the Poison bit is a Don’t Care, that is, when all 8 bytes in the 64-bit chunk are invalid, then it is a Don’t Care.

A Data_Poison property is used to indicate if a component supports Poison.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

9.6 Data Check

The DataCheck field is used to detect data errors in the DAT packet.

When Data Check is supported:

- The DAT packet carries eight Data Check bits per 64 bits of data.
- The Data Check bit is a parity bit that generates Odd Byte parity.

The Data_Check property is used to indicate if Data Check is supported.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

9.7 Interoperability of Poison and DataCheck

If the recipient of a Data packet does not support the Poison and DataCheck features then the interconnect must enumerate and convert, as necessary, the Poison and Data Check error responses to a Data Error in the DAT packet.

If support for the Poison and DataCheck features is not similar across an interface, then 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, then 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 Don’t Care.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

9.8 Hardware and software error categories

This specification defines two error categories, a software based error and a hardware based error.

9.8.1 Software based error

A software based error occurs when multiple accesses to the same location are made with mismatched Snoopable or Memory attributes.

A software based error can cause a loss of coherency and the corruption of data values. This specification requires 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 might be required to return a peripheral device that has been accessed incorrectly to a known working state is IMPLEMENTATION DEFINED.

9.8.2 Hardware based error

A hardware based error is defined as any protocol error that is not a software based error.

Warning

If a hardware based error occurs then recovery from the error is not guaranteed. The system might crash, lock-up, or suffer some other non-recoverable failure.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 10
Quality of Service

This chapter describes the mechanisms in the CHI protocol to support *Quality of Service* (QoS). It contains the following sections:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

10.1 Overview

A system might 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

10.2 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 is dependent on the source type and the class of traffic, with ascending values of QoS indicating a higher priority level. The source might also dynamically vary this value depending on some accumulated latency and required throughput metric.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

10.3 Repeating a transaction with higher QoS value

When a transaction has been sent with a particular QoS value, it is permitted to send the same transaction again with a different, typically higher, QoS value. The Completer is required to handle this situation as multiple different requests.

In this situation, if one of the transactions receives a RetryAck response, then it is permitted to cancel the transaction and return the credit. See *Credit Return* on page 2-127.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 11
Data Source and Trace Tag

This chapter describes the mechanisms of Data Source and Trace Tag that provide additional support for the debugging and tracing of systems and the monitoring of performance. It contains the following sections:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

11.1 Data Source indication

This specification permits 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.

11.1.1 DataSource value assignment

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:
  - **0b110** PrefetchTgt memory prefetch was useful.
    Read data was obtained from Slave with lower latency as the PrefetchTgt request already read or initiated a read of data from memory.
  - **0b111** 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 might not be useful. Examples are that the prefetch was dropped by the Slave, the data obtained by the prefetch was replaced in the buffer, or the Read request arrived at the Slave before the prefetch.

- For a response not from memory, that is, from a cache, the DataSource value is IMPLEMENTATION DEFINED.
  This specification recommends, but does not require, settings for DataSource in these cases.
  A component is permitted 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 responder is permitted to not support sending a useful DataSource value:
  - The responder, except for a memory SN-F, must return a `0b000` value.
  - A memory SN-F component is permitted to return `0b111` as a default value.

  Such exceptions must be understood by the system.

11.1.2 Crossing a chip-to-chip interface

It is the responsibility of the chip interface module, if one exists, to map DataSource values in the incoming Data packets to different values to identify that the response came from the remote chip caches.

Example approaches that the chip interface module might take are:

- Group the remote caches into a single encoding, as Figure 11-1 on page 11-293 shows.
- Have a maximum size of an eight entry table, to remap the implementation values of the DataSource field in the incoming Data packet to new values.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Suggested DataSource values

Figure 11-1 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.
- All the caches in the remote peer chip are grouped together.
- A non-memory component that is not programmed to identify itself as the source of data can return the default value of `0b000`.

Table 11-1 lists the suggested DataSource encodings.

Table 11-1 Suggested DataSource value encodings

DataSource	Suggested mapping
`0b000`	Non-memory default. Source does not support sending a useful DataSource value
`0b001`	Peer processor cache within local cluster
`0b010`	Local cluster cache
`0b011`	Interconnect cache
`0b100`	Peer cluster caches
`0b101`	Remote chip caches

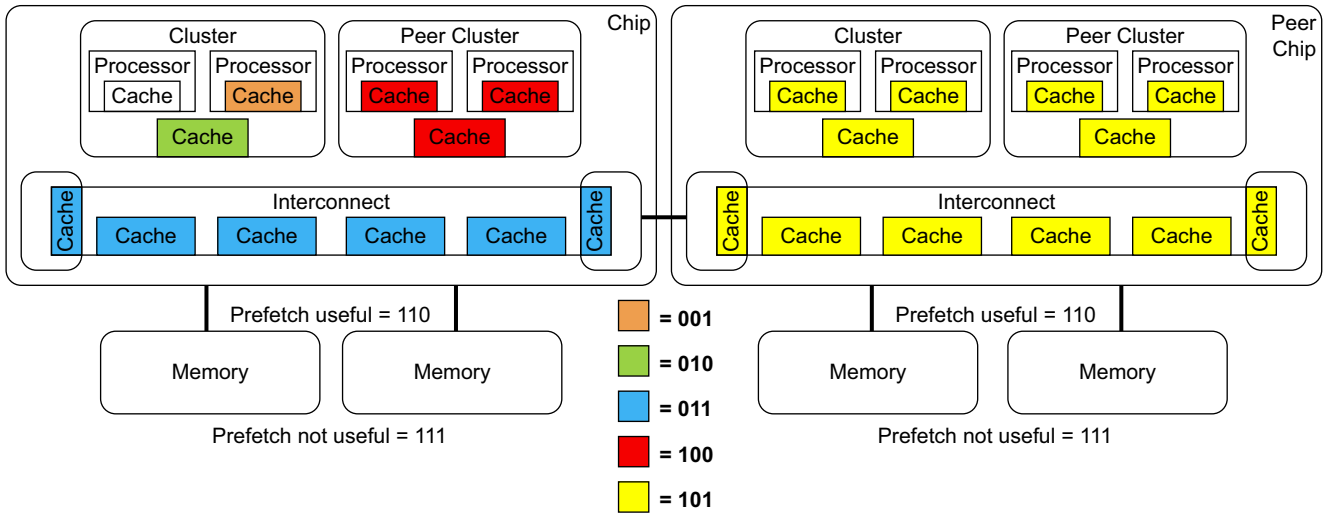


Figure 11-1 Suggested DataSource values

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

11.1.3 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 SNF, the Requester can determine the usefulness of sending PrefetchTgt requests and can modulate the rate, as well as the sending, of PrefetchTgt requests.
- Can be used by performance profiling and debug software to evaluate and optimize the data sharing pattern.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

11.2 Trace Tag

This specification includes a TraceTag bit per channel that provides enhanced support for debugging, tracing, and performance measurement of systems.

11.2.1 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 a 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.
- 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 then 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 RN might 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 be dependent 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 DataSepResp has the TraceTag bit set.
  - The TraceTag bit in the NCBWrDataCompAck response from RN 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, it must preserve the value 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 will be 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

Examples of Request-Response pairs are:

- Snoop response, with or without data, in response to a Snoop request.
- A Snoop response in response to a SnpDVMOp request.
- Data response from SN in response to a Read request.
- Spawned requests from HN-F:
  - Snoops generated in response to a request from RN.
  - Request to SN-F generated in response to a request from RN.
- Spawned request from HN-I:
  - Read or Write request to SN-I generated in response to a request from RN.
- A CompAck from RN in response to CompData, Comp or RespSepData.
- A RetryAck response from HN or SN to any request.
- A ReadReceipt response from HN or SN to a Read request or from SN to a ReadNoSnpSep request.
- A DBIDResp response to a Write request.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 12
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:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.1 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.
- Flow control across a link.

Figure 12-1 shows a typical system using link based communication.

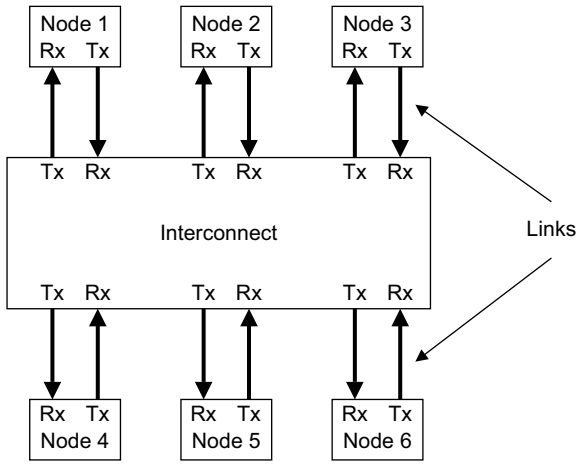


Figure 12-1 System using link based communication

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.2 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 12-2 shows the link requirements.

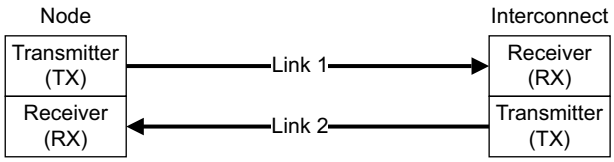


Figure 12-2 Two-way link communication

12.2.1 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 12-3 shows the outbound and inbound links at a node. The interface at the interconnect has a complementary pair of links.

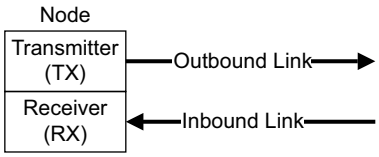


Figure 12-3 Outbound and inbound links

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.3 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. In this specification, 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.4 Channel

In this specification, 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 12-1 shows the channels, and the mapping onto the RN and SN component channels.

Table 12-1 Channels' mapping onto the RN and SN component channels

Channel	Description	Usage	RN Channel	SN Channel
REQ Request	The request channel transfers flits associated with request messages such as Read Requests and Write Requests. See *REQ channel* on page 12-306.	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 *RSP channel* on page 12-307.	Responses from the Completer	RXRSP	TXRSP
		Snoop Response and Completion Acknowledge	TXRSP	-
SNP Snoop	The snoop channel transfers flits associated with Snoop and SnpDVMOp Request messages. See *SNP channel* on page 12-308.	All Snoop requests	RXSNP	-
DAT Data	The data channel transfers flits associated with protocol messages that have a data payload such as read completion and write data messages. See *DAT channel* on page 12-309.	Write data, and Snoop response data from an RN	TXDAT	RXDAT
		Read data	RXDAT	TXDAT

12.4.1 Channel dependencies

The following dependencies are permitted between the channels in the protocol.

For RN:

- An RN must make forward progress on the inbound SNP channel without requiring forward progress on outbound REQ channel.
- An RN is permitted to wait for forward progress on the outbound RSP channel before making forward progress on the inbound SNP channel.
- An RN is permitted to wait for forward progress on the outbound DAT channel before making forward progress on the inbound SNP channel.
- An RN must make forward progress on the inbound RSP channel without requiring forward progress on any other channel.
- An RN must make forward progress on the inbound DAT channel without requiring forward progress on any other channel.

Note

The requirement that an RN must make forward progress on the inbound RSP and DAT channel, without requiring forward progress on any other channel, means that an RN must be able to accept all Comp and CompData responses for outstanding transactions without sending any CompAck responses.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

For SN:

- An SN is permitted to wait for forward progress on the outbound RSP channel before making forward progress on the inbound REQ channel.
- An SN must make forward progress on the inbound REQ channel without requiring forward progress on the outbound DAT channel.
- An SN must make forward progress on the inbound DAT channel without requiring forward progress on any other channel.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.5 Port

A Port is defined as the set of all links at the interface of a node.

Figure 12-4 shows the relationship between links, channels, and port. See *Node interface definitions* on page 12-304 for the specific node requirements, See *Channel interface signals* on page 12-306, and Chapter 13 *Link Handshake* for signal details.

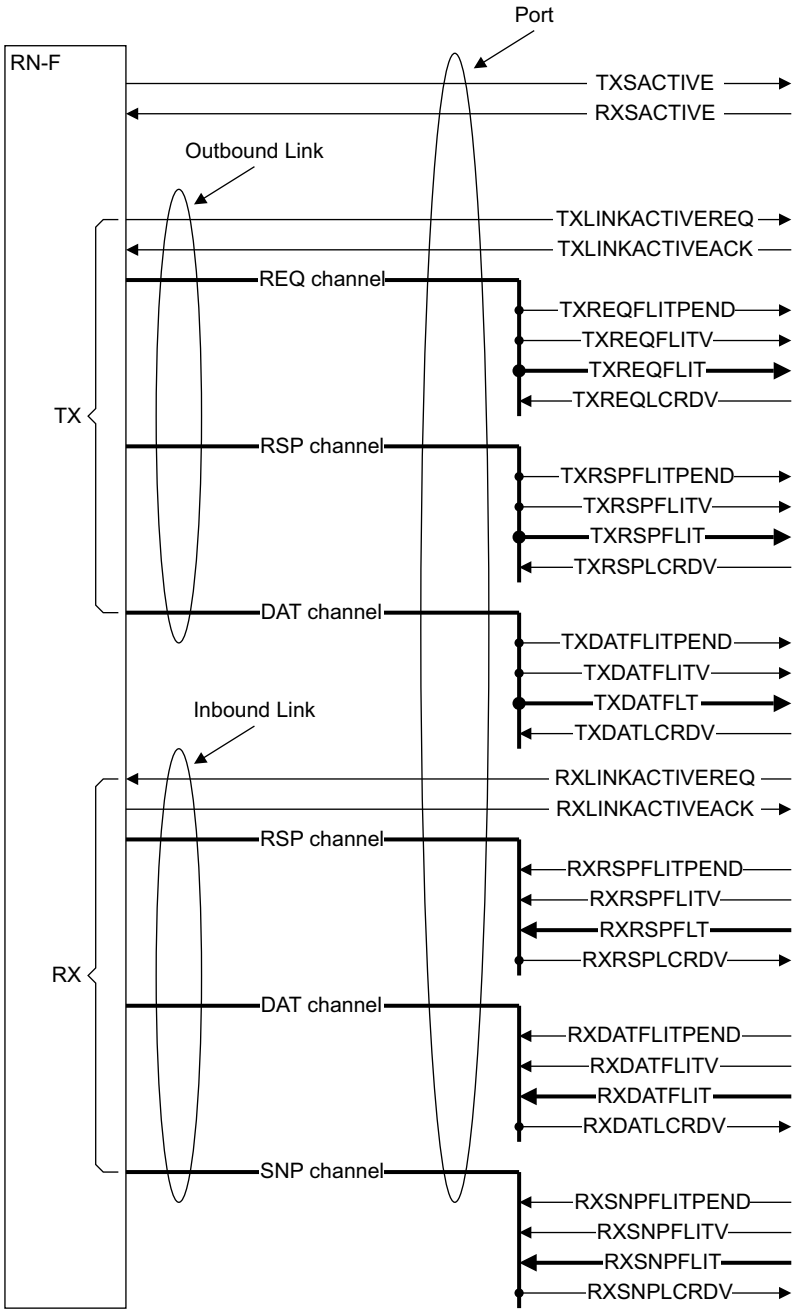


Figure 12-4 Relationship between links, channels, and port

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.6 Node interface definitions

Nodes communicate by exchanging Link flits using the node interface. This section describes the node interfaces:

- *Request Nodes*.
- *Slave Nodes* on page 12-305.

Note

The LINKACTIVE interface pins and signals used by each node for link management are described in Chapter 13 *Link Handshake*.

12.6.1 Request Nodes

This section describes the Request Node interfaces:

- *RN-F*.
- *RN-D*.
- *RN-I* on page 12-305.

RN-F

The RN-F interface uses all channels and is used by a fully coherent Requester such as a core or cluster. Figure 12-5 shows the RN-F interface.

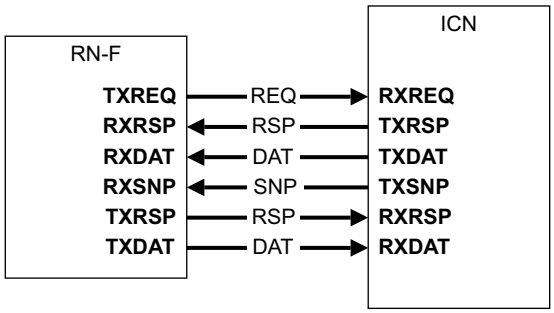


Figure 12-5 RN-F interface

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 *DVM transaction flow* on page 8-252 for details. Figure 12-6 shows the RN-D interface.

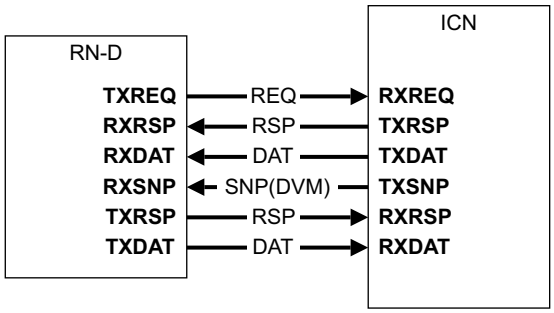


Figure 12-6 RN-D interface

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 12-7 shows the RN-I interface.

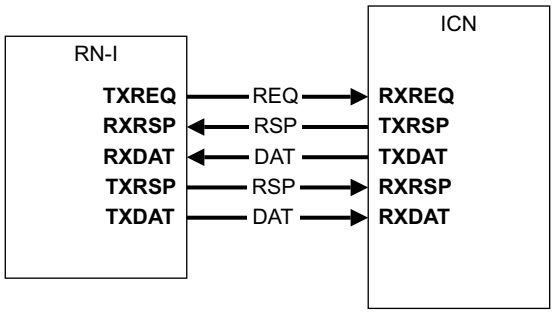


Figure 12-7 RN-I interface

12.6.2 Slave Nodes

This section describes the Slave Node interfaces:

- SN-F.
- SN-I.

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 12-8 shows the SN-F and SN-I interface.

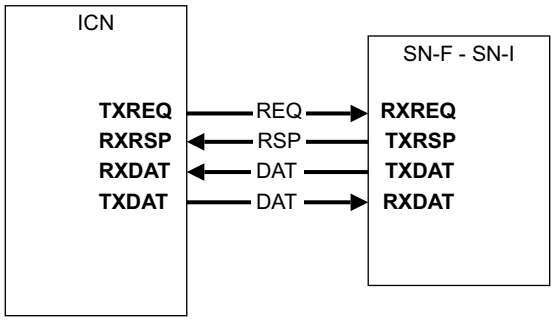


Figure 12-8 SN-F and SN-I interface

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.7 Channel interface signals

This section describes the channel interfaces. It contains the following sections:

- *REQ channel*.
- *RSP channel* on page 12-307.
- *SNP channel* on page 12-308.
- *DAT channel* on page 12-309.

12.7.1 REQ channel

Figure 12-9 shows the REQ channel interface pins, where R is the width of **REQFLIT**.

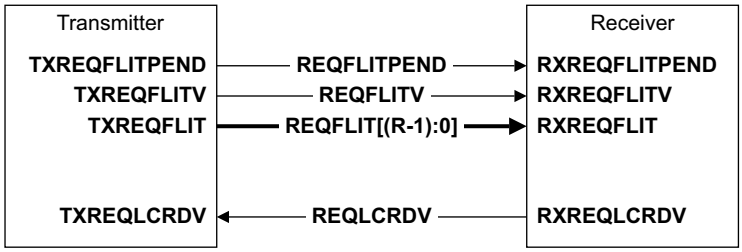


Figure 12-9 REQ channel interface pins

Table 12-2 shows the REQ channel interface signals.

Table 12-2 REQ channel interface signals

Signal	Description
REQFLITPEND	Request Flit Pending. Early indication that a request flit might be transmitted in the following cycle. See *Flit level clock gating* on page 13-341.
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 *Request flit* on page 12-310 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 *L-Credit flow control* on page 13-339.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.7.2 RSP channel

Figure 12-10 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.

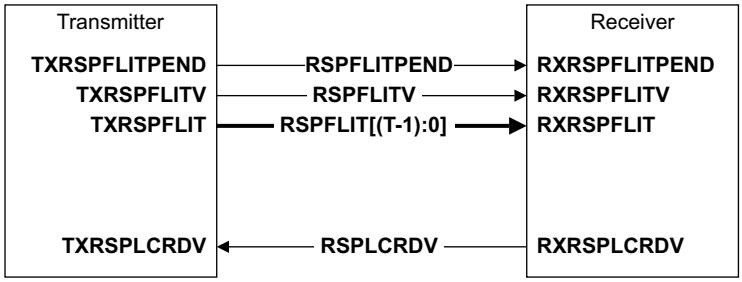


Figure 12-10 RSP channel interface pins

Table 12-3 shows the RSP channel interface signals.

Table 12-3 RSP channel interface signals

Signal	Description
RSPFLITPEND	Response Flit Pending. Early indication that a response flit might be transmitted in the following cycle. See *Flit level clock gating* on page 13-341.
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 *Response flit* on page 12-311 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 *L-Credit flow control* on page 13-339.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.7.3 SNP channel

Figure 12-11 shows the SNP channel interface pins, where S is the width of **SNPFLIT**.

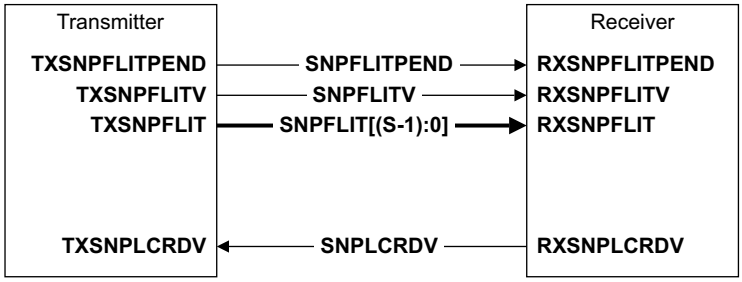


Figure 12-11 SNP channel interface pins

Table 12-4 shows the SNP channel interface signals.

Table 12-4 SNP channel interface signals

Signal	Description
SNPFLITPEND	Snoop Flit Pending. Early indication that a snoop flit might be transmitted in the following cycle. See *Flit level clock gating* on page 13-341.
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 *Snoop flit* on page 12-312 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 *L-Credit flow control* on page 13-339.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.7.4 DAT channel

Figure 12-12 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.

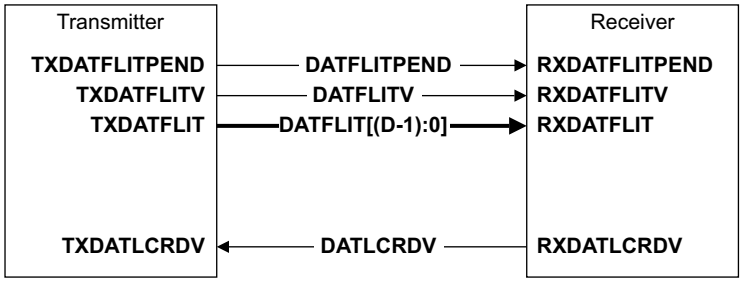


Figure 12-12 DAT channel interface pins

Table 12-5 shows the DAT channel interface signals.

Table 12-5 DAT channel interface signals

Signal	Description
DATFLITPEND	Data Flit Pending. Early indication that a data flit might be transmitted in the following cycle. See *Flit level clock gating* on page 13-341.
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 *Data flit* on page 12-313 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 *L-Credit flow control* on page 13-339.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.8 Flit packet definitions

This section defines the flit format. See:

- *Request flit*.
- *Response flit* on page 12-311.
- *Snoop flit* on page 12-312.
- *Data flit* on page 12-313.

12.8.1 Request flit

Table 12-6 shows the Request flit format in a REQ channel packet starting at bit zero.

Table 12-6 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	8	-
ReturnNID StashNID	7 to 11	Used for DMT Used in Stash transactions
StashNIDValid Endian	1	Used in Stash transactions Used in Atomic transactions
ReturnTxnID[7:0] {0b00, StashLPIDValid, StashLPID[4:0]}	8	Used for DMT SBZ Used in Stash transactions Used in Stash transactions
Opcode	6	-
Size	3	-
Addr	44 to 52	Width determined by Req_Addr_Width
NS	1	-
LikelyShared	1	-
AllowRetry	1	-
Order	2	-
PCrdType	4	-
MemAttr	4	-
SnpAttr	1	-
LPID	5	-
Excl SnoopMe	1	Used in Exclusive transactions Used in Atomic transactions

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 12-6 Request flit format (continued)

REQFLIT[(R-1):0] format		
Field	**Field width**	**Comments**
ExpCompAck	1	-
TraceTag	1	-
RSVDC	X = 0	No RSVDC bus
	X = 4, 12, 16, 24, 32	Permitted RSVDC bus widths
Total	R = (117 to 137) + X	-

12.8.2 Response flit

Table 12-7 shows the Response flit format in a RSP channel packet starting at bit zero,

Table 12-7 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	8	-
Opcode	4	-
RespErr	2	-
Resp	3	-
FwdState DataPull	3	Used for DCT Used in Stash transactions
DBID	8	-
PCrdType	4	-
TraceTag	1	-
Total	T = 51 to 59	-

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.8.3 Snoop flit

Table 12-8 shows the Snoop flit format in a SNP channel packet starting at bit zero.

Table 12-8 Snoop flit format

SNPFLIT[(S-1):0] format		
Field	**Field width**	**Comments**
QoS	4	-
SrcID	7 to 11	Width determined by NodeID_Width
TxnID	8	-
FwdNID	7 to 11	Width determined by NodeID_Width
FwdTxnID[7:0] {0b00, StashLPIDValid, StashLPID[4:0]} VMIDExt[7:0]	8	Used for DCT SBZ Used in Stash transactions Used in Stash transactions Used to extend VMID value
Opcode	5	-
Addr	41 to 49	Width determined by Req_Addr_Width
NS	1	-
DoNotGoToSD DoNotDataPull	1	-
RetToSrc	1	-
TraceTag	1	-
Total	S = 84 to 100	

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.8.4 Data flit

Table 12-9 shows the Data flit format in a DAT channel packet starting at bit zero.

The number of data flits required is dependent on the number of data bytes, and the data bus width. See *Data packetization* on page 2-117.

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 (DC) field width is either zero or equal to the width of the Data field divided by 8.

Poison (P) field width is either zero or equal to the width of the Data field divided by 64.

Table 12-9 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	8	-
HomeNID	7 to 11	Width determined by NodeID_Width
Opcode	4	This field was 3-bits prior to Issue C
RespErr	2	-
Resp	3	-
FwdState DataPull DataSource	3	Used for DCT Used in Stash transactions Indicates Data source in a response
DBID	8	-
CCID	2	-
DataID	2	-
TraceTag	1	-
RSVDC	Y = 0	No RSVDC bus
	Y = 4, 12, 16, 24, 32	Permitted RSVDC bus widths
BE	16, 32, 64	-
Data	128, 256, 512	-
DataCheck	0, 16, 32, 64	-
Poison	0, 2, 4, 8	-
Total	D = (202 to 214) + Y + DC + P	128 bit Data
	D = (346 to 358) + Y + DC + P	256 bit Data
	D = (634 to 646) + Y + DC + P	512 bit Data

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.9 Protocol flit fields

A Protocol flit is identified by a non-zero 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:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.9.1 TgtID

Target Identifier associated with the message. 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 *Details of transaction identifier fields* on page 2-74.

12.9.2 SrcID

Source Identifier associated with the message. 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 has been sent. See *Details of transaction identifier fields* on page 2-74.

12.9.3 HomeNID

Home identifier 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 *Details of transaction identifier fields* on page 2-74.

Applicable in CompData and DataSepResp from the Slave and Home.

Inapplicable and must be zero in all other data messages.

12.9.4 ReturnNID

Return NID. Identifies the node to which the Slave sends the CompData and DataSepResp response. The value can be either the NID of Home or the Requester that originated the transaction. See *Details of transaction identifier fields* on page 2-74.

Applicable in ReadNoSnp, ReadNoSnpSep, and non-store Atomics from Home to Slave.

Inapplicable and must be zero for all other requests. For Stash requests, the same bits in the packet are used for StashNID.

12.9.5 FwdNID

Identifies the Requester to which the CompData response can be forwarded. The value must be the NID of the Requester that initiated the transaction. See *Details of transaction identifier fields* on page 2-74.

Applicable in Forward type snoops.

Inapplicable and must be zero in all other Snoop requests.

12.9.6 LPID

Logical Processor Identifier. Used in conjunction with the SrcID to uniquely identify the logical processor that generated the request. See *Logical Processor Identifier* on page 2-95.

12.9.7 StashNID

Stash NID. Identifies the target of the stash request. Provides a valid stash target value when the corresponding StashNIDValid bit is asserted. See *Stash target identifiers* on page 7-249.

Applicable in Stash requests.

Inapplicable and must be zero for all other requests. For ReadNoSnp and ReadNoSnpSep requests, the same bits in the packet are used for ReturnNID.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.9.8 StashNIDValid

Stash NID valid. Indicates if StashNID field has a valid value. See *Stash target identifiers* on page 7-249.

Applicable in Stash requests, inapplicable in all other requests.

Table 12-10 shows the StashNIDValid value encoding.

Table 12-10 StashNIDValid value encoding

StashNIDValid	Description
0	StashNID field value is inapplicable and must be set to zero.
1	The StashNID field in the Request has a valid Stash target.

12.9.9 StashLPID

Stash Logical Processor ID. Provides a valid Logical Processor target value within the Request Node specified by StashNID. See *Stash target identifiers* on page 7-249.

Applicable in Stash requests and Stash type snoop requests.

Inapplicable and must be zero for all other requests. For ReadNoSnp requests the same bits in the packet are used for ReturnTxnID.

Inapplicable and must be zero for all other snoop requests. For Fwd 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.

12.9.10 StashLPIDValid

Stash LPID valid. Indicates if the StashLPID field has a valid value. See *Stash target identifiers* on page 7-249.

Applicable in Stash requests and Stash type snoop requests, inapplicable in all other requests.

Table 12-11 shows the StashLPIDValid value encoding.

Table 12-11 StashLPIDValid value encoding

StashLPIDValid	Description
0	StashLPID field value is inapplicable and must be set to zero.
1	The StashLPID field in the Request has a valid Stash target.

Table 12-12 shows the valid StashNIDValid and StashLPIDValid encodings.

Table 12-12 Valid StashNIDValid and StashLPIDValid encodings

StashNIDValid	StashLPIDValid	Comments
0	0	Stash target is not specified
0	1	Reserved
1	0	Only a target RN is specified
1	1	Both target RN and LPID are specified

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.9.11 TxnID

Transaction Identifier associated with the message. When there are multiple outstanding transactions from a given source node they will each use a unique transaction ID. See *Details of transaction identifier fields* on page 2-74.

Link flits do not have a unique ID. Table 12-13 shows the link flit TxnID field value encodings.

Table 12-13 Link flit TxnID Encodings

TxnID	Description
`0x00`	L-Credit return
`0x01` to `0xFF`	Reserved

12.9.12 ReturnTxnID

Return TxnID. Identifies the value the Slave must use in the TxnID field of the CompData, and DataSepResp response. It 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 *Details of transaction identifier fields* on page 2-74.

Applicable only in ReadNoSnp, ReadNoSnpSep and non-store Atomics from Home to Slave.

Inapplicable and must be zero for all other requests. For Stash requests the same bits in the packet are used for StashLPID.

12.9.13 FwdTxnID

Identifies the TxnID of the original Request associated with the Snoop transaction. See *Details of transaction identifier fields* on page 2-74.

Applicable in Forward type snoops.

Inapplicable and must be zero in all other snoop requests. For Stash snoops the same bits in the packet are used StashLPID and for SnpDVMOp snoops the same bits in the packet are used for VMIDExt.

12.9.14 DBID

Data Buffer Identifier. The DBID field value in the response packet from a Completer is used as the TxnID for CompAck or WriteData sent from the Requester. In Snoop responses with data pull, this field value indicates the value to be used in the TxnID field of data pull response messages. See *Details of transaction identifier fields* on page 2-74.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.9.15 Opcode

Specifies the operation to be carried out. The Opcode encodings are specific to each channel. See:

- *REQ channel opcodes*.
- *RSP channel opcodes* on page 12-320.
- *SNP channel opcodes* on page 12-321.
- *DAT channel opcodes* on page 12-322.

REQ channel opcodes

Table 12-14 shows the opcodes for the request channel.

Table 12-14 REQ channel opcodes

Opcode[5:0]	Request command
0x00	ReqLCrdReturn
0x01	ReadShared
0x02	ReadClean
0x03	ReadOnce
0x04	ReadNoSnp
0x05	PCrdReturn
0x06	Reserved
0x07	ReadUnique
0x08	CleanShared
0x09	CleanInvalid
0x0A	MakeInvalid
0x0B	CleanUnique
0x0C	MakeUnique
0x0D	Evict
0x0E	Reserved (EOBarrier)
0x0F	Reserved (ECBarrier)
0x10	Reserved
0x11	ReadNoSnpSep
0x12-0x13	Reserved
0x14	DVMOp
0x15	WriteEvictFull
0x16	Reserved (WriteCleanPtl)
0x17	WriteCleanFull
0x18	WriteUniquePtl
0x19	WriteUniqueFull

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 12-14 REQ channel opcodes (continued)

Opcode[5:0]	Request command
0x1A	WriteBackPtl
0x1B	WriteBackFull
0x1C	WriteNoSnpPtl
0x1D	WriteNoSnpFull
0x1E - 0x1F	Reserved
0x20	WriteUniqueFullStash
0x21	WriteUniquePtlStash
0x22	StashOnceShared
0x23	StashOnceUnique
0x24	ReadOnceCleanInvalid
0x25	ReadOnceMakeInvalid
0x26	ReadNotSharedDirty
0x27	CleanSharedPersist
0x28 - 0x2F	AtomicStore
0x30 - 0x37	AtomicLoad
0x38	AtomicSwap
0x39	AtomicCompare
0x3A	PrefetchTgt
0x3B - 0x3F	Reserved

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 12-15 shows the sub-opcodes for AtomicStore and AtomicLoad.

Table 12-15 Sub-codes 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
		101	SMIN
		110	UMAX
		111	UMIN

RSP channel opcodes

Table 12-16 shows the opcodes for the response channel.

Table 12-16 RSP channel opcodes

Opcode[3:0]	Response command
0x0	RespLCrdReturn
0x1	SnpResp
0x2	CompAck
0x3	RetryAck
0x4	Comp
0x5	CompDBIDResp
0x6	DBIDResp
0x7	PCrdGrant
0x8	ReadReceipt
0x9	SnpRespFwded
0xA	Reserved
0xB	RespSepData
0xC-0xF	Reserved

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SNP channel opcodes

Table 12-17 shows the opcodes for the snoop channel.

Table 12-17 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	Reserved
0x11	SnpSharedFwd
0x12	SnpCleanFwd
0x13	SnpOnceFwd
0x14	SnpNotSharedDirtyFwd
0x15 - 0x16	Reserved
0x17	SnpUniqueFwd
0x18 - 0x1F	Reserved

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DAT channel opcodes

Table 12-18 shows the opcodes for the data channel.

Table 12-18 DAT channel opcodes

Opcode[3:0]	Data command
0x0	DataLCrdReturn
0x1	SnpRespData
0x2	CopyBackWrData
0x3	NonCopyBackWrData
0x4	CompData
0x5	SnpRespDataPtl
0x6	SnpRespDataFwded
0x7	WriteDataCancel
0x8-0xA	Reserved
0xB	DataSepResp
0xC	NCBWrDataCompAck
0xD-0xF	Reserved

12.9.16 Addr

Address. Specifies the address associated with the message.

This specification supports a *Physical Address* (PA) of 44 to 52 bits. This permits the REQ and SNP packets to support 49 to 53 bits of *Virtual Address* (VA) in DVM operations.

- Request messages support a 44 to 52 bit address field, Addr[(43-51):0].
- Snoop messages support a 41 to 49 bit address field, Addr[(43-51):3]:
  - Addr[(43-51):6] specifies the aligned address of the 64-byte cache line.
  - Addr[5:4] indicates the 16-byte critical chunk within the cache line. See *Critical Chunk Identifier* on page 2-120.
  - Addr[3] is relevant in SnpDVMOp, for all other Snoop packets it is Don't Care and can take any value.

12.9.17 NS

Non Secure. Indicates a Non-secure access or a Secure access. See *Non-secure bit* on page 2-107.

Table 12-19 shows the NS field value encoding.

Table 12-19 NS value encoding

NS	Description
0	Secure access
1	Non-secure access

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.9.18 Size

Size. Specifies the size of the data associated with the transaction. See *Data size* on page 2-115.

Table 12-20 shows the Size field value encodings.

Table 12-20 Size field value encodings

Size[2:0]	Bytes
0b000	1
0b001	2
0b010	4
0b011	8
0b100	16
0b101	32
0b110	64
0b111	Reserved

12.9.19 MemAttr

Memory Attribute. Memory attribute associated with the transaction.

Table 12-21 shows the MemAttr value encodings.

Table 12-21 MemAttr value encodings

MemAttr[3:0]	Description
[3]	Allocate hint bit. Indicates whether or not the cache receiving the transaction is recommended to allocate the transaction: **0** Recommend that it does not allocate. **1** Recommend that it allocates.
[2]	Cacheable bit. Indicates a Cacheable transaction for which the cache, when present, must be looked up in servicing the transaction: **0** Non-cacheable. Looking up a cache is not required. **1** Cacheable. Looking up a cache is required.
[1]	Device bit. Indicates if the memory type associated with the transaction is Device or Normal: **0** Normal memory type. **1** Device memory type.
[0]	Early Write Acknowledge bit. Specifies the Early Write Acknowledge status for the transaction: **0** Early Write Acknowledge not permitted. **1** Early Write Acknowledge permitted.

See *Memory Attributes* on page 2-107.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.9.20 SnpAttr

Snoop Attribute. Specifies the snoop attribute associated with the transaction.

Table 12-22 shows the SnpAttr value encoding.

Table 12-22 SnpAttr value encoding

SnpAttr	Snoop attribute
0	Non-snoopable
1	Snoopable

See *Snoop Attribute* on page 2-113.

12.9.21 LikelyShared

Likely Shared. Indicates whether the requested data is likely to be shared with another RN. See *Likely Shared* on page 2-112.

Table 12-23 shows the LikelyShared field value encoding.

Table 12-23 LikelyShared value encoding

LikelyShared	Description
0	Not likely to be shared by another RN
1	Likely to be shared by another RN

12.9.22 Order

Specifies the ordering requirements for a transaction. See *Ordering* on page 2-96 for more information on the ordering requirements.

Table 12-24 shows the Order field value encodings.

Table 12-24 Order value encodings

Order[1:0]	Description	Note
`0b00`	No ordering required	
`0b01`	Request accepted	Applicable in Read request from HN-F to SN-F, and HN-I to SN-I.
	Reserved	Reserved in all other cases
`0b10`	Request Order/Ordered Write Observation	
`0b11`	Endpoint Order, which also includes Request Order	

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.9.23 Excl

Exclusive. Indicates that the corresponding transaction is an Exclusive type transaction. The Exclusive bit must only be used with the following transactions:

- ReadNotSharedDirty.
- ReadShared.
- ReadClean.
- CleanUnique.
- ReadNoSnp.
- WriteNoSnp.

Table 12-25 shows the Excl value encoding.

Table 12-25 Excl value encoding

Excl	Description
0	Normal transaction
1	Exclusive transaction

See *Exclusive transactions* on page 6-238.

12.9.24 Endian

Endian. Indicates the endianness of Data in an Atomic transaction. See *Endianness* on page 2-119.

Applicable in Atomic requests, inapplicable in all other requests.

Table 12-26 shows the Endian value encoding.

Table 12-26 Endian value encoding

Endian	Description
0	Little Endian
1	Big Endian

12.9.25 AllowRetry

Allow Retry. Specifies that the request is being sent without a P-Credit and that the target can determine if a retry response is given. See *Transaction Retry mechanism* on page 2-128.

Table 12-27 shows the AllowRetry value encoding.

Table 12-27 AllowRetry value encodings

AllowRetry	Description
0	RetryAck response not permitted
1	RetryAck response permitted

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.9.26 ExpCompAck

Expect CompAck. Indicates that the transaction will include a CompAck response.

Table 12-28 shows the ExpCompAck value encoding.

Table 12-28 ExpCompAck value encoding

ExpCompAck	Description
0	Transaction does not include a CompAck response
1	Transaction includes a CompAck response

12.9.27 SnoopMe

SnoopMe. Indicates that Home must determine whether to send a snoop to the Requester. See *Atomics* on page 2-62.

Only applicable in Atomic requests.

Table 12-29 shows the SnoopMe value encoding.

Table 12-29 SnoopMe value encoding

SnoopMe	Description
0	Home does not need to send a snoop to the Requester.
1	Home must send a Snoop to the Requester if it determines the cache line might be present at the Requester.

12.9.28 RetToSrc

Return to Source. Requesting Snoopee to return a copy of the cache line to Home.

Applicable in the following Fwd and non-Fwd type snoops, inapplicable to all other Snoop requests:

- SnpShared[Fwd].
- SnpClean[Fwd].
- SnpOnce[Fwd].
- SnpUnique[Fwd].
- SnpNotSharedDirty[Fwd].

For RetToSrc bit semantics see *Shared clean state return* on page 4-194.

12.9.29 DataPull

Data Pull. Indicates the inclusion of a Read request, also referred to as a Data Pull, in the Snoop response. See *Snoop requests and Data Pull* on page 7-244.

Applicable in SnpResp and SnpRespData response to a Stash request, not applicable in all other Snoop responses.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 12-30 shows the DataPull field value encodings.

Table 12-30 DataPull value encodings

DataPull[2:0]	Description	Comment
0b000	No Read	Inclusion of Data Pull in the Snoop response
0b001	Read	
0b010- 0b111	-	Reserved

12.9.30 DoNotGoToSD

Do not transition to SD state. This is an indication in Snoop requests.

DoNotGoToSD shares the SNP packet field with DoNotDataPull. See *Snoop flit* on page 12-312.

DoNotGoToSD is applicable in all Snoop requests except:
- SnpUniqueStash.
- SnpMakeInvalidStash.
- SnpStashShared.
- SnpStashUnique.
- SnpOnce.
- SnpOnceFwd.
- SnpDVMOp.

DoNotGoToSD can take any value in:
- SnpOnceFwd, SnpOnce.
- SnpCleanFwd, SnpClean.
- SnpNotSharedDirtyFwd, SnpNotSharedDirty.
- SnpSharedFwd, SnpShared.

DoNoGoToSD value must be set to 1 in:
- SnpUniqueFwd, SnpUnique.
- SnpCleanShared.
- SnpCleanInvalid.
- SnpMakeInvalid.

Table 12-31 shows the DoNotGoToSD value encoding.

Table 12-31 DoNotGoToSD value encoding

DoNotGoToSD	Description
0	Permitted to transition to SD state.
1	Transitioning to SD state is not permitted. If already in SD state, then must exit SD state in response to the snoop. The bit value can be ignored by the Snoopee if the Snoop request is SnpOnce or SnpOnceFwd.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.9.31 DoNotDataPull

Do not Data Pull. Do not combine a Read request with Snoop response.

DoNotDataPull shares the SNP packet field with DoNotGoToSD. See *Snoop flit* on page 12-312.

DoNotDataPull is applicable in:

- SnpUniqueStash.
- SnpMakeInvalidStash.
- SnpStashShared.
- SnpStashUnique.

This field is inapplicable in all other Snoop requests.

Table 12-32 shows the DoNotDataPull value encoding.

Table 12-32 DoNotDataPull value encoding

DoNotDataPull	Description
0	Data Pull with Stash snoop response is permitted but not required.
1	Must not include Data Pull with Stash snoop response.

12.9.32 QoS

Quality of Service priority level. Ascending values of QoS indicate higher priority levels. See Chapter 10 *Quality of Service* for more information.

12.9.33 PCrdType

Protocol Credit Type. Indicates the type of credit being granted or returned. See *Transaction Retry mechanism* on page 2-128.

Table 12-33 shows the PCrdType value encodings.

Table 12-33 PCrdType value encodings

PCrdType	Description
`0x0 - 0xF`	P-Credit type 0 to 15 respectively.

12.9.34 TraceTag

Trace Tag. A bit in a packet used to tag the packets associated with a transaction for tracing purposes.

Table 12-34 shows the TraceTag field value encoding.

Table 12-34 TraceTag value encoding

TraceTag	Description
0	Packet is not tagged.
1	Packet is tagged.

See Chapter 11 *Data Source and Trace Tag*.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.9.35 VMIDExt

Virtual Machine Identifier extension. It is used to extend VMID value from 8-bits to 16-bits. See *DVMOp payload on page 8-261*.

12.9.36 Resp

Response Status. The Resp field must have the same value in all data flits of a multi-flit data transfer.

Table 12-35 shows the Resp value encodings.

Table 12-35 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 target node. For completion responses, this field indicates the final state in the RN. For write data responses, this field indicates the state of the data in the RN when the data is sent.

Table 12-36 shows the valid Resp value encodings.

Table 12-36 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	
	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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 12-36 Valid Resp value encodings for different message types (continued)

Response Type	Resp[2:0]	State	Notes
Comp responses	0b000	I	Final state of the requesting RN-F.
	0b001	SC	
	0b010	UC	
	0b011	-	Reserved.
	0b100	-	
	0b101	-	
	0b110	UD_PD	Final state of the requesting RN-F. Responsibility for updating memory is passed to the Requester.
	0b111	SD_PD	
WriteData responses	0b000	I	State of the cache line at the RN-F when data is sent.
	0b001	SC	
	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 Home.
	0b111	SD_PD	

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.9.37 FwdState

Forward State. Indicates the state in the CompData sent from the Snoopee to the Requester. Applicable in SnpRespFwded and SnpRespDataFwded, inapplicable in all other Snoop responses.

Table 12-37 shows the FwdState value encodings.

Table 12-37 FwdState value encodings

FwdState[2:0]	Description	
FwdState[2]	PassDirty.	
	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.
FwdState[1:0]	Indicates the final state at the Requester. See Table 12-38.	

Table 12-38 enumerates the FwdState value encodings.

Table 12-38 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. Responsibility for updating memory is passed to the Requester.
0b111	SD_PD	

12.9.38 RespErr

Response Error. This field indicates the error status of the response. See Chapter 9 *Error Handling*.

Table 12-39 shows the RespErr value encodings.

Table 12-39 RespErr value encodings

RespErr[1:0]	Description
0b00	Normal Okay. Indicates that either: • The Normal access was successful. • The Exclusive access failed.
0b01	Exclusive Okay. Indicates that either the read or write portion of an Exclusive access was successful.
0b10	Data Error.
0b11	Non-data Error.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.9.39 Data

Data payload. This 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 *Data packetization* on page 2-117.

12.9.40 CCID

Critical Chunk Identifier. The CCID indicates the critical 128-bit chunk of the data that is being requested. See *Critical Chunk Identifier* on page 2-120.

Table 12-40 shows the CCID value encodings.

Table 12-40 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]

12.9.41 DataID

Data Identifier. The DataID indicates the relative position of the data chunk within the 512-bit cache line that is being transferred. See *Data packetization* on page 2-117.

Table 12-41 shows the DataID value encodings.

Table 12-41 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

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.9.42 BE

Byte Enable. Indicates if the byte of data corresponding to this byte enable 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. It consists of a bit for each data byte in the DAT flit. See *Byte Enables* on page 2-116.

Table 12-42 shows the BE value encodings.

Table 12-42 BE value encodings

BE	Byte enable
0	Corresponding byte of data is not valid.
1	Corresponding byte of data is valid.

12.9.43 DataCheck

Data Check. Used to supply the DataCheck bit for the corresponding byte of Data. See *Data Check* on page 9-283.

12.9.44 Poison

Indicates if the 64-bit chunk of data corresponding to a Poison bit is poisoned, that is, has an error, and must not be consumed. See *Poison* on page 9-282.

Table 12-43 shows the Poison value encodings.

Table 12-43 Poison value encodings

Poison	64-bit chunk poisoned
0	Corresponding 64-bit chunk is not poisoned.
1	Corresponding 64-bit chunk is poisoned.

12.9.45 DataSource

Data Source. Identifies the Sender of the data response. See *DataSource value assignment* on page 11-292.

Applicable in CompData response to a Read request and in SnpRespData and SnpRespDataPtl, inapplicable in all other responses.

Table 12-44 shows the DataSource value encodings.

Table 12-44 DataSource value encodings

DataSource	Description	Comment
`0b000`	DataSource is not supported	Applicable only to a non-memory component
`0b001` - `0b101`	IMPLEMENTATION DEFINED	See *Suggested DataSource values* on page 11-293
`0b110`	PrefetchTgt was useful	Indication from memory that the earlier sent prefetch was useful or memory received a prefetch.
`0b111`	PrefetchTgt was not useful	Indication from memory that the earlier sent prefetch was not useful or memory did not receive a prefetch.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.9.46 RSVDC

Reserved for customer use. Any value is valid in a Protocol flit. Propagation of this field through the interconnect is IMPLEMENTATION DEFINED.

This field is applicable to 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

12.10 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 zero. The remaining fields are not used and can be any value. See *Opcode* on page 12-318 for the link flit type encoding.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 13
Link Handshake

This chapter describes the link handshake requirements. It contains the following sections:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

13.1 Clock, and initialization

This section specifies the AMBA 5 CHI requirement for global clock and reset signals.

13.1.1 Clock

This architecture specification does not define a specific clocking microarchitecture, but it is expected that all devices, interconnects, etc. will include 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.

13.1.2 Reset

This architecture specification does not define a specific reset microarchitecture, but it is expected that all devices, interconnects, etc will 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.

13.1.3 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 it is permitted to begin driving these signals HIGH is at a rising **CLK** edge after **RESETn** is HIGH.

All other signals can be any value.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

13.2 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.

13.2.1 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 *Channel interface signals* on page 12-306 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 one. The maximum number of L-Credits that a 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 cycle it is received.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

13.3 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 might occur in the following cycle. This signaling permits local clock gating of certain registers associated with the interface.

Link Activation

Link activation and deactivation is 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

13.4 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 *Channel interface signals* on page 12-306 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 to keep the signal permanently asserted. It might do this, for example, if it is unable to determine in advance when a flit is to be sent.
- A transmitter is permitted to assert this signal when it does not have an L-Credit.
- A transmitter is permitted to assert and then deassert this signal without sending a flit.

Figure 13-1 shows an example of the use of the **FLITPEND** signal.

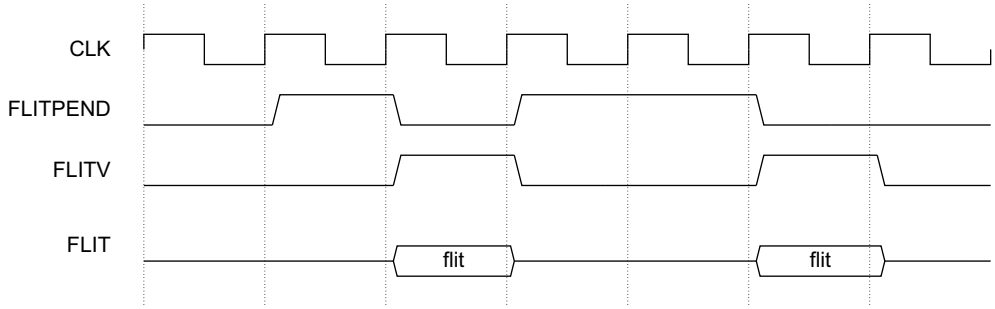


Figure 13-1 FLITPEND indicating a valid flit in next cycle

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

13.5 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. When moving between operational states, including when exiting from reset, it is important that the exchange of L-Credits, and also 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 will start 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.

13.5.1 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. *Transmit and receive link Interaction* on page 13-348 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 13-2 shows the relationship between the Payload, Credit, **LINKACTIVEREQ** and **LINKACTIVEACK** signals.

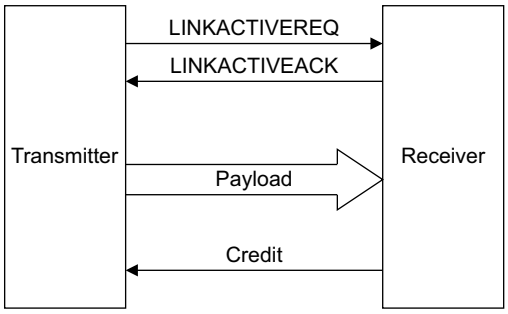


Figure 13-2 Relationship between Payload, Credit and LINKACTIVE signals

As Figure 13-2 shows, during normal operation the transmitter, which sends the payload flits, requires a credit before it can send a flit. A credit is passed from the receiver when it has resources 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 that it was at immediately after reset.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 it 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 and when one of these states is entered a channel can remain in this state for an indefinite period of time.

DEACTIVATE and ACTIVATE are transient states and it is expected that when one of these states is entered a channel will move to the next stable state in a relatively short period of time.

Note

The specification does not define a maximum period of time 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 13-3 shows the relationship between the four states.

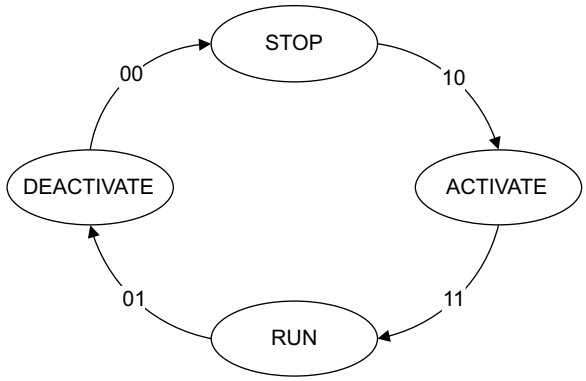


Figure 13-3 Request and Acknowledge handshake states

Table 13-1 shows the mapping of the states to the **LINKACTIVEREQ** and **LINKACTIVEACK** signals.

Table 13-1 Mapping of states to the LINKACTIVE signals

	LINKACTIVEREQ	LINKACTIVEACK
STOP	0	0
ACTIVATE	1	0
RUN	1	1
DEACTIVATE	0	1

Table 13-2 on page 13-344 describes the behavior of both the transmitter and the receiver of a single link for each of the four states.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 13-2 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 transmitter is guaranteed not to receive any credits. The transmitter must assert **LINKACTIVEREQ** to move to the ACTIVATE state if it has flits to send.	The receiver is guaranteed not to receive any flits. The receiver must not send any credits.
ACTIVATE (ACT)	The transmitter must not send any flits. The transmitter must be prepared to receive credits in this state, although it must not use them until in the RUN state. The transmitter remains in the ACTIVATE state while it is waiting for the receiver to acknowledge the move to the RUN state. **Note** The transmitter will only receive 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 receiver is guaranteed not to receive any flits. The receiver must not send any credits. 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 to assert **LINKACTIVEACK** and send a credit in the same cycle. **Note** It can appear that a receiver has 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 transmitter can send flits when it has credits available. The transmitter deasserts **LINKACTIVEREQ** to exit from this state if it wants to move to a low power state.	The receiver can receive flits corresponding to the credits it has sent. The receiver sends credits when it has resources available to accept further flits. The receiver must remain in the RUN state until it observes the deassertion of **LINKACTIVEREQ**.
DEACTIVATE (DEACT)	The transmitter must return credits using Protocol flits or L-Credit return flits. It is recommended that the transmitter enters the DEACTIVATE state only when it has no more Protocol flits to send. Therefore, it is expected that the transmitter will return credits using only L-Credit return flits. The transmitter must be prepared to continue receiving credits. For each additional credit received it must send an L-Credit return flit to return the credit. The transmitter remains in the DEACTIVATE state while it is waiting for the receiver to acknowledge the move to the STOP state. At this point, it will be guaranteed to receive no more credits.	During this state the receiver stops sending credits and collects all returned credits. 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 receiver is permitted to send credits when first entering this state. However, it must have stopped sending credits and had all credits returned before exiting this state. The receiver will receive 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 will only receive 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 13-5 on page 13-352 summarizes the required behavior described in detail in Table 13-2 on page 13-344.

Table 13-3 Summary of behavior for each Request and Acknowledge state

	Transmitter	Receiver
STOP	Must not send flits. Will not receive credits.	Must not send credits. Will not receive flits.
ACT	Must not send flits. Must accept credits.	Must not send credits. Will not receive flits.
RUN	Can send flits. Must accept credits.	Must accept flits. Can send credits.
DEACT	Must not send flits, except for credit return flits. Must accept credits. Must return credits.	Must accept flits. Must stop sending credits.

Race conditions

There are two situations where one side of the interface performs two actions at or around the same time:

- Changing the **LINKACTIVEREQ** or **LINKACTIVEACK** signal to change the state of the interface.
- Sending an associated credit or flit around the time of the state change.

This occurs in the following situations:

- When the receiver is asserting **LINKACTIVEACK**, to move from ACTIVATE to RUN, it is also permitted to start sending credits:
  - A race can occur between the sending of a credit, which is expected in the new state, and the assertion of the **LINKACTIVEACK** signal indicating the state change.
  - This is acceptable because the transmitter is required to be able to accept the credit in the previous state as well as in the new state.
  - For the receiver, it is permitted to send a credit in the same cycle that **LINKACTIVEACK** is asserted.
  - For the transmitter, it is required to accept a credit both before and after the assertion of **LINKACTIVEACK**.

- When the transmitter is deasserting **LINKACTIVEREQ**, to move from RUN to DEACTIVATE, it must stop sending flits, other than L-Credit return flits:
  - A race can occur between the last flit sent, which is expected in the previous state, and the deassertion of the **LINKACTIVEREQ** signal indicating the state change.
  - This is acceptable because the receiver is required to be able to accept the flit in the next state, as well as in the previous state.
  - For the transmitter, it is permitted to send a flit in the last cycle that **LINKACTIVEREQ** is asserted.
  - For the receiver, it is required to accept flits both before and after the deassertion of **LINKACTIVEREQ**.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 might be required to change its behavior.

If the state change requires a component to start sending flits or credits, then there is no defined limit on the time taken for the component to start the new behavior. This new behavior will only occur in the new state.

If the state change requires a component to stop sending flits or credits, then the component is permitted to take some time to respond. In this scenario, it is possible to see behavior 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. However, a race condition might still occur as described in *Race conditions* on page 13-345.

Credits are sent by the receiver, but that component does not determine the state change. The receiver might take some time to react to the state change and therefore it is possible for credits to be sent when first entering the DEACTIVATE state.

The protocol requires that the receiver has stopped sending credits and has had all credits returned before it signals the change from DEACTIVATE to STOP.

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 a number of mechanisms. The following examples are not exhaustive:

- The transmitter can determine that it has flits to send, so must move from STOP to RUN.
- The transmitter can determine that it has no activity to perform for a significant period of time, so can move from RUN to STOP.
- The transmitter can observe an independent sideband signal that indicates it should 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 *Transmit and receive link Interaction* on page 13-348.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Multiple channels in the same direction

Figure 13-4 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.

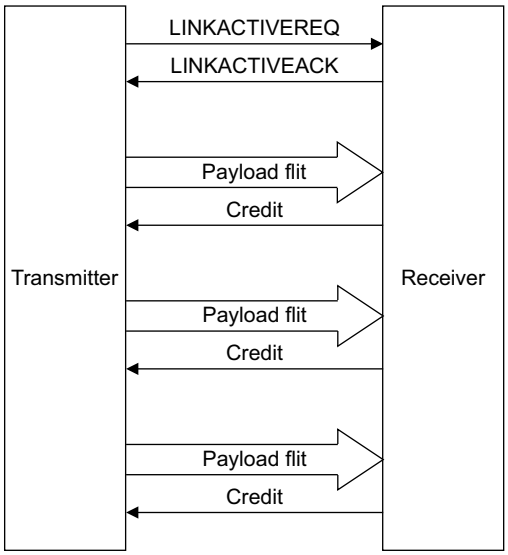


Figure 13-4 Example of a multiple channel unidirectional interface

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 it must be able to accept credits on all channels.
- When a state change requires the receiver to be able to accept flits it must be able to accept flits 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 flits must stop on all channels.
- A credit can only be associated with a flit on the same channel.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

13.6 Transmit and receive link Interaction

This section describes the interaction between a link transmitter and receiver. It contains the following subsections:

- *Introduction*.
- *Tx and Rx state machines* on page 13-349.
- *Expected transitions* on page 13-351.

13.6.1 Introduction

A single component has a number of 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).

This specification requires 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, then 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 it controls, then it can expect the RXLINK 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, then 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 it controls, then it can expect the RXLINK 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

13.6.2 Tx and Rx state machines

Figure 13-5 shows the permitted relationships between the Tx and Rx state machines. It is formatted so that the independent nature of the Tx and Rx state machines can be seen.

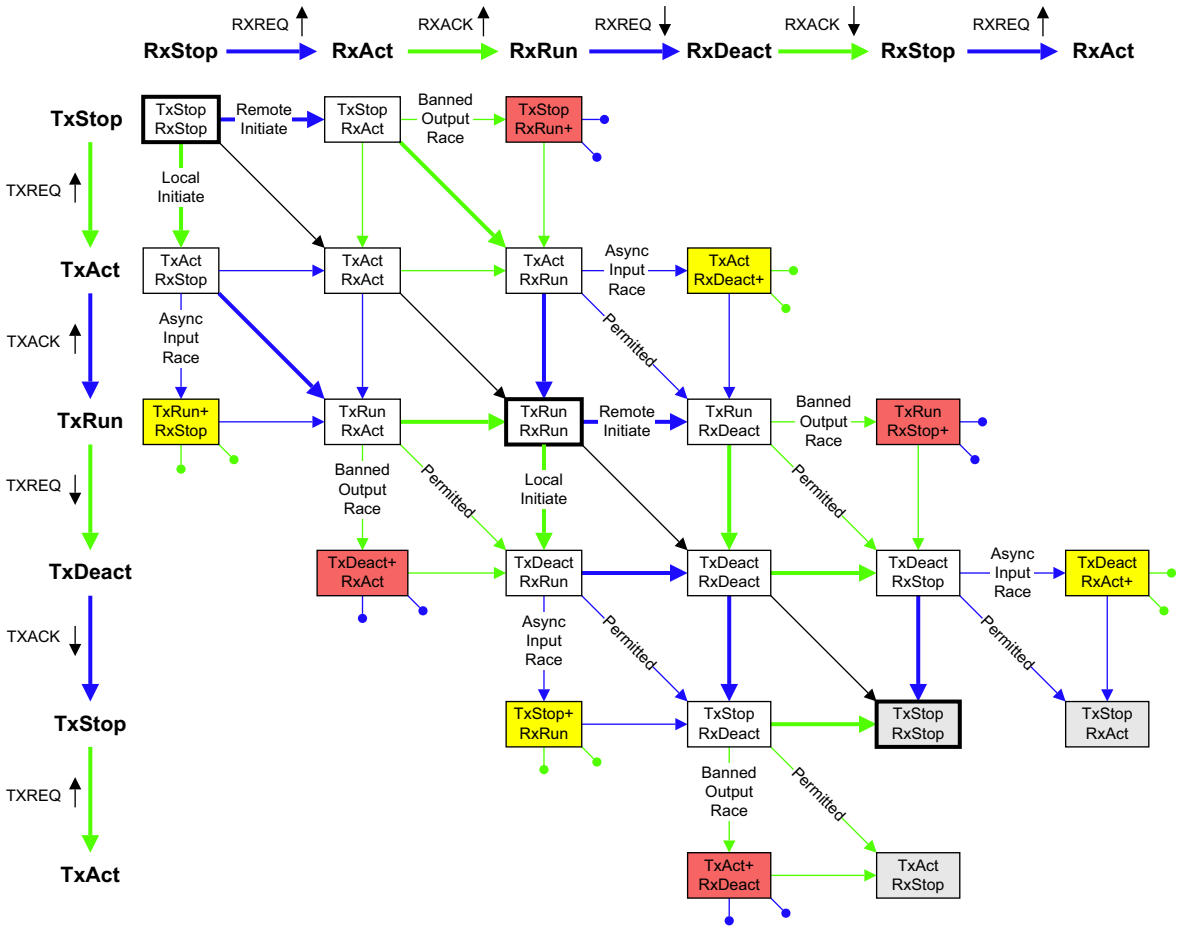


Figure 13-5 Combined Tx and Rx state machines

Figure 13-5 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 13-5 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 then the same agent is changing both signals.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- 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 grey states, on the bottom right of Figure 13-5 on page 13-349, 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 *Asynchronous race condition* on page 13-352.
- 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 *Asynchronous race condition* on page 13-352.
- The bold arrows are used to indicate the expected transitions around the state machine. These are described in more detail in *Expected transitions* on page 13-351.
- The arrows labeled *Permitted* are state transactions that would not normally be expected, but they are permitted by the protocol.

State naming

Figure 13-5 on page 13-349 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 *Asynchronous race condition* on page 13-352.

There are two different TxStop/RxRun states, and two different TxRun/RxStop states. These states differ in how they are reached and how it is permitted to exit from them. To differentiate between these states, a [+] suffix is used to indicate which state machine, that is, 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

13.6.3 Expected transitions

Figure 13-6 shows the expected state transitions.

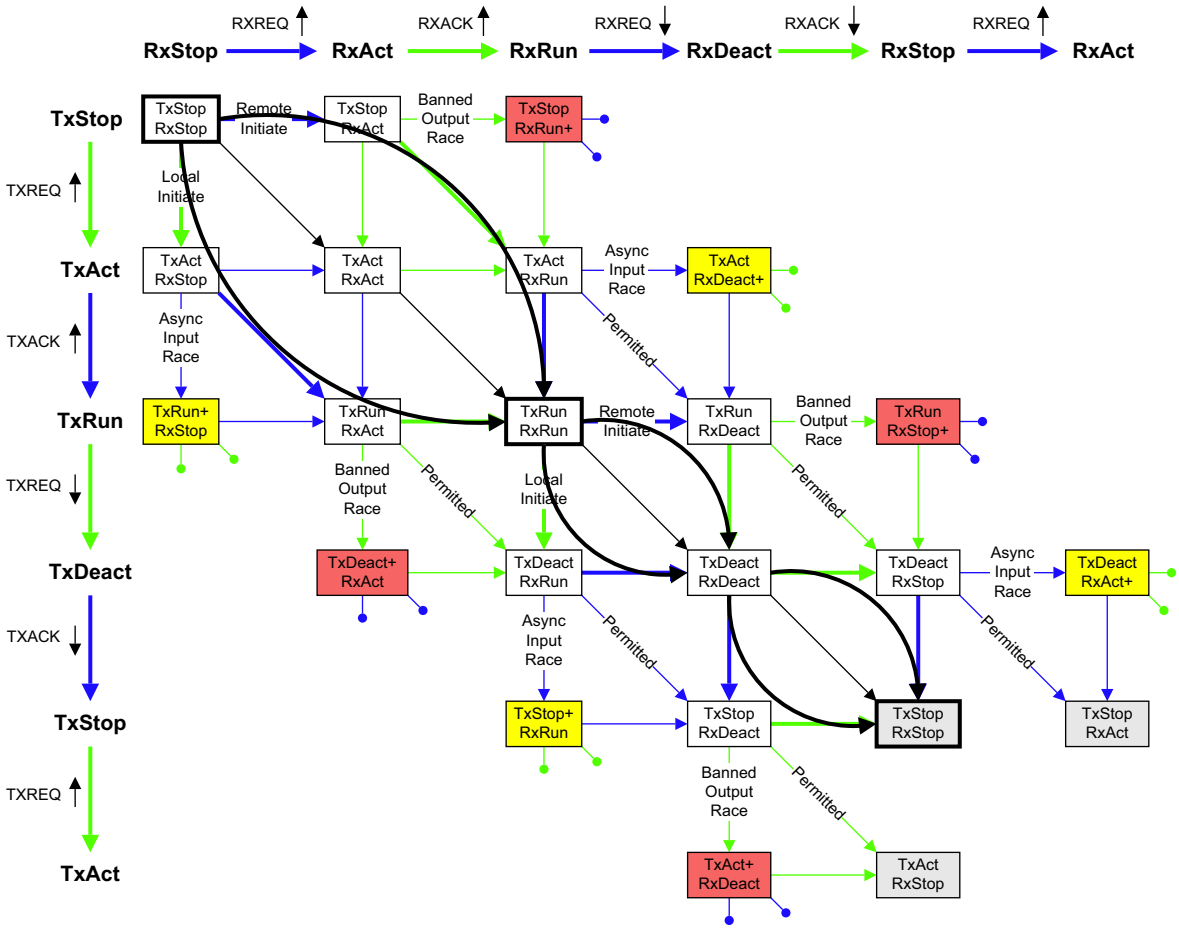


Figure 13-6 Expected Tx and Rx state machines transitions

Figure 13-6 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 *Link layer Credits* (L-Credits) in the latter case. The differences are detailed in the following sections.

Expected transitions from TxStop/RxStop to TxRun/RxRun

There are two expected routes from a stable Stop/Stop to Run/Run state. Table 13-4 shows, in terms of the state transitions, the two expected paths.

Table 13-4 Stop/Stop to Run/Run state paths

	State 1	State 2	State 3	State 4
Path 1	TxStop/RxStop	TxStop/RxAct	TxAct/RxRun	TxRun/RxRun
Path 2	TxStop/RxStop	TxAct/RxStop	TxRun/RxAct	TxRun/RxRun

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The annotations on the diagram arrows in Figure 13-6 on page 13-351 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.

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 13-5 shows, in terms of the state transitions, the four expected paths.

Table 13-5 State 5

	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

Transitioning 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. It might be that the low power state was entered prematurely, or it might be that 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.

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 13-5 on page 13-349, 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 13-5 on page 13-349 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.

Combined Tx and Rx state machines without race conditions

In Figure 13-7 all transitions and states that occur as a result of a race condition in the combined Tx and Rx state machines have been removed.

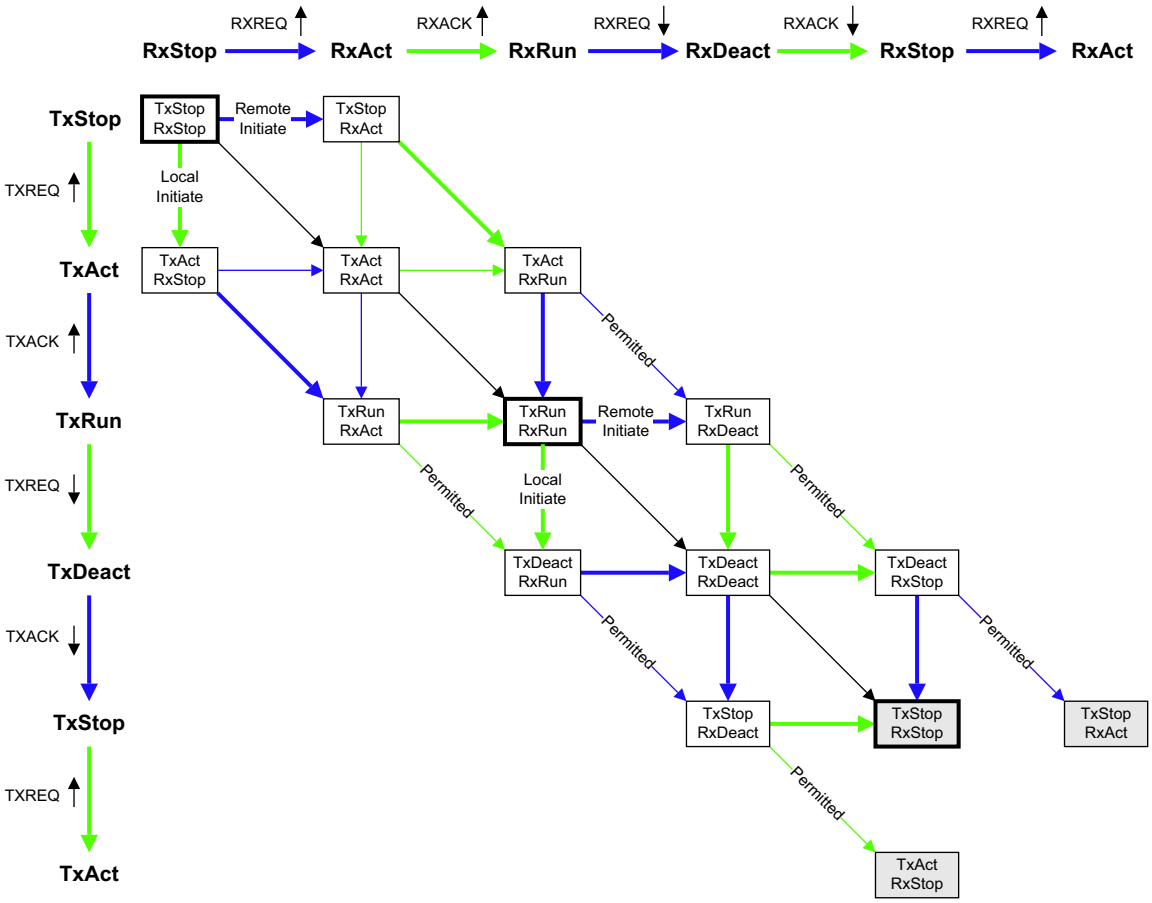


Figure 13-7 Combined Tx and Rx state machines without race conditions

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

13.7 Protocol layer activity indication

This section describes the signals that indicate Protocol layer activity. It contains the following subsections:

- *Introduction*.
- *TXSACTIVE signal*.
- *RXSACTIVE signal* on page 13-357.
- *Relationship between SACTIVE and LINKACTIVE* on page 13-357.

13.7.1 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, so **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.

13.7.2 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 then 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 for **TXSACTIVE** to be deasserted while transmitting link flits as part of the link deactivation sequence.

Note

To ensure an efficient power-down sequence, ARM recommends not to assert a deasserted TXSACTIVE signal during a link deactivation sequence.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Figure 13-8 shows the requirements for **TXSACTIVE** assertion during the life of a transaction.

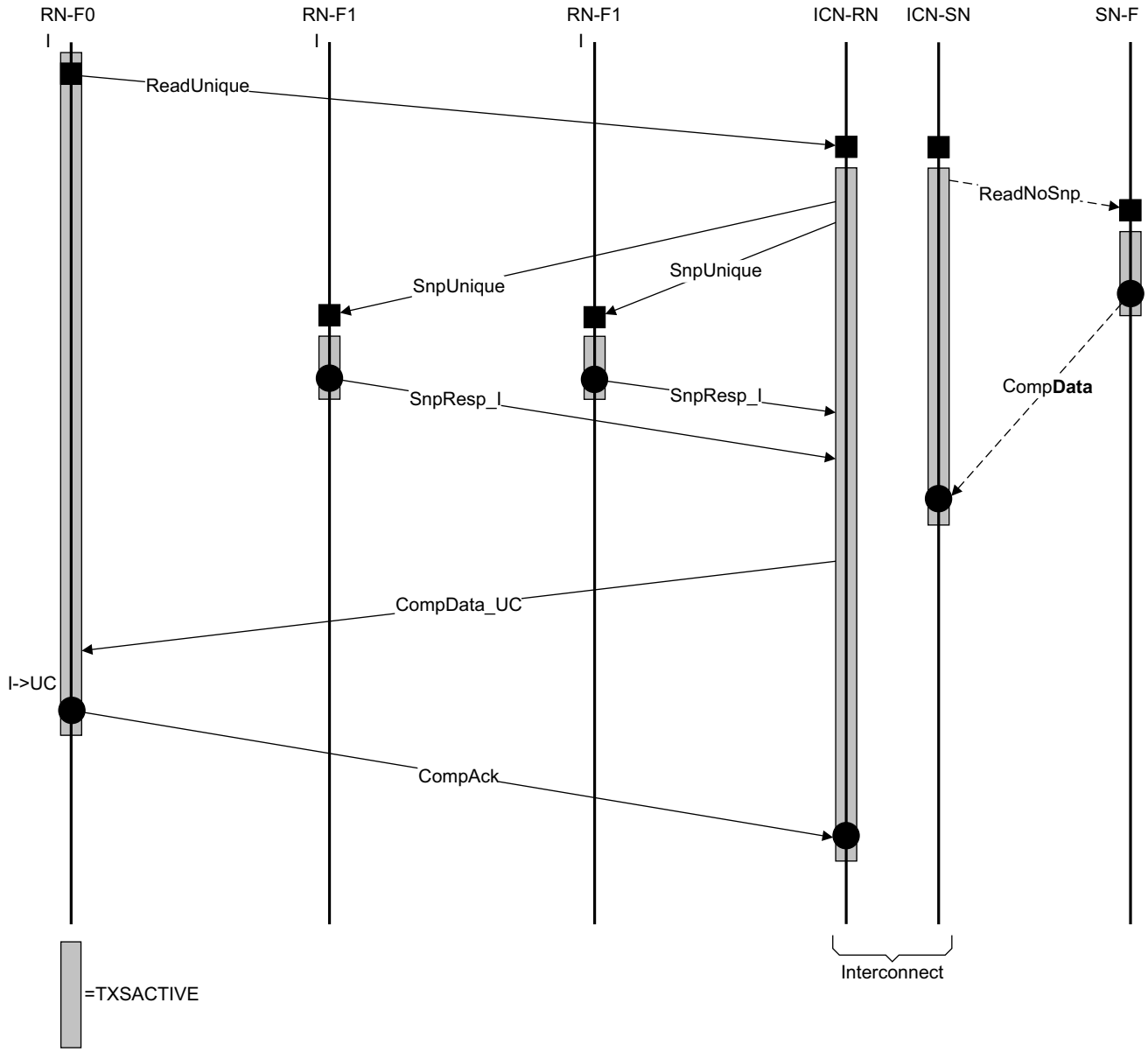


Figure 13-8 TXSACTIVE assertion during the life of a transaction

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TXSACTIVE signaling from an RN

When initiating new transactions, an RN must assert **TXSACTIVE** in the same cycle or before **TXREQFLITV** is asserted and must keep it asserted until after the final completing flit of a transaction is sent or received.

The type of flit that completes a transaction initiated by an RN will depend on both the transaction type and the manner in which the transaction progresses. For example, a ReadNoSnp transaction might 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.

Table 13-6 shows the flit types that can complete a transaction. The PrefetchTgt transaction does not include an explicit completion message, the transaction is considered completed the cycle after it is sent.

Table 13-6 RN completing flits for RN initiated transactions

Completing flit	Channel	Transaction type
CompAck	TXRSP	Read, Dataless, WriteUnique
Comp	RXRSP	Dataless, WriteUnique, WriteNoSnp, AtomicStore, DVM
CompData	RXDAT	Read, Atomic
RespSepData	RXRSP	Read
DataSepResp	RXDAT	Read
ReadReceipt	RXRSP	ReadNoSnp, ReadOnce
PCrdReturn	TXREQ	All transaction types
NonCopyBackWrData	TXDAT	WriteUnique, WriteNoSnp, Atomic
NCBWrDataCompAck	TXDAT	WriteUnique
CopyBackWrData	TXDAT	CopyBack

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. It must keep **TXSACTIVE** asserted until after the final completing flit is sent for all Snoop transactions.

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.

TXSACTIVE signaling from an SN

An SN cannot initiate new transactions and is only required to assert **TXSACTIVE** while it is processing a transaction that is in progress.

It must assert **TXSACTIVE** after receiving a transaction initiating flit and it must be asserted before or in the same cycle in which its first Response flit is sent. It must keep **TXSACTIVE** asserted until after the final completing flit is sent or received.

TXSACTIVE signaling from an ICN interface to an RN

The interconnect interface to an RN must assert **TXSACTIVE** in both the following conditions:

- On receiving a transaction initiating flit, it must be asserted before or in the same cycle in which its first Response flit is sent. It must keep **TXSACTIVE** asserted until after the final completing flit is sent or received.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Before or in the same cycle in which its initiating Snoop or SnpDVMOp flit is sent. It must keep **TXSACTIVE** asserted until after the final completing flit is sent, which will be either SnpResp or SnpRespData.

TXSACTIVE signaling from an ICN interface to an SN

The interconnect interface to an SN must assert **TXSACTIVE** before, or in the same cycle in which its initiating Request flit is sent. It must keep **TXSACTIVE** asserted until after the final completing flit is sent or received.

13.7.3 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. It is possible for a receiver to 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 will be 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.

13.7.4 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 receiver 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 *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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 14
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:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

14.1 Overview

The system coherency interface signals are:

SYSCOREQ Master coherency request.

SYSCOACK Interconnect coherency acknowledge.

Figure 14-1 shows the system coherency interface signals connections.

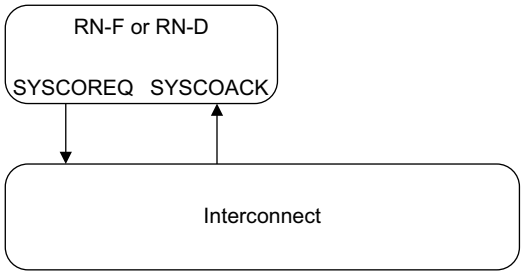


Figure 14-1 System coherency interface signals

Note

In this chapter:

- Coherency when stated, includes the DVM domain, unless explicitly stated otherwise.
- Snoop when stated, includes SnpDVMOp, unless explicitly stated otherwise.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

14.2 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 14-2 shows the system coherency interface handshake timing.

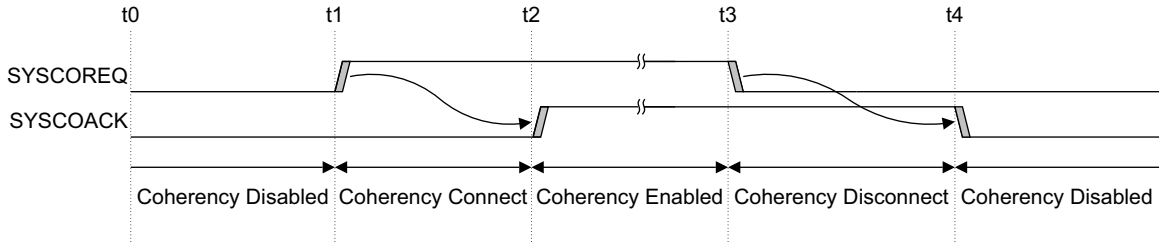


Figure 14-2 System coherency interface handshake timing

As Figure 14-2 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.

14.2.1 RN rules

Referring to Figure 14-2, an RN must:

- Be able to service Snoop requests when it sets **SYSCOREQ** HIGH at **t1**.
- Not issue a transaction that permits it to cache a coherent location until **SYSCOACK** goes HIGH at **t2**.
- Ensure all transactions that permit it to cache a coherent location are complete before it sets **SYSCOREQ** LOW at **t3**.

  SYSCOREQ can only be deasserted on the cycle after all of the following:
  - All data packets are received for Reads.
  - All data packets are sent for CopyBack.
  - 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 connect transition periods to guarantee the **SYSCOACK** transition will occur. See *Protocol layer activity indication* on page 13-354.

Note

The transactions that permit a coherent location to be cached are:

- ReadUnique.
- ReadClean.
- ReadNotSharedDirty.
- ReadShared.
- CleanUnique.
- MakeUnique.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

14.2.2 Interconnect rules

Referring to Figure 14-2 on page 14-361, the interconnect must:

- When it samples **SYSCOREQ** HIGH, be able to service coherent data accesses from the interface when it sets **SYSCOACK** HIGH at **t2**.
- When it samples **SYSCOREQ** LOW, it must complete all snoop accesses to the interface before it sets **SYSCOACK** LOW at **t4**.

14.2.3 Protocol states

Table 14-1 shows the interface states and the rules that the master must follow in relation to the interface state.

Table 14-1 System coherency interface states

State name	SYSCOREQ	SYSCOACK	Description
Coherency Disabled	0	0	• RN caches must not contain coherent data. • RN must not issue transactions that permit it to cache a coherent location or send DVM transactions. • RN not required to respond to Snoop requests. • Interconnect must not send Snoop requests.
Coherency Connect	1	0	• RN caches must not contain coherent data. • RN must not issue transactions that permit it to cache a coherent location or send DVM transactions. • RN must respond to Snoop requests. • Interconnect can send Snoop requests.
Coherency Enabled	1	1	• RN caches can contain coherent data. • RN can issue transactions that cache a coherent location and send DVM transactions. • RN must respond to Snoop requests.
Coherency Disconnect	0	1	• RN caches must not contain coherent data. • RN must not issue transactions that permit it to cache a coherent location or send DVM transactions. • RN must respond to Snoop requests. • Interconnect must complete outstanding Snoop requests but must not generate new Snoop requests.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter 15
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:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15.1 Interface properties and parameters

A property is used to declare a capability. If a property is not declared, it is considered False.

The properties and parameters that specify the interface behavior are:

Atomic_Transactions

An Atomic_Transactions property is used to indicate if a component supports Atomic transactions:

- When not specified, or set to False, Atomic transactions are not supported.
- When set to True, Atomic transactions are 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.

Cache_Stash_Transactions

A Cache_Stash_Transactions property is used to indicate if a component supports Cache Stashing transactions:

- When not specified, or set to False, Cache Stashing transactions are not supported.
- When set to True, Cache Stashing Transactions are supported.

Direct_Memory_Transfer

A Direct_Memory_Transfer property is used to indicate if a component supports Direct Memory Transfer transactions:

- When not specified, or set to False, Direct Memory Transfer transactions are not supported.
- When set to True, Direct Memory Transfer transactions are supported.
- The Direct_Memory_Transfer property is defined at each HN for each SN.

Direct_Cache_Transfer

A Direct_Cache_Transfer property is used to indicate if a component supports Direct Cache Transfer transactions:

- When not specified, or set to False, Direct Cache Transfer transactions are not supported.
- When set to True, Direct Cache Transfer transactions are supported.
- It is the responsibility of the HN-F to determine the correct snoop type to use.

Data_Poison A Data_Poison property is used to indicate if a component supports Poison:

- When not specified, or set to false, Poison is not supported and the Poison field is not present in the DAT packet.
- When set to True, Poison is supported and the Poison field is present in the DAT packet.

See *Poison* on page 9-282.

Data_Check The Data Check property is used to indicate if Data Check is supported:

- When not specified, or set to false, Data Check is not supported and the DataCheck field is not present in the DAT packet.
- When set to Odd_Parity, Data Check is supported and the DataCheck field is present in the DAT packet.

See *Data Check* on page 9-283.

CCF_Wrap_Order

See *Critical chunk first wrap order* on page 2-120.

Req_Addr_Width

This parameter specifies the maximum physical address supported by a component:

- Legal values for this parameter are 44 to 52.
- When Req_Addr_Width is not specified, the default value is 44.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NodeID_Width

This parameter specifies the width of NodeID fields supported by a component, which determines the maximum permitted NodeID value in the system:

- The width specified is uniformly applied to all NodeID related fields.
- Legal values of NodeID_Width are 7 to 11.
- When NodeID_Width is not specified, the default value is 7.

Data_Width This parameter specifies the data width in the DAT channel packet supported by a component:

- Legal values for Data_Width are 128, 256, and 512.
- When Data_Width is not specified, the default value is 128.

Enhanced_Features

The Enhanced_Features property describes the combined support for some of the miscellaneous features in the CHI specification that do not have an explicit property or parameter defined.

When the Enhanced_Features property is True, the component supports all the following enhanced features:

- Data return from SC state.
- I/O Deallocation transactions.
- ReadNotSharedDirty transaction.
- CleanSharedPersist transaction.
- Receiving of Forwarding snoops.

When not specified, or set to False, the component does not support the enhanced features that do not have an explicitly defined property or parameter.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15.2 Optional interface broadcast signals

This specification includes three sets of optional pins to determine broadcasting of certain groups of transactions in the interconnect. These pins are optional at the RN to ICN and ICN to SN interfaces. The three sets of optional broadcast pins are:

- **BROADCASTINNER** and **BROADCASTOUTER**.
- **BROADCASTCACHEMAINTENANCE** and **BROADCASTPERSIST**.
- **BROADCASTATOMIC**.

An implementation that includes these signals at the interface must ensure that the signal values are stable when Reset is deasserted.

BROADCASTINNER and **BROADCASTOUTER** determine respectively if inner and outer domain transactions must be broadcast. This specification requires that these two pins must be set to the same value. When set to zero none of the inner and outer transactions are broadcast except for *Cache Maintenance Operations* (CMO).

The **BROADCASTCACHEMAINTENANCE** and **BROADCASTPERSIST** interface signals provide efficient maintenance of downstream caches in the interconnect space. The broadcast signals are used as follows:

BROADCASTCACHEMAINTENANCE

- When asserted, CMO transactions must be broadcast beyond the interface for maintenance of downstream caches.

  CleanSharedPersist must be converted to CleanShared before broadcasting to downstream caches if **BROADCASTPERSIST** is deasserted.
- When deasserted, broadcasting of Persistent CMO, that is CleanSharedPersist, beyond the interface is determined by the assertion of the **BROADCASTPERSIST** signal.

BROADCASTPERSIST

- When asserted, CleanSharedPersist must be broadcast beyond the interface for maintenance of downstream caches. This requirement is independent of the **BROADCASTCACHEMAINTENANCE** signal value.
- When deasserted, broadcasting of the Persistent CMO beyond the interface is determined by the **BROADCASTCACHEMAINTENANCE** signal value.

The direction of the signal at the RN to ICN interface is input to RN and at the ICN to SN interface it is input to ICN.

Table 15-1 on page 15-367 shows the broadcast signal encodings using the following keys:

BI **BROADCASTINNER**.
BO **BROADCASTOUTER**.
BCM **BROADCASTCACHEMAINTENANCE**.
BP **BROADCASTPERSIST**.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table 15-1 CMO broadcast at the interface with unspecified BI and BO

Broadcast signal			Transaction to be broadcast
BI = BO	**BCM**	**BP**	
0	0	0	None.
0	0	1	Persistent CMO only.
0	1	0	Both Non-persistent and Persistent CMO. Persistent CMO is converted to Non-persistent CMO.
0	1	1	Both Non-persistent and Persistent CMO.
1	0	0	All inner and outer transactions. Persistent CMO is converted to Non-persistent CMO
1	0	1	All inner and outer transactions. All Persistent CMO.
1	1	0	All inner and outer transactions including all Non-persistent and Persistent CMO. Persistent CMO is converted to Non-persistent CMO.
1	1	1	All inner and outer transactions including all Non-persistent and Persistent CMO.

BROADCASTATOMIC

- When asserted, the interface is permitted to generate Atomic transactions.
- When deasserted, the interface must not generate Atomic transactions.

An RN is not required to make use of Atomic transactions. An RN that does not make use of Atomic transactions itself, needs no added functionality to be compatible with an interconnect that supports Atomic transactions.

An RN that supports atomic operations but does not include support for the execution of atomic operations must be able to send Atomic transactions.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15.3 Atomic transaction support

The CHI component support requirements for Atomic transactions are described in the following sections:

- *Request Node support*.
- *Interconnect support* on page 15-369.
- *Slave Node support* on page 15-369.

15.3.1 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.

An RN is not required to make use of Atomic transactions. An RN that does not make use of Atomic transactions itself, needs no added functionality to be compatible with an interconnect that supports Atomic transactions.

An RN that supports atomic operations but does not include support for the execution of atomic operations must be able to send Atomic transactions.

For an RN 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, an RN 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 it would for a store operation, and then 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, then the cache line must be marked as Dirty.

A Requester with a cache can handle an Atomic transaction request to a Snoopable memory region as follows:

- If the cache line is Unique, then it can perform the atomic operation locally without generating an Atomic transaction.
- If the cache line is Shared but not Dirty, it can either:
  - Generate a ReadUnique or CleanUnique 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.
- If the cache line is Shared Dirty, it can either:
  - Generate a CleanUnique or ReadUnique, gain ownership of the cache line, and perform the operation locally.
  - WriteBack and Invalidate the local copy and then send the Atomic transaction to the interconnect.
- Optionally, in all the above cases, the Requester is permitted 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 required, extract the cached copy. See *SnoopMe* on page 12-326.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15.3.2 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 RNs must be configured to not generate Atomic transactions. The **BROADCASTATOMIC** pin can be used for this purpose, when implemented. See *Request Node support* on page 15-368.

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 Slave Node.

Atomic transactions are not required to be supported for every address location.

If Atomic transactions are supported for a given Snoopable address location, then they must be supported for the complete Snoopable address range.

If Atomic transactions are not supported for a given address location, then an appropriate error response can be given for the Atomic transaction. See *Atomic transactions* on page 9-277.

For transactions to a Device the Atomic transaction must be passed to the appropriate endpoint slave. If the slave is configured to indicate that it does not support Atomic transactions, then 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 slave is configured to indicate that it supports Atomic transactions, then the interconnect can pass the Atomic transaction to the slave.
  - The interconnect is still required to perform the appropriate Snoop and Write transactions before issuing the Atomic transaction to the Slave.

15.3.3 Slave Node support

Slave Node support for Atomic transactions is optional.

The Atomic_Transaction property is used to indicate that an SN supports Atomic transactions.

If an SN supports Atomic transactions for particular memory types, or for particular address regions, then on receiving an Atomic transaction that it does not support, the SN must give an appropriate Error response.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Appendix A
Message Field Mappings

This appendix shows the field mappings for the request, response, data, and snoop request messages. It contains the following sections:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table A-1 shows the conventions used in the field mapping tables.

Table A-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	Field value is not defined and can be any value.
1	Applicable. Field value is used, must be set to one.
0	Applicable. Field value is used, must be set to zero.
0[a]	Inapplicable. Field value must be set to zero.
Y	Applicable. Field value is used. See specification for permitted values and usage.
8B	Size field must be set to 8-byte encoding.
64B	Size field must be set to 64-byte encoding.
-	Assigned to another protocol message field that shares the same set of bits in this packet field.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

A.1 Request message field mappings

Table A-2 shows the request message field mappings. See Table A-1 on page A-372 for the conventions used in the field mappings. CF indicates a Common Field. For further information on field use see *Protocol flit fields* on page 12-314.

Table A-2 Request vc message field mappings

Request message	Qos	TgtID	SrcID	TxnID	Opcode	Size	Addr	NS	LikelyShared	AllowRetry	Order	PCrdType	MemAttr: Allocate	MemAttr: Cacheable	MemAttr: Device	MemAttr: EWA	SnpAttr	LPID	CF: Excl	CF: SnoopMe	ExpCompAck	RSVDC	CF: ReturnNID	CF: StashNID	CF: StashNIDValid	CF: Endian	CF: ReturnTxnID	CF: StashLPIDValid	CF: StashLPID	TraceTag
ReqLCrdReturn	X	X	X	0	Y	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X
PrefetchTgt	Y	Y	Y	X	Y	X	Y	Y	X	0	X	X	X	X	X	X	X	Y	X	-	0[a]	Y	0[a]		X		0[a]			Y
PCrdReturn	Y	Y	Y	0[a]	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	Y	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]		0	Y	0[a]		0[a]		0[a]			Y
DVMOp	Y	Y	Y	Y	Y	8B	Y	0[a]	0[a]	Y	0[a]	Y	0[a]	0[a]	0[a]	0[a]	0[a]	Y	0[a]		0	Y	0[a]		0[a]		0[a]			Y
ReadNoSnp	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	-	Y	Y	Y	-	0[a]		Y	-	-	Y
ReadNoSnpSep	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	Y	0	Y	0	-	0	Y	Y	-	0[a]		Y	-	-	Y
ReadShared	Y	Y	Y	Y	Y	64B	Y	Y	Y	Y	0	Y	Y	1	0	1	1	Y	Y	-	1	Y	0[a]		0[a]		0[a]			Y
ReadClean	Y	Y	Y	Y	Y	64B	Y	Y	Y	Y	0	Y	Y	1	0	1	1	Y	Y	-	1	Y	0[a]		0[a]		0[a]			Y
ReadOnce	Y	Y	Y	Y	Y	64B	Y	Y	0	Y	Y	Y	Y	1	0	1	1	Y	0	-	Y	Y	0[a]		0[a]		0[a]			Y
ReadUnique	Y	Y	Y	Y	Y	64B	Y	Y	0	Y	0	Y	Y	1	0	1	1	Y	0	-	1	Y	0[a]		0[a]		0[a]			Y
ReadNotSD	Y	Y	Y	Y	Y	64B	Y	Y	Y	Y	0	Y	Y	1	0	1	1	Y	Y	-	1	Y	0[a]		0[a]		0[a]			Y
CleanShared	Y	Y	Y	Y	Y	64B	Y	Y	0	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	-	0	Y	0[a]		0[a]		0[a]			Y
CleanSharedPersist	Y	Y	Y	Y	Y	64B	Y	Y	0	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	-	0	Y	0[a]		0[a]		0[a]			Y
CleanInvalid	Y	Y	Y	Y	Y	64B	Y	Y	0	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	-	0	Y	0[a]		0[a]		0[a]			Y
MakeInvalid	Y	Y	Y	Y	Y	64B	Y	Y	0	Y	Y	Y	Y	Y	Y	Y	Y	Y	0	-	0	Y	0[a]		0[a]		0[a]			Y
ReadOnceCleanInvalid	Y	Y	Y	Y	Y	64B	Y	Y	0	Y	Y	Y	Y	1	0	1	1	Y	0	-	Y	Y	0[a]		0[a]		0[a]			Y
ReadOnceMakeInvalid	Y	Y	Y	Y	Y	64B	Y	Y	0	Y	Y	Y	0	1	0	1	1	Y	0	-	Y	Y	0[a]		0[a]		0[a]			Y
CleanUnique	Y	Y	Y	Y	Y	64B	Y	Y	0	Y	0	Y	Y	1	0	1	1	Y	Y	-	1	Y	0[a]		0[a]		0[a]			Y
MakeUnique	Y	Y	Y	Y	Y	64B	Y	Y	0	Y	0	Y	Y	1	0	1	1	Y	0	-	1	Y	0[a]		0[a]		0[a]			Y
Evict	Y	Y	Y	Y	Y	64B	Y	Y	0	Y	0	Y	0	1	0	1	1	Y	0	-	0	Y	0[a]		0[a]		0[a]			Y
WriteNoSnpPtl	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	-	0	Y	0[a]		0[a]		0[a]			Y
WriteNoSnpFull	Y	Y	Y	Y	Y	64B	Y	Y	0	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	-	0	Y	0[a]		0[a]		0[a]			Y
WriteEvictFull	Y	Y	Y	Y	Y	64B	Y	Y	Y	Y	0	Y	1	1	0	1	1	Y	0	-	0	Y	0[a]		0[a]		0[a]			Y
WriteCleanFull	Y	Y	Y	Y	Y	64B	Y	Y	Y	Y	0	Y	Y	1	0	1	1	Y	0	-	0	Y	0[a]		0[a]		0[a]			Y
WriteBackPtl	Y	Y	Y	Y	Y	64B	Y	Y	0	Y	0	Y	Y	1	0	1	1	Y	0	-	0	Y	0[a]		0[a]		0[a]			Y
WriteBackFull	Y	Y	Y	Y	Y	64B	Y	Y	Y	Y	0	Y	Y	1	0	1	1	Y	0	-	0	Y	0[a]		0[a]		0[a]			Y
WriteUniquePtlStash	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	1	0	1	1	Y	0	-	Y	Y	-	Y	Y	-	-	Y	Y	Y
WriteUniqueFullStash	Y	Y	Y	Y	Y	64B	Y	Y	Y	Y	Y	Y	Y	1	0	1	1	Y	0	-	Y	Y	-	Y	Y	-	-	Y	Y	Y
WriteUniquePtl	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	1	0	1	1	Y	0	-	Y	Y	0[a]		0[a]		0[a]			Y
WriteUniqueFull	Y	Y	Y	Y	Y	64B	Y	Y	Y	Y	Y	Y	Y	1	0	1	1	Y	0	-	Y	Y	0[a]		0[a]		0[a]			Y
StashOnceUnique	Y	Y	Y	Y	Y	64B	Y	Y	Y	Y	0	Y	Y	1	0	1	1	Y	0	-	0	Y	-	Y	Y	-	-	Y	Y	Y
StashOnceShared	Y	Y	Y	Y	Y	64B	Y	Y	Y	Y	0	Y	Y	1	0	1	1	Y	0	-	0	Y	-	Y	Y	-	-	Y	Y	Y
AtomicLoad	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y	0	Y	Y	-	-	Y	Y	-	-	Y
AtomicStore	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y	0	Y	0[a]		-	Y	0[a]			Y
AtomicCompare	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y	0	Y	Y	-	-	Y	Y	-	-	Y
AtomicSwap	Y	Y	Y	Y	Y	Y	Y	Y	0	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y	0	Y	Y	-	-	Y	Y	-	-	Y

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

A.2 Response message field mappings

Table A-3 shows the response message field mappings. See Table A-1 on page A-372 for the conventions used in the field mappings. For further information on field use see *Protocol flit fields* on page 12-314.

Table A-3 Response message field mappings

Response message	QoS	TgtID	SrcID	TxnID	Opcode	RespErr	Resp	DBID	PCrdType	CF		TraceTag
										FwdState	DataPull	
RspLCrdReturn	X	X	X	0	Y	X	X	X	X	X		X
SnpResp	Y	Y	Y	Y	Y	Y	Y	Y	0[a]	-	Y	Y
SnpRespFwded	Y	Y	Y	Y	Y	Y	Y	X	0[a]	Y	-	Y
CompAck	Y	Y	Y	Y	Y	0	0[a]	X	0[a]	0[a]		Y
RetryAck	Y	Y	Y	Y	Y	0	0[a]	X	Y	0[a]		Y
Comp	Y	Y	Y	Y	Y	Y	Y	Y	0[a]	0[a]		Y
RespSepData	Y	Y	Y	Y	Y	Y	Y	Y	0[a]	0[a]		Y
CompDBIDResp	Y	Y	Y	Y	Y	Y	0	Y	0[a]	0[a]		Y
DBIDResp	Y	Y	Y	Y	Y	0	0[a]	Y	0[a]	0[a]		Y
PCrdGrant	Y	Y	Y	0[a]	Y	0	0[a]	0[a]	Y	0[a]		Y
ReadReceipt	Y	Y	Y	Y	Y	0	0[a]	X	0[a]	0[a]		Y

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

A.3 Data message field mappings

Table A-4 shows the data message field mappings. See Table A-1 on page A-372 for the conventions used in the field mappings. For further information on field use see *Protocol flit fields* on page 12-314.

Table A-4 Data message field mappings

Data message	QoS	TgtID	SrcID	TxnID	Opcode	RespErr	Resp	DBID	CCID	DataID	RSVDC	BE	Data	HomeNID	Common Field			TraceTag	DataCheck	Poison
															FwdState	DataPull	DataSource			
DatLCrdReturn	X	X	X	0	Y	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	0[a]	-	Y	Y	Y	Y	Y
SnpRespDataFwded	Y	Y	Y	Y	Y	Y	Y	X	Y	Y	Y	Y	Y	0[a]	Y	-	-	Y	Y	Y
CopyBackWrData	Y	Y	Y	Y	Y	Y	Y	X	Y	Y	Y	Y	Y	0[a]		0[a]		Y	Y	Y
NonCopyBackWrData	Y	Y	Y	Y	Y	Y	Y	X	Y	Y	Y	Y	Y	0[a]		0[a]		Y	Y	Y
NCBWrDataCompAck	Y	Y	Y	Y	Y	Y	Y	X	Y	Y	Y	Y	Y	0[a]		0[a]		Y	Y	Y
CompData	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	X	Y	Y	-	-	Y	Y	Y	Y
DataSepResp	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	X	Y	Y	-	-	Y	Y	Y	Y
SnpRespDataPtl	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0[a]	-	Y	Y	Y	Y	Y
WriteDataCancel	Y	Y	Y	Y	Y	Y	Y	X	Y	Y	Y	0	Y	0[a]		0[a]		Y	Y	Y

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

A.4 Snoop Request message field mappings

Table A-5 shows the snoop request message field mappings. See Table A-1 on page A-372 for the conventions used in the field mappings. For further information on field use see *Protocol flit fields* on page 12-314.

Table A-5 Snoop Request message field mappings

Snoop Request message	QoS	SrcID	TxnID	Opcode	Addr[(43-51):3]	NS	FwdNID	Common Field				CF		RetToSrc	TraceTag
								FwdTxnID	StashLPIDValid	StashLPID	VMIDExt	DoNotGoToSD	DoNotDataPull		
SnpLCrdReturn	X	X	0	Y	X	X	X	X	X	X	X	X	X	X	X
SnpShared	Y	Y	Y	Y	Y	Y	0[a]	0[a]				Y	-	Y	Y
SnpClean	Y	Y	Y	Y	Y	Y	0[a]	0[a]				Y	-	Y	Y
SnpOnce	Y	Y	Y	Y	Y	Y	0[a]	0[a]				Y	-	Y	Y
SnpNotSharedDirty	Y	Y	Y	Y	Y	Y	0[a]	0[a]				Y	-	Y	Y
SnpUnique	Y	Y	Y	Y	Y	Y	0[a]	0[a]				1	-	Y	Y
SnpCleanShared	Y	Y	Y	Y	Y	Y	0[a]	0[a]				1	-	0	Y
SnpCleanInvalid	Y	Y	Y	Y	Y	Y	0[a]	0[a]				1	-	0	Y
SnpMakeInvalid	Y	Y	Y	Y	Y	Y	0[a]	0[a]				1	-	0	Y
SnpSharedFwd	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	Y	-	Y	Y
SnpCleanFwd	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	Y	-	Y	Y
SnpOnceFwd	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	Y	-	0	Y
SnpNotSharedDirtyFwd	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	Y	-	Y	Y
SnpUniqueFwd	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	1	-	0	Y
SnpUniqueStash	Y	Y	Y	Y	Y	Y	0[a]	-	Y	Y	-	-	Y	0	Y
SnpMakeInvalidStash	Y	Y	Y	Y	Y	Y	0[a]	-	Y	Y	-	-	Y	0	Y
SnpStashUnique	Y	Y	Y	Y	Y	Y	0[a]	-	Y	Y	-	-	Y	0	Y
SnpStashShared	Y	Y	Y	Y	Y	Y	0[a]	-	Y	Y	-	-	Y	0	Y
SnpDVMOp	Y	Y	Y	Y	Y	0[a]	0[a]	-	-	-	Y	0[a]		0[a]	Y

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Appendix B
Communicating Nodes

This appendix specifies, for each packet type, the nodes that communicate using that packet type. It contains the following sections:

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

B.1 Request communicating nodes

Table B-1 shows the Request communicating nodes.

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 B-1 Request communicating nodes

Request	From	To	
		Expected	Permitted
ReadNoSnp WriteNoSnpFull WriteNoSnpPtl	RN-F, RN-D, RN-I	ICN(HN-F, HN-I)	-
	ICN(HN-F)	SN-F	-
	ICN(HN-I)	SN-I	-
ReadNoSnpSep	ICN(HN-F)	SN-F	-
	ICN(HN-I)	SN-I	-
ReadClean ReadShared ReadNotSharedDirty ReadUnique CleanUnique MakeUnique Evict WriteBackFull WriteBackPtl WriteEvictFull WriteCleanFull	RN-F	ICN(HN-F)	ICN(HN-I)
ReadOnce ReadOnceCleanInvalid ReadOnceMakeInvalid StashOnceUnique StashOnceShared WriteUniqueFull WriteUniqueFullStash WriteUniquePtl WriteUniquePtlStash	RN-F, RN-D, RN-I	ICN(HN-F)	ICN(HN-I)
CleanShared CleanSharedPersist CleanInvalid MakeInvalid	RN-F, RN-D, RN-I	ICN(HN-F, HN-I)	-
	ICN(HN-F)	SN-F	-
	ICN(HN-I)	SN-I	-
DVMOp	RN-F, RN-D	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	-

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table B-1 Request communicating nodes (continued)

Request	From	To	
		Expected	**Permitted**
AtomicStore AtomicLoad AtomicSwap AtomicCompare	RN-F, RN-D, RN-I	ICN(HN-F, HN-I)	-
	ICN(HN-F)	SN-F	-
	ICN(HN-I)	SN-I	-
PrefetchTgt	RN-F, RN-D, RN-I	SN-F	-

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

B.2 Snoop communicating nodes

Table B-2 shows the Snoop communicating nodes.

Table B-2 Snoop communicating nodes

Snoop	From	To
SnpShared	ICN(HN-F)	RN-F
SnpClean		
SnpOnce		
SnpNotSharedDirty		
SnpUnique		
SnpCleanShared		
SnpCleanInvalid		
SnpMakeInvalid		
SnpSharedFwd		
SnpCleanFwd		
SnoopOnceFwd		
SnoopNotSharedDirtyFwd		
SnpUniqueFwd		
SnpUniqueStash		
SnoopMakeInvalidStash		
SnpStashUnique		
SnpStashShared		
SnpDVMOp	ICN(MN)	RN-F, RN-D

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

B.3 Response communicating nodes

Table B-3 shows the Response communicating nodes.

Table B-3 Response communicating nodes

Response		From	To
Upstream	RetryAck DBIDResp PCrdGrant Comp	ICN(HN-F, HN-I, MN)	RN-F, RN-D, RN-I
		SN-F	ICN(HN-F)
		SN-I	ICN(HN-I)
	CompDBIDResp ReadReceipt	ICN(HN-F, HN-I)	RN-F, RN-D, RN-I
		SN-F	ICN(HN-F)
		SN-I	ICN(HN-I)
	RespSepData	ICN(HN-F, HN-I)	RN-F, RN-D, RN-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)

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

B.4 Data communicating nodes

Table B-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.

Table B-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, SN-I	RN-F, RN-D, RN-I	-
Downstream	CopyBackWrData	RN-F	ICN(HN-F)	ICN(HN-I)
	WriteDataCancel	RN-F, RN-D, RN-I	ICN(HN-F, HN-I)	-
		ICN(HN-F)	SN-F	-
		ICN(HN-I)	SN-I	-
	NonCopyBackWrData	RN-F, RN-D, RN-I	ICN(HN-F, HN-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	-

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Appendix C
Revisions

This appendix describes the technical changes between released issues of this specification.

Table C-1 Issue B

Change	Location
No changes, first public release	–

Table C-2 Issue C

Change	Location
New feature, Response after receiving first Data packet.	*Snoopable Reads excluding ReadOnce** on page 2-39. *ReadNoSnp, ReadOnce, ReadOnceCleanInvalid, ReadOnceMakeInvalid* on page 2-44.
New feature, Separate Non-data and Data-only response.	*Reads with separate Non-data and Data-only responses* on page 2-50 and multiple related locations.
New feature, Combined CompAck with WriteData.	*WriteUnique transaction* on page 2-89 and multiple related locations.
Clarification, concerning byte enables for Write transactions.	*Byte Enables* on page 2-116.
Update, concerning the list of fields that can change value from the original request in the retried request.	*Request Retry* on page 2-126.
Clarification, concerning the transaction responses permitted to be sent to the same address when a Snoop transaction response is pending.	*At the RN-F node* on page 4-195.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table C-2 Issue C (continued)

Change	Location
Additional information, concerning Legal RespErr field values for WriteDataCancel.	Table 9-6 on page 9-277.
Clarification, regarding TraceTag field value propagation.	*TraceTag usage and rules* on page 11-295.
Corrections and additions, concerning message field mappings.	Table A-2 on page A-373, Table A-3 on page A-374, and Table A-4 on page A-375.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Glossary

This glossary describes some of the technical terms used in AMBA 5 CHI documentation.

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 the 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 (AMBA).

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 that can hold a copy of the data value for a particular address location.

Cache hierarchy
The organization of different size caches in a hierarchy, typically with 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 connected 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.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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.

See also 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 *Cache state model* on page 1-26 for a description of the cache states supported in this specification.

Channel A set of signals grouped together to communicate a particular set of messages between a transmitter and receiver pair. For example Request channel is used to communicate request messages.

A channels consist of a set of information signals and a separate Valid and Credit signal to provide the channel handshake mechanism.

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.

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.

Completer *See* Completer on page 1-21.

Component A distinct functional unit that has at least one AMBA interface. Component can be used as a general term for master, slave, peripheral, and interconnect components.

See also Interconnect component, Master component, Memory slave component, Peripheral slave component, Slave 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 slave component.

Direct Data Transfer
Sending Read data directly from a Snoopee or Slave to the Requester bypassing the Home node.

Don’t Care *See* Don’t Care on page 1-22.

Downstream A transaction operates between a master component and one or more slave 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 slave component, and includes the destination slave component.

Downstream and upstream are defined relative to the transaction as a whole, not relative to individual data flows within the transaction.

See also Master component, Peer to Peer, Slave component, Upstream.

Downstream Cache
See Downstream cache on page 1-21.

Endpoint *See* Endpoint on page 1-22.

Final Destination
Final destination for a Memory transaction is a peripheral or physical memory, also called an Endpoint.

Flit *See* Flit on page 1-21.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HN *See* HN on page 1-22.

ICN *See* ICN on page 1-22.

In a timely manner
See In a timely manner on page 1-22.

IMPLEMENTATION DEFINED
Behavior that is not defined by the architecture, but is 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.

Interconnect component
A component with more than one AMBA interface that connects one or more master components to one or more slave components.

An interconnect component can be used to group together either:

- A set of masters so that they appear as a single master interface.
- A set of slaves so that they appear as a single slave interface.

See also Component, Master component, Slave component.

IO Coherent node
See IO Coherent node on page 1-22.

Line *See* Cache line.

Link A Link is the connection used for communicating between a transmitter and receiver pair.

Link layer Credit
See Link layer Credit on page 1-22.

Load The action of a master 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 master 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 neither in the RNs nor in the Interconnect.

Main memory can be referred to as memory when the context makes the intended meaning clear.

Master *See* Master on page 1-21.

Master component
A component that initiates transactions.

It is possible that a single component can act as both a master component and as a slave component. For example, a Direct Memory Access (DMA) component can be a master component when it is initiating transactions to move data, and a slave component when it is being programmed.

See also Component, Interconnect component, Slave component.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 slave component
A memory slave component, or *memory slave*, is a slave component with the following properties:

- A read of a byte from a memory slave returns the last value written to that byte location.
- A write to a byte location in a memory slave 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, Master component, Peripheral slave component.

Message *See* Message on page 1-21.

Observer A processor or other master 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 ReadReceipt, CompData, DBIDResp, Comp, CompDBIDResp 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.
  - Cancelled, and returns the received credit using the PCrdReturn message.

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, from one RN to another RN.

See also Downstream, Upstream.

Peripheral slave component
A peripheral slave component is also described as a *peripheral slave*. This specification recommends that a peripheral slave 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 slave to fail, but must complete in a protocol-compliant manner to prevent system deadlock. The protocol does not require continued correct operation of the peripheral.

See also Memory slave component, Slave 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 components or the interconnect.

Packet *See* Packet on page 1-21.

Phit *See* Phit on page 1-21.

PoC *See* PoC on page 1-21.

PoS *See* PoS on page 1-21.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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 on page 1-22.

Requester *See* Requester on page 1-21.

RN *See* RN on page 1-22.

Slave *See* Slave on page 1-21.

Slave component
A component that receives transactions and responds to them.

It is possible that a single component can act as both a slave component and as a master component. For example, a Direct Memory Access (DMA) component can be a slave component when it is being programmed and a master component when it is initiating transactions to move data.

See also Master component, Memory slave component, Peripheral slave component.

SN *See* SN on page 1-22.

Snooped cache A hardware-coherent cache that receives Snoop transactions.

Snoop filter A snoop filter is able to track the cache lines that might be allocated within a master.

Speculative read
A transaction that a component 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 speculative read is performed 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 master 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 master 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.

See also Load, Permission to store.

Synchronization barrier
See Barrier.

System Memory Management Unit (SMMU)
A system-level MMU. That is, a system component that provides address translation from 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 on page 1-21.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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.

See also Aligned.

UNPREDICTABLE In the AMBA Architecture 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 master component and one or more slave 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 master component, and includes the originating master 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, Master component, Peer to Peer, Slave component.

Write-Back cache
A cache in which, when a store is permitted to store, 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 next level cache or main memory when the cache line is cleaned or re-allocated. Another common term for a Write-Back cache is a *Copy-Back* cache.

Write-Invalidate protocol
See Write-Invalidate protocol on page 1-22.

Copyright © 2014, 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Non-Confidential