

Concerto-C Messaging Protocol (CCMP)



Author: Michael Frank

Approver: Hunglin Hsu
Bernard Bonardi

Version: Ncore 3.6

Revision No.	Date	Author	Description of Changes
0.00	March 2018	Sanjay Deshpande	
	September 2018	Sanjay Deshpande	<p>Added MrdPref</p> <p>Changed DtwNoData to DtwNullData to make compatible with CHI usage of WriteDataCancel and CopyBackWrData_I (Table 4-42).</p>
	Nov 14, 2018	Sanjay Deshpande	<p>MPF1 in DTW updated to change {Valid StashNId} to StashNId.</p> <p>MPF1 in DTW changed to no more carry ArgV[], e.g for Far Atomics. The DMI receives ArgV[] with the Nc CmdReq, which are used for the Atomics.</p> <p>MPF2 in DTW updated to change {Valid Stash Target Reference Message Id} to Stash Target Reference Message Id.</p> <p>Added explanation for the VZ bit and RL[] field. Updated MRD section for the same and for the semantics of RL[].</p>
	Nov 20, 2018	Sanjay Deshpande	Added verbiage for MPF1 and MPF2 fields in the SNPreq.
	Nov 29, 2018	Sanjay Deshpande	Added Ncore 3 implementation notes.
	Dec 7, 2018	Sanjay Deshpande	Added detailed explanation for the RL[] field values.
	Dec 20, 2018	Sanjay Deshpande	Removed MPF2 field from DTRreq message. (CONC-4586)
	Jan 8, 2019	Sanjay Deshpande	<p>Modified Table 4-27 to indicate snooper choices for snoops of Read operations. Also added Architecture Notes for it.</p> <p>Modified Table 4-16 to change the VZ setting to Nv for Atomic CMDs.</p>
	Jan 10, 2019	Sanjay Deshpande	Added notes related to WriteDataCancel for DtwNullData and DtwM- rgMRDUDty in Table 4-42 and Table 4-43 and to Section 4.8.3.
	Jan 14, 2019	Sanjay Deshpande	Added operation protocols to Chapter 5.
	Jan 15, 2019 - Jan 17, 2019	Sanjay Deshpande	<p>Added TOF, QoS, and MPF1 fields to RBRReq. Added QoS and MPF3 fields in MRD.</p> <p>Added verbiage for RBR.</p> <p>Added Field Source Column for all Concerto-C messages. Added Snoop Protocol tables to Coherency Protocol chapter.</p>
	Jan 23, 2019	Sanjay Deshpande	Added Table 4-15 specifying RL[] settings for messages.
	Jan 28, 2019 - Jan 29, 2019	Sanjay Deshpande	Updated Table 4-31 and Table 4-32 and for DVMOp snoops. Also added Table 4-33.
	Feb 4, 2019	Sanjay Deshpande	Added EO signal to DTRreq, SNPreq, MRDreq, and RBRreq message to enable Ex-Okay transmission.
	Feb 13, 2019	Sanjay Deshpande	Explained Flow Id in MPF2 can carry Processor Id from a processor cluster.
	Feb 15, 2019	Sanjay Deshpande	Added Silent State Transition support information. Section 2.4.6.
	Feb 21, 2019	Sanjay Deshpande	Added a section on Snoop Response Processing (Section 5.4).
	Mar 6, 2019	Sanjay Deshpande	Added Table 4-24.
	Mar 26, 2019	Sanjay Deshpande	Updated Table 4-4 to add entry for SNPrsp for DVM snoops.
	Mar 28, 2019	Sanjay Deshpande	Added IOAIU Coherency State Transition Tables.
	Apr 3, 2019	Sanjay Deshpande	Updated entry for SnpUnqStsh in Table 4-21.
	Apr 4, 2019	Sanjay Deshpande	Added Section 4.5.3.3.1.2.
	Apr 5, 2019	Sanjay Deshpande	Changed entry in Table 5-25 for SnpRespData_I to drop data instead of issuing a DTWreq.
	Apr 8, 2019	Sanjay Deshpande	<p>Fixed SNPrsp entries in Table 5-26 to correctly reflect DTWreqs.</p> <p>Also added Section 4.5.3.3.4.1 and Section 4.5.3.3.4.2.</p>

Revision No.	Date	Author	Description of Changes
	Apr 11, 2019	Sanjay Deshpande	Completion actions added to Chapter 5. Content reorganized for more logical flow. Added Table 5-33.
	May 15, 2019	Sanjay Deshpande	Updated Section 4.6.5.6 with more verbiage on EO field. Also added Section 4.5.3.4.2.1 on OR[] field.
	Apr 23, 2019	Sanjay Deshpande	Updated Section 4.5.3.2 for Addr field with requirements. Added UpdSCLn operation. Added codes for UpdInv and UpdSCLn. Populated Chapter 8.
	May 7, 2019	Sanjay Deshpande	Updated Section 4.8.1.2 for additional comments about Dirty data carried in DtwMrgMrds.
	May 22, 2019 - May 30, 2019	Sanjay Deshpande	Added Section 2.4.2.1 on Reservation and Exclusive sequence, and related comments and notes elsewhere where relevant. Added Section 4.8.2 on DTW ordering.
	June 4, 2019	Sanjay Deshpande	Added a code point for SNPResp with Data Error in CMStatus[] - needed to report to DCE. Changed Success codes to have [7:6] to be 2'b00 instead of 2'b0x. Updated Table 4-4.
	June 6, 2019	Sanjay Deshpande	Updated Chapter 8 with more examples and explanations.
	June 24, 2019	Sanjay Deshpande	Updated protocol entries for SnpNITCMIDtr, SnpUnqStsh, SnpStshShd, and SnpStshUnq in Chapter 5. Reference: Jira entry CONC-5292.
	July 3, 2019	Sanjay Deshpande	Updated Section 4.6.7.5 and Section 4.11.4.6. Reference: Jira entry CONC-5353. Added IntfSize back into DTWreq. Removed from RBRreq. Refer to CONC-5359.
	July 9, 2019 - July 19, 2019	Sanjay Deshpande	Added minimum Data transfer amount requirement for Ncore 3. Section 8.1. Added Section 8.2 on data alignment. Update to wording of the ST field description in Table 4-5.
	July 31, 2019 - Aug 6, 2019	Sanjay Deshpande	Added content to Chapter 6 on Consistency Processing. Added Section 8.3 on Data Identification. Added Section 6.3.1.3.1 on CMO Handling.
	Aug 6, 2019	Sanjay Deshpande	Added Section 4.5.3.3.1.1 on ST bit usage. Added Section 4.5.3.4.1.3 on Narrow Transfers.
	Aug 11, 2019	Sanjay Deshpande	Added Chapter 7 on DVMOp processing in CCMP.
	Aug 13, 2019	Sanjay Deshpande	Added to Chapter 7 on DVMOp processing.
	Aug 15, 2019 - Aug 16, 2019	Sanjay Deshpande	In Chapter 6: Added consistency rules for CHI AIU for Writes. Added CopyBack Write handling flow for ACE AIU and for Proxy Cache. Added Section 6.1.1.
	Aug 19-28, 2019	Sanjay Deshpande	Clean-up, re-org of Chapter 6.
	Aug 29, 2019	Sanjay Deshpande	Updated CCMP Message Flow diagrams for DVMs in Chapter 7.
	Sep 3, 2019	Sanjay Deshpande	Updated Table 4-16. Updated Section 4.5.3.6. Updated Section 2.2.1.
	Sep 5, 2019	Sanjay Deshpande	Updated Section 4.5.3.6. Updated Section 2.2.1. Important updates to Section 4.5.3.4.1 on handling transfers over AXI at DMI and DII for ST=1 and INCR accesses.
	Sep 9, 2019	Sanjay Deshpande	Updated Section 4.5.3.3.1.2 to add precision and to distinguish between handling CHI and ACE.
	Sep 10, 2019	Sanjay Deshpande	Updated Table 4-18. Fixed FlowId[7:0] typo.
	Sep 18, 2019	Sanjay Deshpande	Updated Table 5-17 entry for SnpRespData_I_PD. For SnpClnDtw, the UP-field is don't care. Other miscellaneous updates per Diego feedback.
	Sep 24, 2019 - Sep 25, 2019	Sanjay Deshpande	Updated Table 4-16 to indicate AC bit is set to 1 in CMDreq by the AIU for Atomic operations. Added indication that Footnote f and Footnote g also apply to Atomic operation arriving over an ACE5-Lite interface. Added Application Note to Section 4.5.3.8 and Footnote to Table 4-17 for Unit ID usage.

Revision No.	Date	Author	Description of Changes
	Sep 30, 2019	Sanjay Deshpande	Updated Table 3-1 with additional informational content.
	Oct 3, 2019	Sanjay Deshpande	Added Appendix 1 to contain description of Coherency Directory. This information has been brought over nearly verbatim from Ncore 2 and CMPS documentation.
	Oct 4, 2019	Sanjay Deshpande	Reset Change Bars (RCB).
0.81	Nov 22, 2019	MK	WRAP encoding changed to match AXI Update valid bit positioning in MPF field Update ACE WriteClean handling Updated StashOnce Mrd Prefetch requirements Removed references to SO SS SD ST bits from StrReq Strike across hint messages Updated message fields for all messages: strike across unused fields Updated CH bit to be don't care (strike across)
0.82	Jan 21, 2020	MK	Typo fix for AXI AIU with cache SnpVldDtw to SnpClnDtw Added a note regarding AXI to SMI command mapping when cache is not present and when in NC mode
0.83	Jan 22, 2020	MK	Updated Concerto message fields to match as per implementation, strike through unused fields
0.84	Mar 10, 2020	MK	Updated to indicate that MPF1 must be used for non-modifiable narrow transactions Added clarification on RL values for MRDs generated due to stashing request Clarified OR bits to DCE.
0.85	Apr 17, 2020	MK	Update AXI proxy cache table to include stash snoops Updated table 4-15 for RL 11 request
0.86	May 27, 2020	MK	Added AWUNIQUE signal option to MPF1 field of command
0.87	June 15, 2020	MK	Update Table 4-16 with ACE write clean and clean unique VZ setting
0.88	June 24, 2020	MK	Updated CMDreq mapping Table in Appendix. Added a foot note regarding ACE mapping for Evict, WriteClean, WriteBack & WriteEvict
0.89	July 13, 2020	MK	Removed reference to legacy TR bits in StrRsp
0.90	Sep 27, 2020	MK	Updated Table 4-16 Write back CA value for ACE and NC exclusive attributes
0.91	Dec 19, 2022	Michael Frank	Updated A1.1 CMDreq mappings
0.92	Jan 24, 2023, to Mar 16, 2023	Michael Frank	Reformatted version 0.91 to new Arteris Spec format and made some minor adjustments: Updated Footnote in section A1.1 to remove WriteClean from list of commands causing UPDreq (CONC-11055) Added Author/Reviewer etc. on front page
0.93	Mar 18, 2023	Michael Frank	Processing CONC-11520 – highlight transactions in Table 4-16: Attribute applicability per type of CMD Table 4-16: Attribute applicability per type of CMD , added comment that it may require update in column VZ
0.94	Mar 18, 2023	Michael Frank	Added new CCMP transactions for SYScmd (SYSreq and SYSrsp)
0.95	Mar 19, 2023	Michael Frank	Added a table explaining the mapping from AxCACHE[] = 0000/0010 → Table 6-1 Rewrote section on DVM using new tables
0.95.1	Mar 24, 2023	Michael Frank	Updated Table 4-16 to VZ = Nv for several transactions Updated Table 4-23 to correct snoop command used on ACE
0.96	April 26, 2023	Michael Frank	JIRA CONC-11854: fixed typos and cut and paste errors in Table 7-5 and 7-6
0.97	April 26, 2023	Michael Frank	Update to RBR/RBU Protocol – transition to Ncore 3.6 Sections for RBU have been removed
0.98	June 14, 2023	Michael Frank	Modified RBR/RBU signal mapping as requested by CONC-12020 and added feedback from inspection review of Ncore 3.4 CCMP
0.99	June 20, 2023	Michael Frank	Updated table 4-15/4-16 as requested by CONC-12051

Revision No.	Date	Author	Description of Changes
			<p>Add changes to CCMP to support CHI-E - how new transaction types are mapped - improved protocol efficiency beyond RBR/RBU protocol</p>
0.991	June 27, 2023	Cheng Chung Wang	Update mpf2/mpf3, and DVM tables
0.992	August 22, 2023	Michael Frank	Fixes for CONC-12377, updated drawings
0.993	October 30, 2023	Said Derradji	Fix for CONC-13039 Table 14-5 updated : For stashing, MRDRd has to set RL = 2'11
0.994	January 15, 2024	Said Derradji	Fix for CONC-13422 Table 4-59 updated: SYSreq.RL removed
	March 1, 2024	Hao Luan	Added clarifications to clarify WriteCleanPtl and WriteBackPtl [CONC-13776]
1.0	March 4, 2024	Hao Luan	Updated per AIP_WPs_VerificationReport_Ncore3.6 review comments, removed extra review comments and accepted previous format changes made by Michael Frank
1.0.1	March 27, 2024	Hao Luan	<p>Corrected DVM non-sync (Figure 7-2Figure 7-2) and DVM sync (Figure 7-3Figure 7-3) flows per described in [CONC-14113]</p>
	March 28, 2024	Hao Luan	Further corrected flow diagrams of Figure 7-1 Figure 7-1 , Figure 7-5 Figure 7-5 , and Figure 7-6 Figure 7-6 in Chapter 7: related to DVM flows per described in [CONC-13776]
	April 1 – April 11	Hao Luan	<p>Further clarified and cleaned up misc. items such as:</p> <ul style="list-style-type: none"> • replaced CHI-A by CHI-E, • replaced ACE-Lite-E by ACE5-Lite, • completely removed previous section 4.15 TUNreqs and Tunrsp, • minor cosmetic issues
1.0.2	April 16, 2024	Hao Luan	Made the final release
1.0.3	April 17-18, 2024	Hao Luan	<ul style="list-style-type: none"> • Removed Size field in Table 4-28Table 4-28 as reported in [conc-13776] since this field doesn't exist ever since the beginning • Updated and clarified the user bits usage for CMDreq in Table 4-3Table 4-3 and section 4.5.3.11 as described in [conc-13776] • Further corrected mis referencing of tables in section Chapter 4: • Updated the order policy in section 4.5.3.3.4 as described in [CONC-14258]
1.0.4	May 16, 2024	Hao Luan	Corrected section 7.3.3.3.3 on how an ACE agent receives DVM snoop from a DVE per described in [CONC-14390]
1.0.5	June 12, 2024	Hao Luan	<ul style="list-style-type: none"> • Corrected the typo in Table 5-27Table 5-27 per discussed in [CONC-14752] • Removed two redundant rows in Table 4-25Table 4-25 per described in [CONC-14751] • Corrected the wrong note in one of the sub fields in Table 5-28Table 5-28 per described in [CONC-14753]
1.0.6	June 28, 2024	Hao Luan	<ul style="list-style-type: none"> • Increased MPF2 field size by 10 in Table 4-3Table 4-3 to accommodate 20 bits AxID support per discussed in [MAES-7323]
1.0.7	September 6, 2024	Hao Luan	<ul style="list-style-type: none"> • Reconcilled and corrected stash related specification mismatches created by CHI-A by CHI-E global replacement in version 1.0.1 and made updates in section 4.6.3 <ul style="list-style-type: none"> ◦ Added CHI-E supports for stash in Table 4-21 ◦ Removed a duplicated row for "Snoop Invalidate and Stash" ◦ Corrected CHI-E supports for stash in Table 4-23 ◦ Corrected CHI-E supports for stash in Table 4-25 per described in [CONC-15287], [CONC-15301] and [CONC-15336] • Clarified Write Clean Partial handling for CHI-E in Table 3-1

Revision No.	Date	Author	Description of Changes
			<ul style="list-style-type: none">• Clarified stash supports for CHI-E by removing CHI-E from non-stashing support in section 4.6.7.3• Clarified sentence related to CHI-E on stash support in section 5.3.2• Clarified Write Clean Partial handling for CHI-E in Table 10-8• Removed Snoop Data Error – represented by 2'b01 in Table 4-4 per suggested in [CONC-15373] and made it as a reserved value since it has never been used.

Chapter 1: Introduction	1
1.1 Native Operations and Transactions	3
1.1.1 Operations	3
1.1.2 Transactions	3
1.2 Concerto-C Messaging Protocol.....	4
1.2.1 The Ncore platform.....	4
1.2.1.1 The CCMP Domain	4
1.2.2 Overview of the Concerto-C Messaging Protocol.....	4
1.2.2.1 CCMP Messages	4
1.2.3 Initiators and Targets	5
1.2.4 Data and non-Data message forms.....	5
1.2.5 Request and response messages.....	5
1.2.6 Concerto-C Messaging Protocol Stack	5
1.3 Ncore 3 system overview.....	6
1.3.1 Agents and Targets.....	6
1.3.2 Concerto-C Units.....	8
1.3.2.1 AIUs	8
1.3.2.2 DCEs	8
1.3.2.3 DMIs.....	9
1.3.2.4 DLLs.....	9
1.3.2.5 DVE	9
1.3.3 Legato: The CCMP transportfabric	10
1.3.4 Unit Identifiers	11
1.3.5 Ncore system organization	11
Chapter 2: Storage Architecture.....	13
2.1 Storage Classification	14
2.2 System storage and access sizes	14
2.2.1 Atomic Accesses	15
2.3 Concerto-C Memory	17
2.4 Coherency maintenance	18
2.4.1 PoS and PoC.....	18
2.4.2 States of a Coherency Granule.....	19
2.4.2.1 The Reservation State	20
2.4.2.2 State labels in CCMP Directory.....	21
2.4.3 Access Rules and State Invariants	21
2.4.4 Intervention response.....	22
2.4.5 Ownership levels of cacheline.....	23
2.4.6 Silent cacheline state transitions within agents.....	23
2.4.7 Coherency Protocol Models.....	24

Chapter 3: Concerto-C Operations	26
3.1 Concerto-C Protocol Operations	27
3.2 Mapping of Native Operations to CCMP	36
3.2.1 Special handling of WriteBacks in ACE.....	37
3.3 Mapping of CCMP operations to Native Interfaces	38
Chapter 4: Concerto-C Messages	39
4.1 Conventions and Notations	40
4.2 Concerto Message Types	41
4.3 Message Formats	43
4.4 Concerto-C Message Headers	44
4.4.1 Common Message Header fields	44
4.4.1.1 Target ID.....	44
4.4.1.2 Initiator ID	44
4.4.1.3 CMType.....	44
4.4.1.4 Message ID	45
4.4.1.5 HProtection.....	45
4.4.2 Transmit Header Extension.....	45
4.4.2.1 TTier	45
4.4.2.2 Steering.....	45
4.4.2.3 Priority.....	46
4.4.2.4 QL	46
4.5 CMDreq and CMDrsp Messages	47
4.5.1 CMDreq Messages.....	47
4.5.2 CMDreq message fields	49
4.5.3 CMDreq field descriptions	51
4.5.3.1 CMStatus	51
4.5.3.2 Addr	53
4.5.3.3 Operation attributes.....	53
4.5.3.4 Multi-Purpose Field 1 (MPF1)	63
4.5.3.5 Multi-Purpose Field 2 (MPF2)	64
4.5.3.6 Size[2:0].....	65
4.5.3.7 IntfSize[2:0]	65
4.5.3.8 DId[]	66
4.5.3.9 TOF[2:0]	66
4.5.3.10 QoS	67
4.5.3.11 AUX	67
4.5.3.12 NDProt.....	67
4.5.4 CMD Responses	67
4.5.5 CMDrsp messages.....	67
4.5.6 CMDrsp message fields.....	68
4.5.7 CMDrsp field descriptions.....	68

4.5.7.1	RMessageId	68
4.5.7.2	CMStatus[7:0].....	68
4.5.7.3	NDPROT.....	68
4.6	SNPreq and SNPrsp Messages	69
4.6.1	SNPreq messages	69
4.6.2	Mapping of CMD requests to SNPrequests.....	72
4.6.3	Mapping of SNPreq to Native Snoop transactions.....	73
4.6.3.1	Adjustments for Stashing Snoops.....	74
4.6.3.2	Issuing snoops to AIUs	77-76
4.6.4	SNPreq message fields	79-78
4.6.5	SNPreq field descriptions	80-79
4.6.5.1	CMStatus	80-79
4.6.5.2	Addr	80-79
4.6.5.3	Operation attributes.....	80-79
4.6.5.4	UP[1:0].....	80-79
4.6.5.5	RL[1:0]	84-83
4.6.5.6	EO	84-83
4.6.5.7	Multi-Purpose Field-1 (MPF1).....	85-84
4.6.5.8	Multi-Purpose Field-2 (MPF2).....	86-85
4.6.5.9	Multi-Purpose Field-3 (MPF3).....	87-86
4.6.5.10	Size	87-86
4.6.5.11	IntfSize	87-86
4.6.5.12	Did (Destination ID)	87-86
4.6.5.13	TOF	87-86
4.6.5.14	QoS.....	87-86
4.6.5.15	RBID	87-86
4.6.5.16	AUX	88-86
4.6.5.17	MProt.....	88-87
4.6.6	SNPrsp Messages	89-88
4.6.7	SNPrsp field descriptions	90-89
4.6.7.1	RMessageId	90-89
4.6.7.2	CMStatus	90-89
4.6.7.3	Mapping of CHI snoop responses to CMStatus[5:0]	92-91
4.6.7.4	MPF1.....	93-92
4.6.7.5	IntfSize	93-92
4.6.7.6	QoS	93-92
4.6.7.7	AUX	93-92
4.6.7.8	MProt.....	93-92
4.7	DTRreq and DTRrsp Messages	94-93
4.7.1	DTRreq messages	94-93
4.7.2	Possible DTR requests per Read CMDreqs	95-94
4.7.3	Mapping of DTRreqs to Native transactions	96-95
4.7.4	DTRreq message fields	97-96

4.7.5	<i>DTRreq field descriptions</i>	100
4.7.5.1	Non-Data Payload.....	100
4.7.5.2	QoS.....	101
4.7.5.3	Data Payload.....	101
4.7.6	<i>DTRrsp Messages</i>	103
4.7.7	<i>DTRrsp field descriptions</i>	104
4.7.7.1	RMessageId	104
4.7.7.2	CMStatus	104
4.7.7.3	AUX	104
4.7.7.4	MPROT	104
4.8	DTWreq and DTWrsp Messages	104
4.8.1	<i>DTWreq messages</i>	104
4.8.1.1	DTW Types.....	105
4.8.1.2	DTWMrgMRD Types.....	106
4.8.2	<i>Ordering Considerations for DTWs</i>	107
4.8.2.1	Global ordering of accesses and messages.....	107
4.8.2.2	Order and performance of non-coherent Writes and their DTWs	108
4.8.2.3	Order and performance of coherent requests and their DTWs.....	108
4.8.2.4	Mutual order and performance of Reads and Writes	108
4.8.3	<i>DTWreq and DTWMrgMRDreq message fields</i>	109
4.8.4	<i>DTWreq and DTWMrgMRDreq field descriptions</i>	112
4.8.4.1	Non-Data Payload.....	112
4.8.4.2	RBID	112
4.8.4.3	CMStatus	112
4.8.4.4	IntfSize	114
4.8.4.5	QoS	114
4.8.4.6	AUX	114
4.8.4.7	MPROT	114
4.8.4.8	Data Payload.....	114
4.8.5	<i>DTWrsp Messages</i>	116
4.8.6	<i>DTWrsp field descriptions</i>	116
4.8.6.1	RMessageId	116
4.8.6.2	CMStatus	116
4.8.6.3	AUX	117
4.8.6.4	MPROT	117
4.9	MRDreqs and MRDrsp	118
4.9.1	<i>MRDreq messages</i>	118
4.9.2	<i>DTRreqs issued in response to MRDreqs</i>	120
4.9.3	<i>MRDreq message fields</i>	120
4.9.4	<i>MRDreq field descriptions</i>	122
4.9.4.1	CMStatus	122
4.9.4.2	Addr	122
4.9.4.3	Operation attributes.....	122
4.9.4.4	Multi-Purpose Field #1 (MPF1)	123

4.9.4.5 Multi-Purpose Field #2 (MPF2)	124123
4.9.4.6 Multi-Purpose Field #3 (MPF3)	124123
4.9.4.7 Size	124123
4.9.4.8 IntfSize	124123
4.9.4.9 QoS	124123
4.9.4.10 AUX	124123
4.9.4.11 MPROT	124123
4.9.5 MRDrsp Messages	125124
4.9.6 MRDrsp field descriptions	126125
4.9.6.1 RMessageld	126125
4.9.6.2 CMStatus[7:0]	126125
4.9.6.3 NDPROT	126125
4.10 HNTreqs and HNTrsps	126125
4.10.1 HNTreq messages	126125
4.10.2 HNTreq message fields	126125
4.10.3 HNTreq field descriptions	128127
4.10.3.1 CMStatus	128127
4.10.3.2 Addr	128127
4.10.3.3 Operation attributes	128127
4.10.3.4 IntfSize	128127
4.10.3.5 AUX	128127
4.10.3.6 MProt	128127
4.10.4 HNTrsp Messages	129128
4.10.5 HNTrsp field descriptions	130129
4.10.5.1 RMessageld	130129
4.10.5.2 CMStatus[7:0]	130129
4.10.5.3 MPROT	130129
4.11 STRreqs and STRrsps	131130
4.11.1 STRreq messages	131130
4.11.2 STRreq message fields	132131
4.11.3 STRreq field descriptions	133132
4.11.3.1 RMessageld	133132
4.11.3.2 CMStatus	133132
4.11.3.3 Operation attributes	133132
4.11.3.4 RBID	134133
4.11.3.5 Multi-Purpose Field #1 (MPF1)	134133
4.11.3.6 Multi-Purpose Field #2 (MPF2)	134133
4.11.3.7 IntfSize	134133
4.11.3.8 QoS	134133
4.11.3.9 AUX	134133
4.11.3.10 MPROT	134133
4.11.4 STRrsp Messages	135134
4.11.5 STRrsp field descriptions	136135
4.11.5.1 RMessageld	136135

4.11.5.2 CMStatus[7:0].....	136135
4.11.5.3 MPROT.....	136135
4.12 RBReq and RBReqsp	137136
4.12.1 Request Buffers.....	137136
4.12.1.1 Using RBs for data transfers in coherent CMDreqs	138137
4.12.1.2 Using RBs for non-coherent CMDreqs.....	140139
4.12.1.3 Independence of RBs and Data Buffers.....	140139
4.12.2 RBReq message.....	140139
4.12.3 RBReq message fields	141140
4.12.4 RBReq field descriptions	142141
4.12.4.1 RBID.....	142141
4.12.4.2 CMStatus	142141
4.12.4.3 RType	142141
4.12.4.4 Addr	142141
4.12.4.5 Operation attributes.....	142141
4.12.4.6 TOF	144143
4.12.4.7 QoS.....	144143
4.12.4.8 MPF1.....	144143
4.12.4.9 AUX	144143
4.12.4.10 MPROT.....	144143
4.12.5 RBReqsp Message	145144
4.12.6 RBReqsp message fields	145144
4.12.7 RBReqsp field descriptions	146145
4.12.7.1 RBID.....	146145
4.12.7.2 RBGen.....	146145
4.12.7.3 CMStatus[7:0].....	146145
4.12.7.4 AUX	146145
4.12.7.5 MPROT.....	146145
4.13 SYSreq and SYSrsp	147146
4.13.1 SYSreq message.....	147146
4.13.1.1 SYSreq Commands	148147
4.13.2 SYSreq message fields.....	148147
4.13.3 SYSreq field descriptions	149148
4.13.3.1 RMessageID	149148
4.13.3.2 CMStatus[7:0].....	149148
4.13.3.3 AUX	149148
4.13.3.4 MPROT.....	150149
4.13.4 SYSrsp message.....	151150
4.13.5 SYSrsp field descriptions	151150
4.13.5.1 RMessageID	151150
4.13.5.2 CMStatus[7:0].....	151150
4.13.5.3 AUX	152151
4.13.5.4 MPROT.....	152151

4.14 UPDreqs and UPDrsp	153	152
4.14.1 UPDreq message	153	152
4.14.1 UPDreq message fields	153	152
4.14.2 UPDreq field descriptions	155	154
4.14.2.1 CMStatus	155	154
4.14.2.2 Addr	155	154
4.14.2.3 Operation attributes.....	155	154
4.14.2.4 QoS	155	154
4.14.2.5 AUX	155	154
4.14.2.6 MProt.....	155	154
4.14.3 UPDrsp Message	156	155
4.14.4 UPDrsp message fields	156	155
4.14.5 UPDrsp field descriptions	157	156
4.14.5.1 RMessageID	157	156
4.14.5.2 CMStatus[7:0].....	157	156
4.14.5.3 AUX	157	156
4.14.5.4 MPROT.....	157	156
4.15 Miscellaneous Response Messages	158	157
4.15.1 CMPrsp Message	158	157
4.15.2 CMPrsp message fields	158	157
4.15.3 CMPrsp field descriptions	159	158
4.15.3.1 RMessageId	159	158
4.15.3.2 CMStatus[7:0].....	159	158
4.15.3.3 AUX	159	158
4.15.3.4 MPROT.....	159	158
4.15.4 CMErsp Message	160	159
4.15.5 CMErsp message fields	160	159
4.15.6 CMErsp field descriptions	161	160
4.15.6.1 RMessageId	161	160
4.15.6.2 ECMType	161	160
4.15.6.3 CMStatus[7:0].....	161	160
4.15.6.4 AUX	161	160
4.15.6.5 MPROT.....	161	160
4.15.7 TRErsp Message	162	161
4.15.8 TRErsp message fields	162	161
4.15.9 TRErsp field descriptions	163	162
4.15.9.1 RMessageId	163	162
4.15.9.2 ECMType	163	162
4.15.9.3 CMStatus[7:0].....	163	162
4.15.9.4 AUX	163	162
4.15.9.5 MPROT.....	163	162
4.16 Unit Pairs for Messages	164	163
Chapter 5: Coherency Processing	165	164

5.1 Processing of operations	166	165
5.1.1 Command Request Process	166	165
5.1.2 Snoop Process.....	167	166
5.1.3 Intervention Direct DTR Process.....	168	167
5.1.4 Intervention Indirect DTR Process.....	169	168
5.1.5 Intervention DTW Process.....	169	168
5.1.6 Write Data DTW Process	171	170
5.1.6.1 Classification of Writes	171	170
5.1.6.2 The Write DTW process.....	171	170
5.1.7 The Write Stashing process	173	172
5.1.7.1 Write Stash Request and Snoop Process.....	173	172
5.1.7.2 The Full Cacheline Write Stashing Process.....	173	172
5.1.7.3 The Partial Cacheline Write Stashing Process.....	173	172
5.2 Processing of coherent operations	175	174
5.3 Snoop Protocols.....	177	176
5.3.1 Snoop Protocols.....	177	176
5.3.1.1 SnpNITCDtr	177	176
5.3.1.2 SnpNITCCIDtr.....	178	177
5.3.1.3 SnpNITCMIDtr	179	178
5.3.1.4 SnpCleanDtr.....	180	179
5.3.1.5 SnpVldDtr	181	180
5.3.1.6 SnpInvDtr	182	181
5.3.1.7 SnpNoSDIntDtr	183	182
5.3.1.8 SnpClnDtw	184	183
5.3.1.9 SnpInvDtw.....	185	184
5.3.1.10 SnpInv	186	185
5.3.2 Stashing Snoop Protocols	187	186
5.3.2.1 SnpInvStsh	187	186
5.3.2.2 SnpUnqStsh	189	188
5.3.2.3 SnpStshShd.....	191	190
5.3.2.4 SnpStshUnq	193	192
5.4 Snoop Response Processing in DCE	195	194
5.4.1 Snoop responses.....	195	194
5.4.1.1 Retained State of Cacheline.....	195	194
5.4.2 Implied Retained State in System.....	196	195
5.4.3 Cumulative Retained State in System.....	196	195
5.4.4 Processing Snoop Responses from AIUs	196	195
5.4.5 Update of snooper's retained state	196	195
5.4.6 Summary Snoop Response Calculation	197	196
5.4.7 STRreq generation.....	197	196
5.4.8 Cache Maintenance Operation Processing	199	198
5.4.9 DTRreq generation for Reads.....	199	198
5.5 Message Transcriptions.....	201	200

5.5.1	<i>DTRreqs issued in response to MRDreqs.....</i>	201 200
5.5.2	<i>Native Completion Responses for Reads.....</i>	202 201
5.6	STRreq and Operation Completions	203 202
5.6.1	<i>Coherent Read Operations.....</i>	203 202
5.6.1.1	ReadNITC	203 202
5.6.1.2	ReadNITCCI	203 202
5.6.1.3	ReadNITCMI	203 202
5.6.1.1	ReadUnique	203 202
5.6.1.2	ReadClean	204 203
5.6.1.3	ReadValid or ReadShared	204 203
5.6.1.4	ReadNotSharedDirty	204 203
5.6.2	<i>Dataless Operations</i>	205 204
5.6.2.1	CleanValid or CleanShared.....	205 204
5.6.2.2	CleanSharedPersist	205 204
5.6.2.3	CleanInvalid	205 204
5.6.2.4	Makelnvalid.....	205 204
5.6.2.5	MakeUnique.....	206 205
5.6.2.6	CleanUnique	206 205
5.6.3	<i>Writes.....</i>	207 206
5.6.3.1	WriteUniqueFull, WriteUniquePtl, WriteUniqueStashFull, and WriteUniqueStashPtl	208 207
5.6.3.2	WriteBackFull, WriteBackPtl, WriteCleanFull, WriteCleanPtl, WriteEvictFull	208 207
Chapter 6:	Consistency Processing	209 208
6.1	Concepts.....	210 209
6.1.1	<i>Global observability of operations</i>	210 209
6.1.1.1	Indication of observability requirement for operations	211 210
6.1.2	<i>Coherence ordering</i>	211 210
6.1.2.1	Execution of coherent operations	212 211
6.1.3	<i>Indication of global observability over native interfaces.....</i>	212 211
6.1.4	<i>Ordered Write Observation</i>	212 211
6.2	Consistency mechanisms in CCMP.....	214 213
6.2.1	<i>Response Level: RL[]</i>	214 213
6.2.2	<i>CMDrsp</i>	214 213
6.2.3	<i>STRreq</i>	214 213
6.2.3.1	Execution of operations in sequential order.....	215 214
6.2.3.2	Use of STRreq for Data Transfers.....	215 214
6.2.3.3	Use of STRreq in operation completion indication.....	216 215
6.2.4	<i>STRrsp</i>	217 216
6.2.5	<i>UPDrsp.....</i>	218 217
6.2.6	<i>SNPrsp</i>	218 217
6.2.7	<i>DTRrsp</i>	218 217
6.2.8	<i>DTWrsp</i>	219 218
6.2.8.1	DTWrsp for DTWMrgMRDs	219 218

6.3 Component Requirements	220219
6.3.1 DCE requirements	220219
6.3.1.1 Coherence ordering and processing in DCE	220219
6.3.1.2 Consistency considerations for coherent accesses	222221
6.3.1.3 Consistency rules for DCE.....	222221
6.3.2 DMI consistency requirements.....	224223
6.3.2.1 Consistency rules for DMI	224223
6.3.2.2 Enforcing CMO operation semantics at DMI.....	225224
6.3.3 DII consistency requirements.....	226225
6.3.3.1 Consistency rules for DII.....	229228
6.3.4 AIU requirements.....	230229
6.3.4.1 Times of operation completions on native interfaces.....	231230
6.3.4.2 Informing DCE of completions of operations at the agent.....	232231
6.3.4.3 Special handling of selected operations at the AIU	232231
Chapter 7: DVM Op Processing	233232
7.1 Introduction.....	234233
7.2 CCMP Message Flows for CHIDVMOps.....	234233
7.2.1 CCMP Request Message Flow for CHI Requester for non-Sync or Sync DVMOp.....	235234
7.2.2 CCMP Request Message Flow for ACE Requester for non-Sync DVMOp	236235
7.2.3 CCMP Request Message Flow for ACE Requester for Sync DVMOp.....	237236
7.2.4 Sync or non-Sync DVMOp Snoop Process for a CHI snooper.....	238237
7.2.5 Non-Sync DVMOp Snoop Process for an ACE or ACE-Lite snooper	238237
7.2.6 Sync DVMOp Snoop Process for an ACE or ACE-Lite Snooper.....	239238
7.3 DVM Message Contents.....	241240
7.3.1 DVMOp CMDreq Message for CHI and ACE.....	241240
7.3.2 DVMOp CMDreq Message for ACE.....	243242
7.3.3 CCMP DVM SNPreq Messages.....	244243
7.3.3.1 DVMOp Identifier	244243
7.3.3.2 DVM SNPreq Identifier	244243
7.3.3.3 Contents of the Addr[] fields for the two SNPreq messages	244243
Chapter 8: DataTransfers	248247
8.1 Preliminaries and Definitions	249248
8.2 Data Alignment on Buses.....	250249
8.3 Data Identification	251250
8.3.1 DWID identification.....	251250
8.3.2 DataID identification	251250
8.3.3 CCID identification.....	251250
8.4 Data transfer formats.....	251250
8.4.1 DTR requests.....	252251
8.4.2 DTW requests	255254

8.4.3 A few reminders	256	255
Chapter 9: Miscellaneous Notes.....	257	256
9.1 CHI Support	258	257
9.1.1 RetToSrc.....	258	257
9.1.2 Forwarding snoop	258	257
9.1.3 FWDNodeID	258	257
9.1.4 DoNotGoToSD	258	257
9.1.5 LikelyShared	258	257
9.1.6 AllowRetry	258	257
9.1.7 DoNotDataPull.....	258	257
9.1.8 FwdState	258	257
9.1.9 DataSource	258	257
9.1.10 DVE Completions.....	259	258
9.1.11 Handling of StashOnces	259	258
9.1.12 DVM payload distribution: VMIDext=VMID[15:8] is a separate field outside of the Addr field.	260	259
9.1.13 EWA for CHI Atomics.....	260	259
9.2 MPF usage	261	260
9.3 I/O AIU Coherency State Transition Tables	262	261
9.3.1 AXI I/O AIU Tables.....	262	261
9.3.2 Proxy Cache state transitions	263	262
Chapter 10: Coherency Directory	272	271
10.1 Organization	273	272
10.1.1 Interfaces	273	272
10.2 Null Filters	276	275
10.2.1 Lookups.....	276	275
10.2.2 Commits	277	276
10.2.3 Tag Filters.....	277	276
10.2.4 Lookups.....	278	277
10.2.5 Allocations.....	280	279
10.2.6 Commits	281	280
10.2.7 Deallocation.....	282	281
10.3 Ownership Filtering	282	281
10.4 Coherency Processing in DCE	283	282
10.4.1 System Directory Lookup.....	283	282
10.4.2 RecallTransactions.....	284	283
10.4.3 Address Collision Check.....	284	283
10.4.4 System Directory State	285	284
A.1 CMDreq mappings	287	286

Chapter 1: Introduction

The Concerto-C Messaging Protocol (CCMP) specifies a communication architecture for designing an efficient, scalable and distributed on-chip or multi-chip multi-core system.

The architecture is comprised of specification of communication primitives in the form of messages which can be used to achieve the communication needs of the system's components. The architecture also specifies message sequences that could be undertaken by various components to enforce cache coherency.

The architecture anticipates that standard system components such as processor cores, peripheral devices, memory components, etc. to connect via their native interfaces to a set of specialized hardware blocks specified by the architecture. It is anticipated that these blocks together form a sub-system within the multicore system and are connected to each other via a transport facility to carry architectural messages between them.

The architecture specifies how these blocks communicate with each other via multiple types of message types to achieve the communication requested by the system's standard components.

The goal of the architecture is to provide a message communication regime that is agnostic of specific native protocols, and yet have semantic richness to functionally to be able cover semantics of all the necessary native protocols. Specifically, CCMP is currently defined to cover the following ARM protocols¹:

- AXI-4
- ACE
- ACE-Lite
- ACE5-Lite
- CHI-B and CHI- E

¹ These protocols are only referenced herein. For their details, refer to the respective Architecture manual from ARM Inc.

1.1 Native Operations and Transactions

1.1.1 Operations

Processors execute programs, which are comprised of a sequence of instructions. These instructions invoke primitive **operations** the processors must execute. While many of the operations occur all within the processor, some involve interactions with the system. Examples of these operations include simple Reading and Writing of storage locations but can also include more sophisticated actions depending on the semantics of the instruction being executed and the properties associated with the storage being accessed. For example, such operations can be related to cache management or synchronization, which can affect states of other agents in the system, as a side effect, as is the case in **coherent** Reads and Writes.

Some other agents in the system may perform similar operations without having those being necessarily caused by the execution of a program.

The system itself may also initiate secondary operations aimed at target devices on behalf of agents, such as a Read to a memory controller on an AXI interface. It may also issue operations to agents, as in the case of snooping for achieving coherency.

Interactions with the system implied by an operation are carried over the **native interface** of the agent or target device.

The operation initiated by an agent or by the system on a native interface is called a **native operation**.

1.1.2 Transactions

A single native operation typically involves multiple interactions on the native interface. The individual interactions carried out by the agent or the system over agent's native interface in order to perform a native agent operation are called transactions. An agent transaction is any transaction initiated by an agent on its native interface. A target transaction is any transaction initiated by a target device on its native interface. A system transaction is any transaction by the Concerto-C system on a native interface.

A native agent begins its operation with a **request transaction** that fully expresses the original operation. It carries the operation code and additional information that together completely define the semantics of the operation, including the **type of storage accessed** or referenced and other **attributes of the operation**, if any.

Execution of the agent's request may involve several transactions along the way, but also involves a **response transaction** to the native agent.

A **native snoop transaction** is initiated by a C-AIU on its native interface to perform a coherency operation on a coherent agent.² Snooping interaction begins with a **snoop request transaction** and ends with a **snoop response transaction**

² Native snoop transactions are also issued for Distributed Virtual Memory (DVM) operations, which a NC-AIU is also capable of doing so.

1.2 Concerto-C Messaging Protocol

1.2.1 The Ncore platform

Ncore 3 is a *platform* of connected architectural units, called **Ncore-3 units**, which maintain coherence among the IP blocks by exchanging coherence protocol messages, specified herein, on their behalf. This set of architectural units implemented in hardware forms a hardware coherency *platform* around which a cache-coherent multiprocessor system can be built.

Native snoop transactions are also issued for Distributed Virtual Memory (DVM) operations.

The Ncore 3 *platform* has the following component units:

AIU: These units offer *native interfaces* to which external requester devices such as coherent processors and non-coherent peripherals can be attached. The receive request transactions and initiate message activity among the Ncore 3 platform units to execute semantics of those requests.

DCE: These units orchestrate the coherency semantics among the coherent requesters and system caches internal to the Ncore 3 platform for transactions that are indicated to require such enforcement.

DMI: These units are *target devices* within the Ncore 3 platform that offer *native interfaces* to system memory that can be accessed coherently. The system memory can also be accessed non-coherently.

DII: These units are *target devices* within the Ncore 3 platform that offer *native interfaces* to storage in the system that may be accessed only non-coherently.

DVE: These units are target devices for *DVM* transactions initiated by coherent requesters in the system. These units also orchestrate messages to AIUs for executing the semantics of these transactions.

1.2.1.1 The CCMP Domain

CCMP Domain refers to a region containing all the hardware units and functional blocks whose behavior is influenced by the CCMP protocol. These typically include all the Ncore 3 units, plus the coherent agents, contents of whose cache hierarchies, as well as those of Proxy Caches in IO-AIUs, are influenced by the protocol. The CCMP Domain further includes processor MMU TLBs and SMMU TLBS that can also be influenced via CCMP.

The *CCMP Domain* is also referred to as the **coherency domain** in this document.

1.2.2 Overview of the Concerto-C Messaging Protocol

The Concerto-C Messaging Protocol (CCMP) specifies communication primitives in the form of messages that are exchanged by Ncore units to accomplish native agent operations while maintaining hardware cache coherency in a distributed and scalable manner.

1.2.2.1 CCMP Messages

A **Concerto-C protocol operation** is initiated by a Concerto-C AIU to service a native agent operation as expressed in the native request transaction. The execution of a protocol operation consists of multiple **protocol messages** of different types exchanged among various Ncore 3 units and may also require one or more native transactions to occur.

1.2.3 Initiators and Targets

The agent that issues a native operation is referred to as the ***operation initiating agent***. The device that issues the native transaction is referred to as the ***transaction initiating device***. A device that receives a transaction is called ***transaction target device***. The Ncore 3 unit that issues a Concerto-C message is referred to as the ***initiator*** of the message and the one receiving is referred to as the ***target*** of the message.

1.2.4 Data and non-Data message forms

CCMP messages come in two forms:

- A ***data message (DM)*** which carries data. ***DTR messages*** carry data to requesting AIUs for Reads, and ***DTW messages*** carry data from AIUs to DMIs, DII, etc. for operations, such as Write operations.
- A ***non-data message (NDM)*** which does not carry data, but carries ***control*** or other ***protocol signaling information***, such as address, message attributes, success or failure indication, etc. most messages.

1.2.5 Request and response messages

Each protocol message is designated as either a ***request message*** or a ***response message***. A given request message is either of DM or NDM format, but a response message is always of NDM format.

Flow of request messages among Ncore 3 units is controlled via end-to-end credits. Thus, there must exist at least 1 credit for any request message before it can be sent out by an Ncore 3 unit. Transmission of a request message decreases by 1 the number of credits available for that type of request at the sender.

Each request message causes a response message to be generated by the target of the request message.

Reception of a response thus nominally returns the credit for another similar request message. However, the CCMP protocol makes it possible to indicate what additional protocol event, from which level of the protocol, should be associated with the response.

Response message transmission is not controlled by credits. They are always expected to be accepted by their target unit.

1.2.6 Concerto-C Messaging Protocol Stack

A Concerto-C messaging protocol is a layered protocol that enables Ncore units to communicate using messages. The functionality of the CCMP protocol is divided into 3 layers, Operation semantic, Transport, and Network, all of which are implemented in hardware:

- ***Operation Semantic Layer:*** In this layer resides the understanding of the semantics of the native and/or CCMP operations that may occur in the system. It also contains the comprehension of the coherency protocol. This layer implements hardware data structures, combinational logic, and state machines to orchestrate the execution of these operations.

Every Ncore unit implements the subset of the functionality of this layer that is relevant to the unit. For example, in an AIU this layer contains all the hardware necessary to execute native agent operations and native transaction sequences needed to be conducted for them for the native interface the AIU offers. It also incorporates the knowledge of the corresponding CCMP operations and of issuing appropriate CCMP message sequences required to conduct them in the Ncore system. It also contains the capability to act on snoop messages sent by DCE units in the system. Within the AIUs, this layer also incorporates the system address map to analyze the addresses associated with native agent operation to determine destinations of

CCMP messages it will use.

The DCE unit, in turn, for example, incorporates the semantic understanding of the coherent CCMP operations and the CCMP message activity necessary to enforce coherency.

- **Transport layer:** The primary responsibility of this layer is control of flows of different types of messages to and from other Ncore units. The flow control is based on a system of **credits**.

The CCMP messages are broadly distinguished as **Request messages** or **Response messages**. As a mnemonic, string “**req**” or “**rsp**” suffix is added to the message type, e.g., CMDreq and CMDrsp.

Request messages consume credits and response message return credits. Response messages themselves do not require credits and are required to always be accepted and consumed by the receiving Ncore unit when they arrive.

The Transport layer is also responsible for assignment of Traffic Tier values to various message types to prevent message deadlocks.

- **Network layer:** This layer communicates to the Transport layer of the Ncore unit on one side and the Ncore Concerto Transport Fabric (CTF) on the other side via the SMI TX and RX interfaces.

This layer translates logical unit identifiers into FUIDs and constructs headers for message being transmitted by the unit. For the messages being received from the CTF, the layer checks for CTF-indicated errors and forwards the messages to the Transport layer of the unit.

The next two layers of the overall communication protocol stack are not part of the CCMP but are supplied by the Concerto Transport Fabric.

- **Data-Link Layer:** This layer talks to the CCMP protocol layers. Its functions are to compute the route the message will take through the CTF and packetize the messages arriving over the SMI TX interface before transmitting over the fabric. For incoming packets, to de-packetize them into messages and deliver them to the Network layer of the CCMP stack via the SMI RX interface.
- **Physical Layer:** This layer is also part of the CTF and handles the transmission of packet information along the components of the switching network.

1.3 Ncore 3 system overview

The Ncore system enables cache coherent systems to be built from coherent and non-coherent IP blocks using standard interfaces such as AXI, ACE, CHI, etc. These IP blocks are connected to Ncore-3 units via these standard interfaces. Ncore-3 units themselves are connected via an optimized transport fabric, called **Legato**, which carries the CCMP messages between the Ncore units.

1.3.1 Agents and Targets

IP blocks in the system connect to Ncore-3 units to access system storage locations and to perform other related operations.

1. Each IP block in a Ncore 3-based system that **initiates** communication into the Ncore 3 system to access storage is called an **agent** or **agent device**. An agent that may be classified as either fully coherent, IO-coherent, or non-coherent depending on how the agent consumes or modifies data with respect to copies of data in caches and in main memory. An agent **consumes** data when it reads data values, e.g., executes a load instruction, and **modifies** data when it writes data values, e.g., executes a store instruction. A **fully coherent agent**, e.g., a processor cluster with an ACE interface, consumes or modifies data by issuing coherent transactions and may cache copies of data that must remain coherent with

respect to transactions from other agents. An **IO-coherent agent**, e.g., a GPU with an ACE-lite interface, consumes or modifies data by issuing coherent transactions but does not cache copies of data that must remain coherent with respect to transactions from other agents. Finally, a **non-coherent agent**, e.g., a DMA with an AXI interface, primarily consumes or modifies data by issuing non-coherent transactions directly to main memory.

A fully coherent agent is also known as a **caching agent**, while an IO-coherent agent or a non-coherent agent is also known as a **non-caching agent**. The Concerto-C protocol extends the definition of caching agents to entities in the system that behave like fully coherent agents.

Each agent implements a **native agent interface**, such as CHI-B, CHI-E, ACE, ACE5-Lite, ACE-lite, or AXI, although other interfaces may be supported. An agent may implement one or more native agent interfaces.

Some other IP blocks in the Ncore 3-based system hold accessible storage locations. These blocks don't initiate communication into Ncore 3 system, but instead **receive communication** from the Ncore system to access storage locations within them. Such IP blocks are referred to as **targets** or **target devices**.

1.3.2 Concerto-C Units

1.3.2.1 AIUs

An agent performs coherent and non-coherent accesses through one **Agent Interface Units (AIUs)**, which exposes a single native interface to the agent. Concerto-C also supports a model in which a single agent connects to more than one AIUs, each of which offers a single native interface. In this configuration, each cacheline address is associated with a single AIU for a given agent. For an agent configured with multiple AIUs, cacheline address are distributed across the AIUs using an *implementation-defined* mapping function, e.g., address bit interleaving or hashing, that routes all transactions to and from the agent based on the cacheline address. In other words, with respect to a given agent, all transactions that operate on a cacheline address *must* be performed via a single AIU.

An AIU may be configured in a system in one of several ways:

- A single fully coherent communicates directly with one or more AIUs through the native agent interfaces. The AIU in this case is classified as a **Coherent-Agent AIU (C-AIU)**.
- A set of IO-coherent agents communicates directly or indirectly with an AIU through the native agent interfaces of ACE-Lite or ACE5-Lite. The AIU in this case is classified as a **Non-Coherent-Agent AIU (NC-AIU)**.
- A set of non-coherent agents communicates directly or indirectly with one or more AIUs through the native agent interfaces of AXI. At times, such an AIU may be configured to generate coherent accesses on behalf of non-coherent agents. The AIU in this case is also classified as a **Non-coherent-Agent (NC-AIU)**.

In all cases, the AIU connects on the other side to the message transport fabric via one or more **Transmit (TX)** and **Receive (RX) Symphony Message Interface (SMI)** ports. The coherent and non-coherent transactions issued by an agent are translated by the AIU into Concerto-C messages and transmitted to other Concerto-C units as necessary.

An AIU may also be configured to implement a cache internally for use by an IO-coherent or non-coherent agent. In this case, the AIU acts as a caching agent on behalf of the IO-coherent or non-coherent agent.

To support different agent interface behaviors, the Concerto-C architecture specifies a different AIU for each type of native agent interface.

1.3.2.2 DCEs

Cacheline copies are kept coherent by one or more **Distributed Coherence Enforcement Units (DCEs)**. Each cacheline address is associated with a single DCE, known as the **Home DCE** for that cacheline address.

The Home DCE acts as the **point of serialization (PoS)** and **point of coherence (PoC)** for, and enforces coherence on, the given cacheline address. In a system configured with multiple DCEs, cacheline address are distributed across the DCEs using an *implementation-defined* mapping function, e.g., address bit interleaving or hashing, that routes coherent memory accesses to the appropriate Home DCE based on the cacheline address. Internally, a DCE may implement a system directory partition for tracking and managing the contents of caching agent caches with respect to the addresses associated with the DCE.

Externally, a DCE connects to the transport interconnect via one or more SMI TX and RX interfaces.

Note: The DCEs may also be configured without a directory, effectively “broadcasting” snoop messages to all caching agents. Hybrid directory and broadcast configurations are also possible.

1.3.2.3 DMIs

System memory locations are accessed through one or more **Distributed Memory Interface Units (DMIs)**. Each cacheline address is associated with a single DMI, known as the **Home DMI** for that cacheline address.

The Home DMI is responsible for preserving the order of accesses from Concerto-C units to a given cacheline address and determining the visibility of data from those accesses; in particular, a DMI represents the coherence domain visibility point and controls access to the system visibility point. In a system configured with multiple DMIs, cacheline address are distributed across the DMIs using an *implementation-defined* mapping function, e.g., address bit interleaving or hashing, that routes coherent memory accesses to the appropriate Home DMI based on the cacheline address. The distribution of cacheline addresses across DMIs may differ from the distribution across DCEs; for example, DMI address bit interleaving may be different from DCE address bit interleaving.

On the transport interconnect side of the unit, a DMI implements one or more SMI TX and RX interfaces while on the system memory side of the unit, a DMI implements a **native memory interface**, e.g., AXI.

Note: Depending on the features implemented by the DMI, the Concerto-C architecture will specify different versions of a DMI.

1.3.2.3.1 System Memory Cache

A DMI may optionally hold a **System Cache** unit called the **System Memory Cache (SMC)**. The SMC acts as an in-line cache to system memory and helps to improve latency and bandwidth characteristics for accesses to it. By reducing traffic to memory devices, the SMC also reduces power expenditure for memory accesses.

1.3.2.3.2 Prefetch Buffers

A DMI may optionally implement a separate and independent buffer for prefetched cachelines. Prefetched cachelines shall be held there instead of installing them in the SMC, unless a specific request is made to allocate them in the system cache.

1.3.2.4 DIIs

Peripheral storage locations are accessed through one or more **Distributed I/O Interface Units (DIIs)**. Each peripheral address is associated with a single DII, known as the **Home DII** for that address. The Home DII is responsible for preserving the order of accesses from Concerto-C units to a given address and determining the visibility of data from those accesses; in particular, a DII represents the peripheral storage visibility point and controls access to the system visibility point. Unlike the system memory, addresses are not interleaved across multiple DIIs. Thus, a given storage associated with a given peripheral device is accessible only via a single home DII.

On the transport interconnect side of the unit, a DMI implements one or more SMI TX and RX interfaces while on the system memory side of the unit, a DMI implements a **native memory interface**, e.g., AXI.

1.3.2.5 DVE

Distributed Virtual Memory Engines (DVE) is a block that coordinates coherency operations on TLBs, both within processor and System MMUs. DVE is connected to the transport fabric via one or more SMI TX and RX interfaces.

Finally, a **coherence domain** is defined to be a set of Concerto-C units, this includes all agents attached to AIUs and system memory connected via DMIs among which accesses to system memory are kept coherent through a set of DCEs. A Concerto-C-based system may consist of any number of coherence domains, and the sets that define different coherence domains must be disjoint with respect to each other.

1.3.3 Legato: The CCMP transport fabric

The Ncore 3 system uses a special purpose fabric, optimized to carry CCMP messages among Ncore units. It has the following salient features.

- Packetized transmission of messages

This allows messages to be encapsulated into packets that can be suitably formatted for Power-Performance trade-offs. It also makes possible the use of common network components to be used for multiple types of messages, potentially via multiple networks.

- Light-weight: Legato implements a pure data-link level transport facility. The packet protocol does not incur Network and Transport layer information in the packets, thereby minimizing packetization overhead

The actual CCMP messages from one Ncore unit to another form the application layer of the CCMP protocol stack. The upper layers of the protocol stack such as Network, Transport, etc. are implemented in individual Ncore units.

- Pure transport. Does not impose additional semantics or requirement of packet types. Neither interprets CCMP messages
- Flexible specifiable data arrangements to minimize latencies and maximize bandwidth in presence of width change
- Ability to change widths of network paths
- Ability to define just the packet as needed
- Arbitrary topologies that can be tuned to system requirements of performance and cost.
- Optimized for CCMP messages.
 - Simple yet flexible message types to accommodate the needs of CCMP messages
 - Standardized light-weight interface for Ncore units to receive and transmit messages (SMI)
 - Flexible message contents without transport interpreting the message contents
 - Data and non-data messages
 - Variable sizes for each. CCMP message can vary greatly in size
 - Ability to implement means for ensuring guaranteed delivery or signal error. (response messages.)
 - Architectural means to prevent protocol-level deadlocks (TTier)
 - Support for VCs to implement architectural scheme of avoiding deadlocks
 - Ability to request steering of traffic (Steering) along appropriate paths
 - Ability to express Priority level
 - Ability to express other QoS requirements, such as bandwidth or periodicity
- Ability to define Multiple networks
- Ability to specify message paths
- Scalable fabric
- Ability to connect to multiple networks
- Ability to connect to a network via multiple ports
- Ability to create system wide error architecture that includes and seamlessly incorporates detection

- and signaling of transport errors
- Ability to perform PPA trade-offs

1.3.4 Unit Identifiers

CCMP uses integers as ***unit identifiers*** to specify the sending Ncore unit of a message and to specify the Ncore unit to which the message should be delivered. These identifiers are used by the message transport facility to determine the route the message should take through Legato.

Each caching agent and Concerto-C unit are assigned identifiers for the purposes of tracking cache state and routing protocol messages. A caching agent is identified by a unique ***CacheID***, which represents a single logical caching entity that may be partitioned into one or more multiple physical caching structures. The protocol uses the CacheID to track the state of a particular caching agent's cache in the system directory.

For the purposes of Concerto-C protocol, every Concerto-C unit is identified by a Unit Identifier (Uid). The identifiers for AIUs are referred to as AIUIDs, for DCEs as DCEIDs, for DMIs as DMIIIDs, for DIs as DIIDs, and for DVE as DVEID, respectively.

Generally, if A equals the number of AIUs, D is the number of DCEs, M is the number of DMIs, and I is the number of DIs in the Ncore system, units in the system are assigned identifiers in the Uid name space as follows:

- 0 to $A-1$: AIUIDs for AIUs
- A to $A+D-1$: DCEIDs for DCEs
- $A+D$ to $A+D+M-1$: DMIIIDs for DMIs
- $A+D+M$ to $A+D+M+I-1$: DIIDs for DIs, and
- $A+D+M+I$: DVEID for DVE.

Every Concerto-C unit is also identified by a unique ***Fabric UnitID (FUID)*** across the entire Ncore system. The Concerto-C protocol uses the FUID to route protocol messages on the transport interconnect to the designated unit.

A CacheID is associated with an AIU and a mapping exists from the CacheID to the FUID of the corresponding AIU.

1.3.5 Ncore system organization

Figure 1-1 shows a schematic representation of the organization of an Ncore 3 system. The box labeled Concerto-C Domain depicts the boundary of the Ncore system. All elements in the figure that are outside the said box are external to the Ncore system.

The figure depicts one or more instances of the various types of AIUs discussed above connected to the Concerto Transport Fabric (CTF).

The coherent processors or accelerators or clusters thereof are connected individually via a native interface offered by ACE, teLi, or CHI-B AIU.

One or more non-coherent agents, typically peripheral devices that don't possess coherent cache hierarchies and carry interfaces such as AXI, may optionally be first connected via an external network to concentrate their traffic, and then be connected to an AXI-AIU. This traffic may optionally be intercepted by

an SMMU before getting to the AIU. The SMMUs may optionally interface with an ACE-Lite or an ACE5-Lite AIU, depending on the desired functionality. ACE5-Lite interfaces enable peripherals to perform stashing and Far Atomic operations.

The figure also shows the inclusion of a CXIU to enable coherent connection to another SoC via CCIX.

The CTF is comprised of one or more independent transport networks that carry Concerto-C message traffic that is generated in response to native request transactions issued by agents to the AIUs. The connectivity capabilities, performance parameters, physical placement, and topology of these networks are customizable per the needs of the SoC.

The target of an agent's transaction could be system memory or peripheral storage. System memory is accessed via one or more DMI units. System memory address ranges can be interleaved across sets of DMIs.

One or more DII units are used to access all peripheral storage in the system.

The Ncore 3 system supports the ARM DVM architecture operations including required interactions with the SMMUs in the SoC. The DVE unit coordinates the execution of the DVM protocol.

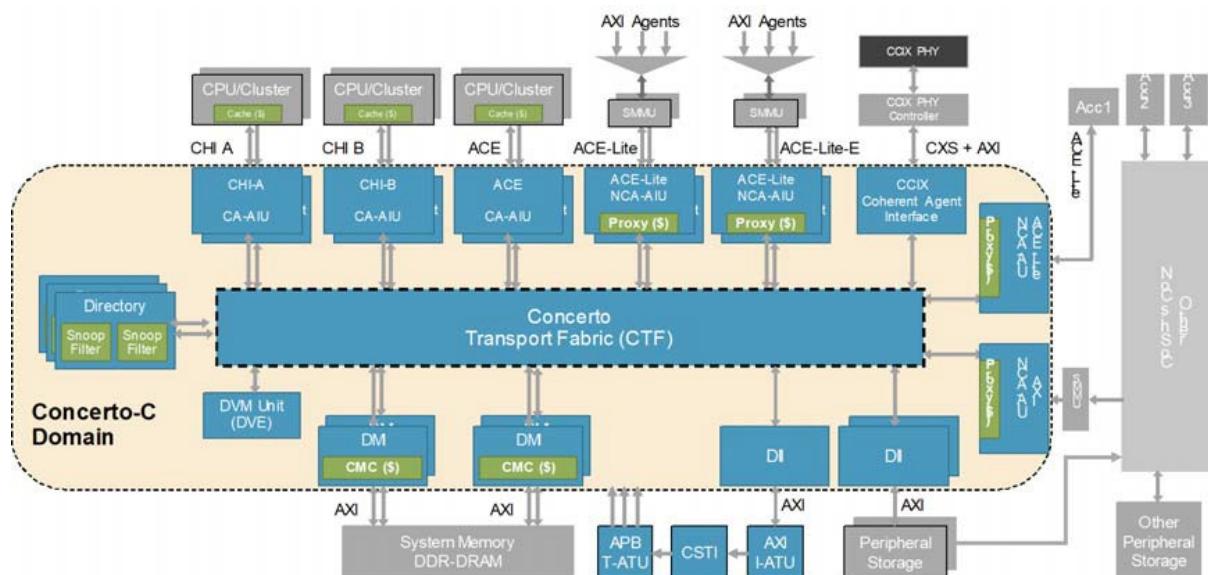


Figure 1-1: Ncore 3 system organization

Also shown in the figure is a schematic representation of a configuration network meant to access configuration and status registers within Ncore units that is outside the Ncore domain. It is accessed via a DII, and involves an AXI Initiator ATU, an ATP transport interconnect, and one or more APB Target ATUs. The APB buses generated by the APB Target ATUs connect to the APB configuration interfaces of the Ncore units.

Chapter 2: Storage Architecture

This chapter specifies the storage architecture supported by the Concerto-C Messaging Protocol.

2.1 Storage Classification

For coherency purposes, Concerto-C protocol classifies any form of storage within an SoC-based computing system as being one of two types:

- System memory, and
- Peripheral storage

System memory is commonly implemented via Dynamic or Static RAM devices. Peripheral storage typically includes control and status registers within peripheral devices, although it might also include device-internal data buffers as well as stand-alone RAM devices.

In a Concerto-C based system, **system memory** refers to storage that is characterized as being **well-behaved**.³ For such storage a Write to a given memory location updates the value stored at that location and a Read from a memory location returns the last value written to that memory location. Furthermore, performing an access to system memory does not have side-effects.

Because of the well-behaved nature of system memory, blocks of it can be freely accessed without side effects. An access of a small number of bytes can be translated to an access of a larger sized block and stored in a cache taking advantage of spatial locality and improve latency for future accesses and to improve data access efficiency and bandwidth. Well behaved storage, and thus system memory, has the property of being **cacheable** storage.

Not all **Peripheral storage** is guaranteed to be well-behaved. In some of such storage locations, data values may not be persistent, e.g., two reads without an intervening write may not return the same value, or a read may not return the last value written. Additionally, a read or write access may cause side-effects. Also, this storage may be sparse such that in each aligned 64-byte block of addresses, for example, not all bytes may have actual physical storage to hold data. Writing data to such non-existent storage location results in loss of that data. Reading such location often returns some fixed or potentially random value.

On the other hand, some *peripheral storage* may indeed be well-behaved. This type of storage is distinguished from non-well-behaved storage by referring to it as **peripheral memory**.

Although well-behaved, such memory can still be accessed only non-coherently via CCMP. Concerto-C Messaging protocol enables access to all addressable storage but maintains coherency only for system memory locations.

2.2 System storage and access sizes

Concerto-C system recognizes data being stored and accessed in terms of 8-bit bytes. A byte is stored at a **physical site** in a **physical device** in the system. These are called the byte's **home location** and **home device**, respectively. In the context of the Ncore 3 system, the target unit (DMI or DII) to which the location is mapped is called the location's **home unit**.

During a given operation of a Concerto-C system, each such home location has one-to-one association to a unique whole number. The whole number associated with a given location is referred to as its **address**. In practice, only a finite subset of whole numbers is utilized as addresses. It is not required that whole numbers utilized in the system form one contiguous set.

Addresses are commonly expressed as binary or hexadecimal numbers.

³ Well-behaved storage is also referred to as “normal storage” or “normal memory”

A set of bytes with a contiguous set of addresses is referred to as a **block** or **granule** of data, the smallest block being 1 byte. Bytes within such a block are ordered, with ones with lower address value as being **less significant** and those with higher address values as being **more significant**.

An access in Concerto-C can be made only to a *block* or *granule* of data. The number of bytes so accessed is referred to as the **size** of access.

The smallest access size is 1 byte, although most accesses refer to blocks of larger size. The address associated with an access is defined to be the address of the **least significant byte** of the block. The **least significant byte** of a block is also referred to as the **starting byte** of the block.

An access is said to be **aligned** to size n if its address has 0s in its least significant n places.

An **aligned block** of data is of size 2^n such that it has 0s in the n least significant bits of its address. Thus, a byte is always aligned.

Note that the **ending byte** of an *aligned block* of size 2^n is the byte that has 1s in the n least significant bits of its address. It is the most significant byte within that *aligned block*.

Given any address A, an aligned block that contains the byte with address A, is called a **container aligned block** of address A.

An access is defined to be **aligned access** if it is of size 2^n and the least significant n bits of its address are 0. Thus, a byte access is always an aligned access. Thus, an *aligned access* accesses an *aligned block* of data.

The following terminology is commonly used in this architecture:

n = 0: 1 Byte

n = 1: Half-word (= 2 Bytes) n = 2: Word (= 4 Bytes)

n = 3: Double-word (= 8 Bytes)

n = 4: Quad-word (= 16 Bytes)

n = 5: Oct-word (= 32 Bytes)

When devices make accesses in a Concerto-C system, they indicate the logarithm of the size, **n** and the address of the block accessed. The address itself is not required to be aligned to the size of the access and can be aligned to a byte-sized block. The address of the 2^n -byte block accessed is computed by masking out the least significant n bits of the address. Thus, for example, an access specifying n = 5 and address 0xAB CDEF 1235 refers to an aligned 32byte block located at 0xAB CDEF 1220.

2.2.1 Atomic Accesses

An access is said to be performed **atomically** if the access is made as a unit and without interference from other accesses to the same data block with respect to data content. Thus, if two Write accesses A and B are made to the same block of data to deposit distinct values *a* and *b*, respectively, the value deposited in the block at the end of both accesses A and B is either *a*, or *b*, and not a combination of *a* and *b*.

In Concerto-C, only aligned accesses can be assumed to be **atomic**. Such an access is referred to as **Atomic Access (AA)**.

The amount of data accessed via an atomic access is also termed **Atomic Block (AB)**, since it can be accessed within the Concerto-C system in a single uninterrupted action. The size in number of bytes of an *atomic block* is called **Atomic Size (AS)**.

Any block of memory that is not atomic is referred to as **non-atomic**.

Implementation Note

In Ncore 3 any access to a non-atomic block is broken down into multiple atomic accesses.

Atomic access in Concerto-C system is limited to aligned blocks of size 2^n bytes, $n = 0, 1, 2, \dots, N$, where N is the maximum exponent supported for such accesses.

2.3 Concerto-C Memory

System memory is divided into equal-sized data blocks on which coherence is maintained. These blocks are referred to as **coherency granules (CGs)**. Caches in the systems are organized and accessed in units that are called **cachelines**. Concerto-C expects these two units to be identical in size. Because of this, in Concerto-C *coherency granules* are synonymous to *cachelines* and henceforth are also referred to as the latter. Each *cacheline* consists of a contiguous block of a power-of-two number of bytes and is uniquely identified by a **cacheline address**, which is a system memory address aligned to the size, in bytes, of a *cacheline*. A coherent agent may hold a copy of a *cacheline*, or a **cacheline copy**, in its cache.

Architecture Note

Within a Concerto-C system, the *coherency granule* is the largest sized block that can be *atomically accessed*. Any access to a block larger than a coherency granule is broken down into multiple accesses

Implementation Note

In Ncore 3 system coherency granule size is 2^6 or 64 bytes. Any access to a larger block is broken down into multiple accesses.

Memory accesses are classified relative to the size and alignment of a *cacheline*. A memory access that operates on all the bytes within a *cacheline*-aligned data granule is a **full-cacheline access**. A memory access that operates on a subset of the bytes within a *cacheline*-aligned data granule is a **partial-cacheline access**. Finally, a memory access that operates on bytes in two or more *cacheline*-aligned data granules is classified as a **multiple-cacheline access**.

The visibility of data from a memory access from one agent relative to memory accesses from other agents is defined in terms of two logical points in the system. Data are **coherence domain visible** once an access from a coherent agent is visible to all other coherent agents in the coherence domain (see section TBD); in other words, the access has been ordered with respect to accesses to the same address from those coherent agents. The point at which data from a coherent agent is coherence domain visible is known as the **coherence domain visibility point**. Similarly, data are **system visible** once an access from an agent is visible to any other agent; in other words, the access has been ordered with respect to accesses to the same address from all agents. The point at which data from an agent is system visible is known as the **system visibility point**.

When data becomes system visible, it is also assumed in Concerto-C to have reached the **point of persistence**.

Data from certain operations are allowed to be coherence domain visible, while data from others are required to be system visible or to have reached the point of persistence. Furthermore, copies of data that are coherence domain visible may not be consistent with copies of data in system memory. Mechanisms are provided to ensure that data are system visible so that communication may occur between coherent and non-coherent agents.

Non-system memory is not guaranteed to behave as normal storage. As a result, data values may not be persistent, e.g., two reads without an intervening write may not return the same value, or a read may not

return the last value written. Additionally, a read or write access may cause side-effects.

The Concerto-C **system address space** consists of the set of addresses that map to system and non-system memory locations accessible by all fully coherent and IO-coherent agents. Each address in the system address space maps to a single memory location.

Note

A fully coherent or IO-coherent agent may be able to generate addresses outside the system address space. In addition, a non-coherent agent may *not* be able to generate all addresses in the system address space.

Concerto-C allows system storage locations to be accessed coherently or non-coherently. Hardware can enforce coherence only on system memory locations and only when all agents agree on the access semantics of the memory location. If all agents do not agree on the access semantics, coherence can be enforced through cache maintenance or other techniques. Enforcing coherence on non-system memory locations is beyond the scope of the Concerto-C protocol.

2.4 Coherency maintenance

Processors, typically, have private *caches* or *cache hierarchies* within them for quicker access to data, at a higher rate. By holding copies of often referenced cachelines in their caches, processors are able to mitigate the disparity in the lag with, and the rate at, which they can access data and then process it, and the effective rate that could be achieved if the data must always be retrieved from system memory, instead.

In general, at any given time, there can potentially be multiple copies of the same cacheline in private caches of the processors in the system, and, for the most part, these private copies are accessed privately by the individual processors that hold them.

For the processors to correctly cooperate in each computation, they must have a **coherent** view of the data they share. That is, all copies in the system of a cacheline that is shared by multiple processors must be always kept identical.

No extra effort is needed when the contents of the cacheline are constant. But when the contents change, all copies must be updated together to not violate the *coherency requirement* above.^{4,5}

2.4.1 PoS and PoC

In order to achieve coherency, CCMP orders accesses to a given coherency granule. CCMP employs the following architectural constructs in the context of such ordering.

A **point of serialization (PoS)** refers to the point in the Ncore 3 system at which all accesses from all agents to a given address are serialized, establishing a **total order** on all accesses in a system via the Ncore 3 platform to that address. This order is referred to as the **system order** of accesses. Ncore 3 system ensures

⁴ Almost all current coherency protocols implement a simpler alternative of invalidating all but a single copy and then allowing the update of that one copy, before more copies are allowed to proliferate again from that single updated copy. This trivially preserves the coherency invariance that *all* copies must be identical. Such protocols are referred to as *invalidation-based* coherency protocols.

⁵ For the sake of efficiency, protocols postpone updating the cacheline at its home location in system memory until such a time when a cache holding the modified cacheline must cast it out. Until then, all accesses attempting to access cacheline from system memory are *intercepted* and instead are satisfied from a cache with an updated copy.

that accesses are **performed** in the Ncore 3 system strictly in the *system order*.

The **point of coherence (PoC)** refers to the point in time in the progress of a coherent access to a coherency granule in the Ncore 3 system at which the said access has been performed with respect to all agents that coherently access the said granule. Thus, when an access is completed to the PoC, its effects on the coherency granule, if any, are visible to all coherent agents in the system.

2.4.2 States of a Coherency Granule

The Concerto-C protocol comprehends a generic cache state model that enables support for caching agents with different cache state models.

Each copy of a cacheline may be in one of several “states” defined by the protocol. With respect to a given cacheline copy, these states determine whether an agent may consume or modify cacheline data, and they delegate protocol responsibilities to that agent.

The cache states in the Concerto-C cache state model are based on a set of properties for each copy of the cacheline; these properties are defined as follows:

- **Invalid vs. Valid:** An **invalid** cacheline copy *must* not be accessed by a caching agent, while a **valid** cacheline copy may be accessed by a caching agent. A caching agent may consume cacheline data from a valid cacheline copy but may only modify cacheline data depending on the other properties of the cacheline copy.
- **Owned vs. Non-owned:** An **owned** cacheline copy is a valid cacheline copy that identifies an agent as the **cacheline owner**, who, depending on the other properties of the cacheline copy, is responsible for forwarding data in response to any snoop messages and for updating memory on a replacement. A **non-owned** cacheline copy is a valid cacheline copy that identifies an agent as a **cacheline sharer**, who has no responsibilities with respect to the protocol.
- **Shared vs. Unique:** A **shared** cacheline copy is a valid cacheline copy that may be valid in another agent’s cache. A **unique** cacheline copy is a valid cacheline copy that *must* not be valid in any other agent’s cache. A caching agent may modify cacheline data only if its copy of the cacheline is unique, and that modification may occur without any additional protocol transactions. A unique cacheline copy implies the cacheline copy is also owned.
- **Clean vs. Dirty:** A **clean** cacheline copy is a valid cacheline copy that is consistent with respect to the latest copy of data, which may be present in either another agent’s cache or system memory. A **dirty** cacheline copy is a valid cacheline copy that represents the latest copy of data and is known to be inconsistent with respect to system memory. Upon modifying a clean cacheline copy, a caching agent changes the state to dirty, and upon replacing a dirty cacheline copy, a caching agent must update memory. A dirty cacheline copy implies the cacheline copy is also *owned*.

The operation of updating memory with cacheline data from the dirty copy while retaining the copy in place causes the state to transition to the **Clean** state. This operation is thus termed **cleaning of the cacheline copy to memory**, or simply **cleaning the cacheline to memory**. An explicit operation to achieve such *cleaning* of the cacheline is referred to as a **Clean** operation.

The operation of updating memory with cacheline data from a dirty copy and then invalidating that copy in the cache causes the state of the cacheline in the cache to transition to an *Invalid* state. This operation is called **flushing the cacheline to memory**. An explicit operation to achieve such *flushing* of the cacheline is referred to as a **Flush** operation.

2.4.2.1 The Reservation State

Systems implement **Reservation Protocol** for achieving mutual exclusion primitives such as **semaphores**. Achieving mutual exclusion involves executing a **Load-Modify-Store sequence** in an **atomic** manner on an address location. Once started, the **atomic sequence** can be **interrupted** if another agent successfully performs a **Store** to the address location before the agent attempting the said atomic sequence performs its Store. This successful Store can also be part of the atomic sequence from the other agent.

The reservation protocol is achieved via **Exclusive access operations**. The exclusive access operations are implemented via **Exclusive access transactions**, which themselves are subject to the underlying coherency protocol. The Exclusive Load can get converted to an **Exclusive Read** transaction if the cacheline is not present in the processor's cache and an Exclusive Store can get converted to another appropriate Exclusive transaction.^{6,7}

The Exclusive Read transaction, if successful, returns data and possibly a **permission to start the atomic Load-Modify-Store sequence** within the processor. This permission is registered via a structure called **Reservation**, which participates in the coherency protocol. A Reservation becomes **active** with the successful execution of the Exclusive Load. The Reservation is **lost** or **reset** when an **invalidating snoop** is received by the processor, the said snoop being treated as being indicative of a successful Store by another agent in the system. It may also be reset when the processor executing the atomic sequence completes it without interruption.

It is sufficient to note here that for the sake of coherency protocol, the presence of an **active** Reservation within an agent is treated as there being a valid copy of coherency granule within it, independent of the actual copy of the granule being present in the agent.

The state of Reservation is tracked independently of the granule's states (see above) within the agent, an active Reservation being treated as a **Shared** state of the granule. This state is considered when enforcing state invariance rules of Section 2.4.3.

Table 2-1 lists the legal combinations of cacheline properties and the resulting Concerto-C states, as well as the mapping to the "common" cache state names typically implemented.

Table 2-1: Concerto-C valid cache states

I/V	N/O	S/U	C/D	Concerto-C State	Standard Name	Possible states in other caches
I	---	---	---	IX	Invalid	IX SC OC OD UC UD
V	N	S	C	SC	Shared	Others: IX SC OC OD
V	O	S	D	SD	Owned	Others: IX SC
V	O	U	C	UC	Exclusive	Others: IX
V	O	U	D	UD	Modified	Others: IX

⁶ For the Exclusive Store to be successfully performed, the Reservation must be active and the coherency granule must exist in Unique state within the agent

⁷ Refer to ARM architecture documents for more specific information about the operations and transaction involved and the internal operation of the Reservation protocol

2.4.2.2 State labels in CCMP Directory

CCMP directory tracks agent cacheline states using a three-state model. Table 2-2 below shows the correspondence between Concerto-C States and the states maintained by the Concerto-C directory.

Table 2-2: Correspondence between Concerto-C and Directory states for a cacheline

Concerto-C State	Directory State
IX	Invalid
SC	Sharer
SD	Owner
UC	Owner
UD	Owner

Note

Although in Table 2-1 the SD state alone is labeled as *Owned*, all states with O bit set are considered as **Owner** states in CCMP directory. Thus SD, UC, and UD are all *Owner* states.

Architecture Note

The Concerto-C classification of valid combinations of cacheline copy properties is not the most efficient encoding of cacheline states. Specifically, Concerto-C defines only five valid cacheline states shown above, so where necessary, these states can be encoded in three bits.

Certain cache coherency protocols allow agents, such as processor cores, to claim Unique and Valid copies of cachelines without actual data being present within the cache hierarchy in order to optimize *Store* performance, allowing the processor to deposit bytes in an empty cacheline buffer. Such a state is labeled **UCE**.

CCMP does not track such a state explicitly. Instead, it labels the state as UC, without any negative effect of the efficiency of the protocol.

As Store instructions deposit data into a cacheline in UCE state, the state changes to Unique Dirty. But the cacheline can be only partially populated with valid bytes. Such a state is labeled **UDP**. While CCMP can handle the effects of presence of the UDP state, the protocol does not track the state explicitly; it simply labels the state as UD, without any loss in correctness or efficacy.

UCE and *UDP* are additional allowed *owner* states for agents, their presence is implementation dependent.

2.4.3 Access Rules and State Invariants

To distinguish from Reads and Writes generated by other agents in the system, Reads issued by an agent to its local cache are referred to as **Loads** and Writes issued by an agent to its local cache are referred to as **Stores**.⁸

The following rules are observed in the CCMP domain with respect to system memory accesses by agents.

- Loads can complete in the local cache if the cacheline is in any valid and complete state (SC, SD, UC, or

⁸ The terminology is derived from processors that execute operations of Loads and Stores per instructions they execute. In the case of proxy caches, native transactions that can be completed at and within the proxy cache are considered *local*

UD). Loads can also complete locally with the cacheline in UDP state only if the bytes being accessed are in valid state within the agent's cache hierarchy.

- Stores can complete in the local cache hierarchy *only if* the cacheline is in any one of *Unique* states (UCE, UC, UDP, UD).

If the above rules are not satisfied the agent must issue a native operation into the coherency domain in order to satisfy the above requirements. Thus, if an agent wants to perform a Store in its cache, it must first ensure that the cacheline is in a Unique state; if it is not, it must issue an appropriate operation into the coherency domain to achieve the Unique state for its copy of the cacheline, and then complete the Store.

Note that the agent may also issue a native operation into the system to perform types of coherent operation other than Reads and Writes.

It is common that a processor has a hierarchy of multi-level caches. To maintain coherency, CCMP maintains the following invariants with respect to cacheline states in the various caches in the system.

- At most one cacheline copy in the coherency domain can be in Owned state.
- If a cacheline copy in the coherency domain is in Unique state, no other valid copy may exist in the coherency domain.
- At most one cacheline copy in the coherency domain can be in Dirty state.

From the above invariants, at most one agent in the system may be in any one of the *owner states* for a cacheline at any given time.

If any agent or cache achieves an *owner state* for a cacheline (including UCE and UDP), all other caches and agents in the system must have their copies of the cacheline in I or SC states.

Further, if any agent or cache achieves a *Unique state* for a cacheline (including UCE and UDP), all other caches and agents in the system must invalidate their copies of the cacheline.

Table 2-1 shows for each cacheline state in any cache, possible compatible cacheline states in other caches.

2.4.4 Intervention response

A Store into a copy of the cacheline within a cache makes that copy *dirty*, meaning it is modified with respect to the copy in the cacheline's home location, which is system memory. The dirty copy thus is the most recent copy of the data in the cacheline.

The coherency protocol ensures that a Read issued into the coherency domain receives the latest copy of the cacheline within the coherency domain. Therefore, a Read for a cacheline that has a dirty copy in a cache, for example, must receive its data response from that cache instead of from memory.

The response to external accesses by an agent (or its cache) by supplying cacheline data that supersedes a data response from memory is termed as an ***intervention response***.

The data supplied via an intervention response is referred to as ***intervention data***.

Some coherency protocols allow an intervention response from a *Shared and Clean* copy of the cacheline. However, this can potentially result in multiple caches providing intervention responses for the same operation. It is then left up to the implementation to select which intervention response to use.

Possibility of multiple interventions can complicate system design and increase area. CCMP therefore employs architectural mechanisms to ensure that at most one AIU in the coherency domain supplies intervention data for a given Read operation, including from an SC state.

2.4.5 Ownership levels of cacheline

Different states of the cacheline shown in Table 2-1 represent different **ownership levels** within the cache. The ownership levels determine what types of operation an agent can perform locally within its cache, without having to initiate an operation into the coherency system or is obligated to perform in response to an operation in the system from another agent.

The higher the ownership level, the more the types of local operations that can be executed and more the obligation to respond to external accesses. Table 2-3 shows the cache states in increasing level of ownership, from I to UD.

Table 2-3: Cache states in increasing level of ownership and corresponding capabilities

Cache State	Operation
I	No access to cacheline
SC	Loads In limited cases, may supply intervention data to Reads (only). ^a
SD	Loads Must provide intervention data for external coherent Reads and Partial updates to the cacheline. Must respond to a coherent operation that requires updating the system memory with latest data.
UCE	Stores
UC	Loads and Stores May provide intervention data for external coherent Reads (only) to the cacheline.
UDP	Loads limited to validated bytes of the cacheline. Stores Must provide intervention data for external coherent Reads and Partial updates to the cacheline Must respond to a coherent operation that requires updating the system memory with latest data.
UD	Loads and Stores Must provide intervention data for external coherent Reads and Partial updates to the cacheline Must respond to a coherent operation that requires updating the system memory with latest data.

Note:

a. This type of intervention may be suppressed in certain system implementations

2.4.6 Silent cacheline state transitions within agents

Load and Store operations that can complete within the agent's cache hierarchy do not result in transactions on the agent's native interface. Cacheline state transitions that result from such operations internal to the agent are thus **invisible** or **silent** from the point of view of CCMP protocol and the associated directory keeping track of agents' cacheline states. Based on the agent implementation, there are other actions that may also cause certain *silent* state transitions.

Because of the possibility of silent state transitions, the cache states registered in the directory can temporarily be out of synchrony with the actual state within the agents. However, at the completion of a given native transaction, the directory states achieve synchrony with the states within the agents.

Note that such silent state transitions must still preserve invariants stated in Section 2.4.3.

The CCMP protocol and directories permit the following **silent cacheline state transitions**. See Table 2-4.

No other state transitions to cachelines within an agent are permitted to be silent. Such transitions require explicit requests from the agent or snoops to it that are visible to the CCMP to cause them.

Table 2-4. Silent Cache State Transitions supported by CCMP

From	To	Possible causes
UC	I	Cache Eviction
UCE	I	Cache Eviction
SC	I	Cache Eviction
UC	SC	Local sharing of cacheline within a processor cluster
UD	SD	Local sharing of cacheline within a processor cluster
UC	UD	Store hit in the cache hierarchy (Full or Partial cacheline)
UCE	UDP	Store hit in the cache hierarchy (Partial cacheline only)
UCE	UD	Store hit in the cache hierarchy. (One or more Stores that update all bytes of a cacheline)
UDP	UD	Store hit in the cache hierarchy. (One or more Stores that also update remaining bytes of the cacheline in UDP)
UD	I	Cacheline invalidate (Since local is in UD, no additional copy exists externally that needs to be invalidated.)
UDP	I	Cacheline invalidate (Since local is in UDP, no additional copy exists externally that needs to be invalidated.)

2.4.7 Coherency Protocol Models

Because the Concerto-C cacheline states can be mapped to standard cache states, a Concerto-C-based system can support multiple caching agent models. In effect, the protocol messages modify the cacheline properties accordingly, and any translations between the states implemented by the agents and the Concerto-C states are performed by the AIUs. A fully coherent agent that supports shared cachelines *must* implement the IX and SC states at a minimum; however, a fully coherent agent that does *not* support shared cachelines *must* implement only the IX, UC, and UD states. The legal combinations of states are shown in Table 2-5.

Table 2-5: Cache State Models supported by CCMP

UD	UC	OD	SC	Legal?	Notes
0	0	0	0	Yes	Consistent with an SI cache state model (clean only)
0	0	0	1	No	Ownership without UD is not supported
0	0	1	0	No	Ownership without UD is not supported
0	0	1	1	No	Ownership without UD is not supported
0	1	0	0	No	Ownership without UD is not supported
0	1	0	1	No	Ownership without UD is not supported
0	1	1	0	No	Ownership without UD is not supported
0	1	1	1	No	Ownership without UD is not supported
1	0	0	0	Yes	Consistent with an MSI cache state model
1	0	0	1	Yes	Consistent with MESIF cache state model
1	0	1	0	Yes	Consistent with a MOSI cache state model
1	0	1	1	Yes	Consistent with MOSIF cache state model (but not supported in Concerto-C)
1	1	0	0	Yes	Consistent with an MESI/MEI cache state model
1	1	0	1	Yes	Consistent with MESIF cache state model (but not supported in Concerto-C)
1	1	1	0	Yes	Consistent with an MOESI cache state model
1	1	1	1	Yes	Default for Concerto-C cache state model

To consume data, a fully coherent agent that supports shared cachelines may issue coherent transactions that acquire either a shared or unique copy of the cacheline. On the other hand, a fully coherent agent that does *not* support shared cachelines *must* issue only coherent transactions that acquire a unique copy of the cacheline.

To modify data, a fully coherent agent *must* issue only coherent transactions that acquire or create a unique cacheline.

Depending on the system configuration, the system directory in a Concerto-C-based system tracks a subset of the cacheline properties. In particular, the system directory may track whether a cacheline copy is valid or invalid and/or may track whether a cacheline is owned or non-owned. If the system directory tracks all caching agents in the system, the distinction between a unique copy and a set of shared copies can be made. Most importantly, however, the system directory is not aware of the cache state model of the individual caching agents. See section TBD for more information.

Chapter 3: Concerto-C Operations

This chapter specifies the native operations and transactions supported by Concerto-C Messaging Protocol and describes messages used within it.

In this section and in the remainder of this document, the terms “CCMP” and “protocol,” are substituted for “Concerto-C messaging protocol,” for the sake of brevity.

3.1 Concerto-C Protocol Operations

The Concerto-C protocol supports several **operation types** that can be categorized as coherent or non-coherent. The former cause changes in the state of the caches in the initiating agent and any snooped agents. These state changes also cause corresponding state changes in the system directory. The non-coherent operations do not involve system directory and snooping.

Operation types provided by CCMP are described in Table 3-1.

Note that for coherent operations, the actions taken are with respect to all the agents in the coherency domain and the state of the cacheline refers to either at the requester agent or all agents across the coherency domain.

Each CCMP operation effected in CCMP via a well-defined sequence of messages. The first of these messages is the Command request message (CMDreq) that carries all the relevant and necessary semantic content of the operation.

Table 3-1 shows the CMDreq that is issued for each of the operation. **CMDreq** message contents are discussed in Section 4.5.1.

Table 3-1: CCMP Operations

CCMP Operation (Mnemonic)	C or NC ^a	Operation Description
Read with No Intention to Cache (ReadNITC)	C	<p>Obtain the latest snapshot^b of the cacheline. The operation does not cause the snooped agents to change the states of their copies of the cacheline, although they may change purely due to internal operation of the snoopers' caches.</p> <p>The snapshot is not installed in the CCMP coherency domain. Thus, the snapshot itself is no longer kept coherent after the operation. As a result, the operation <i>does not cause</i> a system directory to allocate a new entry for the cacheline.</p> <p>Note: This transaction corresponds to the ACE and CHI ReadOnce transaction. In ACE, this transaction can request any sized Atomic Block of coherent memory, while in CHI, only cacheline size is supported.</p> <p>Implementation Note: It is up to implementation if ReadNITC of size other than a cacheline is supported. If only cacheline size is supported, it is the responsibility of the requesting AIU to select the requested Atomic Block out of the cacheline provided by the system and deliver to the native agent.</p> <p>Final Cacheline States: Requester: I Snooper: I, SC, SD, UC, UCE, UD, UDP.</p>
Read Clean (ReadClean)	C	<p>Obtain the latest <i>clean</i> coherent copy of a cacheline to be installed into an agent's cache.</p> <p>Note: This operation corresponds to the ACE and CHI ReadClean operations. This operation is also used for an Exclusive Read access. In ACE, signal ARLOCK, and in CHI, signal Excl is additionally asserted to indicate the <i>Exclusiveness</i> of the access.</p> <p>Final Cacheline States: Requester: SC or UC. Snooper: I, SC, SD.</p>

Table 3-1: CCMP Operations

CCMP Operation (Mnemonic)	C or NC ^a	Operation Description
Read Valid / Read Shared (ReadValid / ReadShared)	C	<p>Obtain a copy of a cacheline <i>in any valid state</i>.</p> <p>Note: This operation corresponds to the ACE and CHI ReadShared operations.</p> <p>This operation is also used for an <i>Exclusive Read access</i>. In ACE, signal <i>ARLOCK</i>, and in CHI, signal <i>Excl</i> is additionally asserted to indicate the <i>Exclusiveness</i> of the access.</p> <p>Final Cacheline States: Requester: SC, SD, UC, or UD. Snooper: I, SC, SD.</p>
Read Unique (ReadUnique)	C	<p>Obtain a copy of a cacheline in <i>unique</i> state, requiring all other agents to invalidate all copies of the cacheline.</p> <p>Note: This operation corresponds to the ACE and CHI ReadUnique operation.</p> <p>Final Cacheline States: Requester: UC or UD. Snooper: I.</p>
Read with no Shared-Dirty (ReadNotShDirty)	C	<p>Obtain a copy of a cacheline in <i>non-Shared-Dirty</i> state.</p> <p>Note: This operation corresponds to the ACE and CHI ReadNotSharedDirty operations.</p> <p>Final Cacheline States: Requester: SC, UC or UD. Snooper: I, SC, or SD.</p>
Non-coherent Read (ReadNC)	NC	<p>Obtain an Atomic Block up to a cacheline non-coherently, i.e., without snooping other agents in the system.</p> <p>Access may be to system memory or peripheral storage.</p> <p>Since the data is accessed non-coherently, it is not tracked by the directory.</p> <p>Note: This operation corresponds to the ACE and CHI ReadNoSnoop operation.</p> <p>Final Cacheline States: Requester: I. Snooper: NA.</p>
Clean Unique (CleanUnique)	C	<p>Create a unique copy of a cacheline in the initiating agent when the cacheline is in the invalid (IX) or Shared (SC or SD) state in the agent.</p> <p>If a snooped agent has a dirty copy of a cacheline in its cache, that cacheline is <i>flushed</i> to the next level cache or system memory. All other copies of the cacheline are invalidated.</p> <p>No data is transferred to the initiating agent during this operation. So, the starting IX state results in the UCE state, and SC and SD result in UC and UD respectively.</p> <p>This operation is also used for an <i>Exclusive Store access</i>. In ACE, signal <i>ARLOCK</i>, and in CHI, signal <i>Excl</i> is additionally asserted to indicate the Exclusive access.</p> <p>Final Cacheline States: Requester: UCE, UC or UD. Snooper: I.</p>
Clean Valid (or Clean Shared) (CleanValid or CleanShared)	C or NC	<p>This is a cache maintenance operation (CMO).</p> <p>Ensure that dirty copy of the cacheline in any of the caches in the coherency domain is written back to memory to the point of coherent visibility.</p> <p>If a snooped agent has a dirty copy of a cacheline in its cache, that agent <i>must</i> update the system memory. This also applies to system caches.</p> <p>The agents may retain a clean copy of the cacheline.</p> <p>This is also referred to as a cache <i>Clean</i> operation.</p> <p>Note: In Ncore 3, home location of the cacheline in system memory is updated.</p> <p>Final Cacheline States: Requester: I, SC, UC. Snooper: I, SC, UC.</p>

Table 3-1: CCMP Operations

CCMP Operation (Mnemonic)	C or NC ^a	Operation Description
Clean Shared to Persistent Storage (CleanSharedPersist)	C or NC	<p>This is a cache maintenance operation (CMO). Dirty copy of the cacheline in any of the caches in the coherency domain is written back to memory to the point of coherent visibility. If a snooped agent has a dirty copy of a cacheline in its cache, that agent <i>must</i> update the system memory. This also applies to system caches. The agents may retain a clean copy of the cacheline. Furthermore, the dirty data must be written to the copy of the cacheline in persistent (or non-volatile) storage, if available. This is also a cache Clean operation.</p> <p>Note: In Ncore 3, home location of the cacheline in system memory is updated.</p> <p>Final Cacheline States: Requester: I, SC, UC. Snooper: I, SC, UC.</p>
Clean Invalid (or Flush) (CleanInvalid)	C or NC	<p>This is a cache maintenance operation (CMO). Dirty copy of the cacheline in any of the caches in the coherency domain is written back to memory to the point of coherent visibility. If a snooped agent has a copy of a cacheline in a valid and dirty state, that agent <i>must</i> update system memory. This also applies to <i>system caches</i>. In addition, snooped agents <i>must</i> invalidate all copies of the cacheline. This is also referred to as a cache Flush operation.</p> <p>Note: In Ncore 3, home location of the cacheline in system memory is updated.</p> <p>Final Cacheline States: Requester: I. Snooper: I.</p>
Make Invalid (MakeInvalid)	C or NC	<p>This is a cache maintenance operation (CMO). Invalidate the copy of the cacheline <i>in situ</i> for all agents in the coherency domain, without updating the system memory.</p> <p>Final Cacheline States: Requester: I. Snooper: I.</p>
Make Unique (MakeUnique)	C	<p>Make copy of the cacheline in the requester unique, <i>in place</i>, for a full replacement of its contents. Also, invalidate the copies of the cacheline, <i>in situ</i>, for all other agents in the coherency domain, without updating the system memory. No data is transferred to the requester.</p> <p>Final Cacheline States: Requester: UD. Snooper: I.</p>

Table 3-1: CCMP Operations

CCMP Operation (Mnemonic)	C or NC ^a	Operation Description
Write Unique Partial Cacheline (WriteUniquePtl)	C	<p>Write Atomic Block (AB) amount (up to a cacheline) of new data to the cacheline in system memory when the cacheline is invalid in the requesting agent.</p> <p>The actual new bytes being written can be sparse within the AB and are indicated via Byte Enables accompanying the data.</p> <p>If a snooped agent has the cacheline in partial or full dirty state, the dirty bytes are written to memory before the new data is written to memory.</p> <p>The new data associated with the operation overrides, on a per byte basis, the dirty data supplied by the agent and existing bytes of the cacheline.</p> <p>Also, invalidate the copies of the cacheline, in situ, for all other agents in the coherency domain.</p> <p>Final Cacheline States: Requester: I. Snooper: I.</p>
Write Unique Full Cacheline (WriteUniqueFull)	C	<p>Write a whole cacheline worth of new data to the cacheline in system memory when the cacheline is invalid in the requesting agent.</p> <p>All Byte Enables accompanying the data are set.</p> <p>Also, invalidate the copies of the cacheline, including any dirty copy, in situ, for all other agents in the coherency domain.</p> <p>Final Cacheline States: Requester: I. Snooper: I.</p>
Write Clean Partial Cacheline (WriteCleanPtl)	NC	<p>Write Atomic Block (AB) amount (up to a cacheline) of cacheline data to system memory when the partial cacheline is dirty in the requesting agent.</p> <p>The actual new bytes being written can be sparse within the AB and are indicated via Byte Enables accompanying the data.</p> <p>The requester may retain the cacheline in clean but empty state (UCE) or invalidate it.</p> <p>Note: This operation is not supported in CHI-B (and later versions of CHI protocol). It corresponds to the ACE and CHI-A/E-WriteCleanPtl operation.</p> <p>Final Cacheline States: Requester: I, UCE. Snooper: I.</p>
Write Clean Full Cacheline (WriteCleanFull)	NC	<p>Write a full cacheline worth data to system memory when the cacheline is dirty in the requesting agent.</p> <p>Since a full cache line is being written, all Byte Enables accompanying the data are asserted.</p> <p>The requester may retain the cacheline in clean state (UC).</p> <p>Final Cacheline States: Requester: SC, UC. Snooper: I.</p>
Write Back Partial Cacheline (WriteBackPtl)	NC	<p>Write Atomic Block (AB) amount (up to a cacheline) of cacheline data to system memory when the partial cacheline is dirty in the requesting agent.</p> <p>The actual bytes being written can be sparse within the AB and are indicated via Byte Enables accompanying the data.</p> <p>The requester invalidates its copy of the cacheline.</p> <p>Note: This operation corresponds to the ACE and CHI-E/B WriteBackPtl operation.</p> <p>Final Cacheline States: Requester: I. Snooper: I.</p>

Table 3-1: CCMP Operations

CCMP Operation (Mnemonic)	C or NC^a	Operation Description
Write Back Full Cacheline (WriteBackFull)	NC	<p>Write a full cacheline worth data to system memory when the cacheline is dirty in the requesting agent.</p> <p>Since a full cache line is being written, all Byte Enables accompanying the data are asserted.</p> <p>The requester invalidates its copy of the cacheline.</p> <p>Note: This operation corresponds to the ACE and CHI-E/B WriteBackFull operation.</p> <p>Final Cacheline States: Requester: I. Snooper: I.</p>
Write Evict Full Cacheline (WriteEvict)	NC	<p>Write a full cacheline worth data to system memory when the cacheline is in UC state in the requesting agent.</p> <p>Since a full cache line is being written, all Byte Enables accompanying the data are asserted.</p> <p>The requester invalidates its copy of the cacheline.</p> <p>Note: This operation corresponds to the ACE and CHI-E/B WriteEvictFull operation.</p> <p>Final Cacheline States: Requester: I. Snooper: I.</p>
Write Partial Cacheline non-coherently (WriteNCPtl)	NC	<p>Write Atomic Block (AB) amount (up to a cacheline) of cacheline data non-coherently to system memory or peripheral storage.</p> <p>The actual bytes being written can be sparse within the AB and are indicated via Byte Enables accompanying the data.</p> <p>Note: This operation corresponds to the ACE and CHI-E/B WriteNoSnpPtl operation.</p> <p>Final Cacheline States: Requester: Not applicable. Snooper: Not applicable.</p>
Write Full Cacheline non-coherently (WriteNCFull)	NC	<p>Write a full cacheline worth data non-coherently to system memory or peripheral storage.</p> <p>Since a full cache line is being written, all Byte Enables accompanying the data are asserted.</p> <p>Note: This operation corresponds to the ACE and CHI-E/B WriteNoSnpFull operation.</p> <p>Final Cacheline States: Requester: Not applicable. Snooper: Not applicable.</p>

Table 3-1: CCMP Operations

CCMP Operation (Mnemonic)	C or NC^a	Operation Description
Write Stash Partial Cacheline (WriteStashPtl)	C	<p>Write Atomic Block (AB) amount (up to a cacheline) of new data to the cacheline in system memory when the cacheline is invalid in the requesting agent.</p> <p>The actual new bytes being written can be sparse within the AB and are indicated via Byte Enables accompanying the data.</p> <p>If a snooped agent has the cacheline in partial or full dirty state, the dirty bytes are written to coherent visibility.</p> <p>The new data associated with the operation overrides, on a per byte basis, the dirty data supplied by the agent and existing bytes of the cacheline.</p> <p>Also, invalidate the copies of the cacheline, in situ, for all other agents in the coherency domain.</p> <p>Also, a target agent is identified so that, snoop to target agent contains hint to ask permission to install the cacheline in cache. If accepted, the full cacheline is installed in UD state in the targeted agent instead of writing to system memory. If not accepted, the updated cacheline is written to memory with allocate hint to system cache.</p> <p>If a specific target is not indicated in the operation, the cacheline is written to system memory with allocate hint to system cache.</p> <p>Final Cacheline States: Requester: I. Snooper: I. Target agent: UD if stash is accepted; I otherwise.</p>
Write Stash Full Cacheline (WriteStashFull)	C	<p>Write a whole cacheline worth of new data to the cacheline in system memory when the cacheline is invalid in the requesting agent.</p> <p>All of the Byte Enables accompanying the data are set.</p> <p>Operation invalidates the copies of the cacheline, including any dirty copy, in situ, for all other agents in the coherency domain.</p> <p>Also, a target agent is identified so that, snoop to target agent contains hint to ask permission to install the cacheline in its cache. If accepted, the full cacheline is installed in UD state in the target agent instead of writing to system memory. If not accepted, the updated cacheline is written to memory with allocate hint to system cache.</p> <p>If a specific target is not indicated in the operation, the cacheline is written to system memory with allocate hint to system cache.</p> <p>Final Cacheline States: Requester: I. Snooper: I. Target agent: UD if stash is accepted; I otherwise.</p>
Load external cache with cacheline in Shared state (LoadCchShared)	C	<p>Snoop a target agent to ask permission to install the cacheline in its cache. If accepted, the full cacheline is installed in a shared or unique state in the target agent. If not accepted, the cacheline may be read from memory and allocated into the system cache.</p> <p>If a specific target is not indicated in the operation, the cacheline may be read from the memory and allocated into the system cache.</p> <p>Final Cacheline States: Requester: I. Snooper: I, SC, or SD. Target agent: SC, UC or UD if stash is accepted; I otherwise.</p>

Table 3-1: CCMP Operations

CCMP Operation (Mnemonic)	C or NC^a	Operation Description
Load external cache with cacheline in Unique state (LoadCchUnique)	C	<p>Snoop a target agent to ask permission to install the cacheline in its cache. If accepted, the full cacheline is installed in a shared or unique state in the target agent. If not accepted, the cacheline may be read from memory and allocated into the system cache. If a specific target is not indicated in the operation, the cacheline may be read from the memory and allocated into the system cache.</p> <p>Final Cacheline States: Requester: I. Snooper: I. Note: Target agent: UC or UD if stash is accepted; I otherwise.</p>
Read with No Intention to Cache with Clean and Invalidate (ReadNITCCleanInvalid)	C	<p>Obtain the latest snapshot of the cacheline. The snapshot is not installed in the CCMP coherency domain. Thus, the snapshot itself is no longer kept coherent after the operation.</p> <p>After the snapshot is taken, dirty copy of the cacheline in any of the caches in the coherency domain is written back to memory to the point of coherent visibility. In addition, snooped agents <i>must</i> invalidate all copies of the cacheline.</p> <p>Final Cacheline States: Requester: I. Snooper: I Note: This transaction corresponds to the CHI-B ReadOnce-CleanInvalid transaction.</p>
Read with No Intention to Cache with Make Invalidate (ReadNITCMakelnInvalid)	C	<p>Obtain the latest snapshot of the cacheline. The snapshot is not installed in the CCMP coherency domain. Thus, the snapshot itself is no longer kept coherent after the operation.</p> <p>After the snapshot is taken, all copies of the cacheline are invalidated <i>in situ</i>, without updating system memory.</p> <p>Final Cacheline States: Requester: I. Snooper: I Note: This transaction corresponds to the CHI-B ReadOnce-MakeInvalid transaction.</p>
Read Atomic (Read Atomic)	C or NC	<p>Perform an atomic update to an aligned block of data of size 1, 2, 4, or 8 bytes and return the original value of the block.</p> <p>Perform an atomic update (Read-Modify-Write) of the block specified in the operation with the operand data accompanying the operation. For efficiency, the update is performed close to the home location of the data.</p> <p>The original value of the aligned block is returned for the Read. The data returned is not cached by the requester.</p> <p>For a coherent Read Atomic operation, coherency phase is performed first in which, the addressed location is cleaned to system memory first and the cached copies invalidated, before conducting the Atomic Read operation.</p> <p>Note: CCMP supports Read Atomic only for addresses located in system memory.</p> <p>Final Cacheline States: Requester: I. Snooper: I</p>

Table 3-1: CCMP Operations

CCMP Operation (Mnemonic)	C or NC ^a	Operation Description
Write Atomic (Write Atomic)	C or NC	<p>Perform an atomic update to an aligned block of data of size 1, 2, 4, or 8 bytes.</p> <p>Perform an atomic update (Read-Modify-Write) of the block specified in the operation with the operand data accompanying the operation. For efficiency, the update is performed close to the home location of the data.</p> <p>No data is returned for the Write Atomic.</p> <p>For a coherent Write Atomic operation, coherency phase is performed first in which, the addressed location is cleaned to system memory first and the cached copies invalidated, before conducting the Write Atomic operation.</p> <p>Note: CCMP supports Write Atomic only for addresses located in system memory.</p> <p>Final Cacheline States: Requester: I. Snooper: I</p>
Compare and Swap Atomic	C or NC	<p>The operation specifies two data values: an operand for the compare and the second for the swap.</p> <p>Perform a compare of an aligned block of data of size 1, 2, 4, or 8 bytes in memory with the operand data of the same size specified by the operation. If a match occurs, then the second operand is swapped with the second operand.</p> <p>For efficiency, the compare and swap atomic is performed close to the home location of the data.</p> <p>The original value of the aligned block is returned for the operation. The data returned is not cached by the requester.</p> <p>For a coherent Compare and Swap Atomic operation, coherency phase is performed first in which, the addressed location is cleaned to system memory first and the cached copies invalidated, before conducting the Compare and Swap Atomic operation.</p> <p>Note: CCMP supports Compare and Swap atomic only for addresses located in system memory.</p> <p>Final Cacheline States: Requester: I. Snooper: I</p>
Swap Atomic	C or NC	<p>Perform a swap atomic of an aligned block of data of size 1, 2, 4, or 8 bytes in memory with the operand data of the same size specified by the operation.</p> <p>For efficiency, the swap atomic is performed close to the home location of the data.</p> <p>The original value of the aligned block is returned for the operation. The data returned is not cached by the requester.</p> <p>For a coherent Swap Atomic operation, coherency phase is performed first in which, the addressed location is cleaned to system memory first and the cached copies invalidated, before conducting the Swap Atomic operation.</p> <p>Note: CCMP supports Swap Atomic only for addresses located in system memory.</p> <p>Final Cacheline States: Requester: I. Snooper: I</p>
Prefetch	C / NC	<p>Prefetch a cacheline from system memory device to system cache.</p> <p>Final Cacheline States: Requester: NA. Snooper: NA.</p>
Distributed Virtual Memory Management operation (DVMOp)	NC	Perform a Distributed Virtual Memory Management operation

Table 3-1: CCMP Operations

CCMP Operation (Mnemonic)	C or NC ^a	Operation Description
Update Directory Invalid (UpdInv)	NC	<p>Update the agent state in the directory to invalid for the cacheline indicated. If no entry exists in the directory for the cacheline, the operation is treated as don't care. If the entry does exist and it does track the state of the cacheline within the agent, the entry is updated to indicate that the agent is in invalid state for the cacheline.</p> <p>The system <i>must guarantee</i> that an Update Directory Invalid operation, including all its associated protocol messages, can make forward progress under all circumstances.</p> <p>Note: This operation corresponds to the ACE Evict operation.</p> <p>Final Cacheline States: Requester: I. Snooper: NA.</p>
Update Directory Shared Clean (UpdSCIn)	NC	<p>Update the agent state in the directory to Shared Clean for the cacheline indicated. The directory entry is updated to indicate that the agent is in shared clean state for the cacheline.</p> <p>The system <i>must guarantee</i> that an Update Directory Shared Clean operation, including all its associated protocol messages, can make forward progress under all circumstances.</p> <p>Final Cacheline States: Requester: SC. Snooper: NA.</p>
Notes:		<ul style="list-style-type: none"> a. Coherent or Non-coherent operation. An operation is labeled in CCMP as non-coherent if, by definition, it is not required to be snooped to other agents b. The term <i>snapshot</i> distinguishes from the term <i>copy</i> of a cacheline. The latter is retained by a cache that participates in the CCMP coherency protocol to keep the copy coherent. The snapshot, on the other hand, is the value of the cacheline at a point of time. The receiver of the snapshot does not participate in the CCMP protocol to keep the value coherent with respect to the rest of the system

The processing of an operation in CCMP commences with the issuance by the AIU of a **Command Request** message, shortened as a **CMDreq**. There is a one-to-one correspondence between CCMP operations and CMDreqs. The CMDreq message carries all the information to fully specify the type and semantics of the operation.

Table 4-2 shows the correspondence from CCMP operations to mnemonics of CMDreqs by which they are represented. The table also shows the specific encoding for each CMDreq.

3.2 Mapping of Native Operations to CCMP

Table 3-2 below shows the mapping of request transactions from various native interfaces and protocols to CCMP operations.

The ACE, ACE-Lite, ACE5-Lite column shows the channel — Read (R) or Write (W) — on which the request transaction is expected to be issued. It also shows the legal values of AxBAR, AxDOMAIN, and AxSNOOP fields that may be used in the request transaction. Thus, for example {01/10} for ReadOnce indicates that ARDOMAIN field may be legally set to either 0b01 or 0b10 for the ReadOnce transaction. “:” is used to separate the fields. “|” is used to indicate concatenation of bits, which is necessitated by the increase in the size of the field in ACE5-Lite.

Table 3-2: Mapping of CHI-E/B, ACE, ACE-Lite, and ACE-Lite operations to CCMP operations

Native Operation / Request Transaction Mappings		CCMP Operation	
CHI-E / CHI-B	ACE, ACE-Lite, ACE5-Lite	BAR : DOMAIN : SNOOP	Mnemonic
Mnemonic	Code [5:0]	Mnemonic	
ReadOnce	0x03	ReadOnce	R: 0 : {01/10} : 0000
ReadClean	0x02	ReadClean	R: 0 : {01/10} : 0010
ReadShared	0x01	ReadShared	R: 0 : {01/10} : 0001
ReadUnique	0x07	ReadUnique	R: 0 : {01/10} : 0111
ReadNotSharedDirty	0x26	ReadNotSharedDirty	R: 0 : {01/10} : 0011
ReadNoSnp	0x04	ReadNoSnoop	R: 0 : {00/11} : 0000
CleanUnique	0x0B	CleanUnique	R: 0 : {01/10} : 1011
CleanShared	0x08	CleanShared	R: 0 : {00/01/10} : 1000
CleanSharedPersist	0x27	ALE: CleanSharedPersist	R: 0 : {00/01/10} : 1010
CleanInvalid	0x09	CleanInvalid	R: 0 : {00/01/10} : 1001
MakeInvalid	0x0A	MakeInvalid	R: 0 : {00/01/10} : 1101
MakeUnique	0x0C	MakeUnique	R: 0 : {01/10} : 1100
Evict	0x0D	Evict	W: 0 : {01/10} : 100
WriteUniquePtl	0x18	WriteUnique	W: 0 : {01/10} : 000
WriteUniqueFull	0x19	WriteLineUnique	W: 0 : {01/10} : 001
(WriteCleanPtl)	(0x16)	Supported with discretion ⁹	NA
WriteCleanFull	0x17	WriteClean	W: 0 : {00/01/10} : 010
WriteBackPtl	0x1A	Supported with discretion ¹⁰	W: 0 : {00/01/10} : 011
WriteBackFull	0x1B	WriteBack	W: 0 : {00/01/10} : 011
WriteEvictFull	0x15	WriteEvict	W: 0 : {00/01/10} : 101
WriteNoSnpPtl	0x1C	WriteNoSnp	W: 0 : {00/11} : 000
WriteNoSnpFull	0x1D	WriteNoSnp	W: 0 : {00/11} : 000
WriteUniquePtlStash	0x21	ALE: WriteUniquePtlStash	W: 0 : {01/10} : {1 000}
WriteUniqueFullStash	0x20	ALE: WriteUniqueFullStash	W: 0 : {01/10} : {1 001}
StashOnceShared	0x22	ALE: StashOnceShared	W: 0 : {01/10} : {1 100}
			LoadCchShared

⁹ For ACE, WriteClean is allowed to do the partial update with sparse enables, CCMP supports it with different scenarios as described in section 5.1.6.1

¹⁰ For Ace, WriteBack is allowed to do the partial write with sparse enables, CCMP supports it with different scenarios as described in section 5.1.6.1

Table 3-2: Mapping of CHI-E/B, ACE, ACE-Lite, and ACE-Lite operations to CCMP operations

Native Operation / Request Transaction Mappings				
CHI-E / CHI-B		ACE, ACE-Lite, ACE5-Lite		CCMP Operation
Mnemonic	Code [5:0]	Mnemonic	BAR : DOMAIN : SNOOP	Mnemonic
StashOnceUnique	0x23	ALE: StashOnceUnique	W: 0 : {01/10} : {1 101}	LoadCchUnique
ReadOnceCleanInvalid	0x24	ReadOnceCleanInvalid	R: 0 : {00/01/10} : {0 100}	ReadNITCCleanInvalid
ReadOnceMakeInvalid	0x25	ReadOnceCleanInvalid	R: 0 : {00/01/10} : {0 101}	ReadNITCMakeInvalid
AtomicLoad	0x28-0x2F	ALE: AtomicLoad	W: 0 : {01/10} : {1 001} ^{aa}	Read Atomic
AtomicStore	0x30-0x37	ALE: AtomicStore	W: 0 : {01/10} : {1 001} ^{aa}	Write Atomic
AtomicCompare	0x39	ALE: AtomicCompare	W: 0 : {01/10} : {1 001} ^{aa}	Compare And Swap Atomic
AtomicSwap	0x38	ALE: AtomicSwap	W: 0 : {01/10} : {1 001} ^{aa}	Swap Atomic
PrefetchTgt	0x3A	NA	NA	Prefetch
DvmOp	0x14	DVM Message	R: 0 : {01/10} : 1111	DVMOp

Note:

- a. AtomicLoad: AWATOP[5:0] = 0b10exxx; AtomicStore: AWATOP[5:0] = 0b01exxx; AtomicCompare: AWATOP[5:0] = 0b11001; AtomicSwap: AWATOP[5:0] = 0b11000. “e” indicates Endianess of data. In AXI / ACE- Lite-E, AWATOP[5:0] = 0b000000 for all non-atomics.

3.2.1 Special handling of WriteBacks in ACE

See Section 6.3.4.3.1.

3.3 Mapping of CCMP operations to Native Interfaces

To be added

Chapter 4: Concerto-C Messages

This chapter specifies the format of CCMP messages and describes all the field used within them.

4.1 Conventions and Notations

The Non-data portion of a message is organized as a bit-string in the ***Big-Endian order***, with bit 0 being at the left extremity of the message. The first field of a message starts at bit 0 through bit number equal to the width of the field, minus 1, with the bits of the field laid out in their natural order from left to right. The next field starts at bit position after that, and so on.

The fields in the message are thus expected to be ordered in the order they appear in this text.

The Data portion of a message, if present, on the other hand is organized as a byte-string in the ***Little-Endian order***, such that least significant byte appears at the right extremity of the data portion.

Some fields are formed by concatenating subfields. The “|” symbol is used to indicate bit-wise concatenation of values on either side of this symbol.

4.2 Concerto Message Types

The following table shows different classes of messages and their mnemonics and classifies them as request or response messages. Request messages are paired with their corresponding response messages in the same row.

These message classes and individual message type within them are dealt with in detail in subsequent chapters.

Table 4-1: Concerto-C Message Classes

Message Class	Function	Request Mnemonic	Response Mnemonic
Non-data Messages:			
Coherent Commands (CMD)	Represents CCMP operation, and in turn native operation request C-AIU, NC-AIU → DCE	CMDreq	C-CMDrsp
Non-coherent Commands (CMD)	Represents CCMP operation, and in turn native operation request C-AIU, NC-AIU → DCE, DMI, or DII (see Section 4.5.3.3.1.2 for details)		C-CMDrsp, NC-CMDrsp (see Section 4.5.5)
Snoop (SNP)	Req Message: Snoop commands are issued to enforce coherency among agent caches DCE → C-AIU or to NC-AIU with proxy caches DVE → C-AIU and NC-AIU with SMMU or with agents that have MMUs kept coherent with processor MMUs Rsp Message: Communicates completion of coherency activity at the snooper and its agents. Also transmits coherency responses from the C-AIU on behalf of its agents.	SNPreq	SNPrsp
Memory Read Directive (MRD)	Message class to ask that data to be sent in response to a Read CMD. DCE → DMI	MRDreq	MRD rsp
Prefetch Hint	Asks data to be pre-fetched in response to a Prefetch CMD. DCE → DMI	HNTreq	HNTrsp
State Reply	Req Message: Reports summary cache state DCE → C-AIU or NC-AIU Provides buffer reference for transferring data to initiating AIU and to signal that data transfer may now be initiated. Rsp Message: Signals a completion of a CMD semantics at the initiating AIU and its agents.	STRreq	STRrsp
Directory Update	Reports deletion of a cacheline copy from native agent. C-AIU or NC-AIU with proxy cache --> DCE	UPDreq	UPDrsp
Request Buffer Request	Requests allocation/release of a Request Buffer in a DMI for use by Write Data Transfers. DCE → DMI.	RBRreq	RBRrsp
Completion	(Unsolicited) Indication of completion of a non-buffered Write CMD. DMI or DII to a C-AIU or NC-AIU..	None	CMPrsp
Concerto Message Error	Message class used to report that an error was detected in a CCMP message by the Target of that message. DMI or DII to an AIU.	None	CMErsp

Table 4-1: Concerto-C Message Classes

Message Class	Function	Request Mnemonic	Response Mnemonic
Transport Error	Message class used to report an error that was detected by the transport service component (Legato fabric component) during transit from an Ncore 3 unit to another. The message indicates the original CCMP message that encountered the error. Transport component → An Ncore 3 unit.	-	TRErsp
Data Messages:			
Data Reply	Message class used to transmit data response to an initiator AIU. DMI, DII, or snooper AIU → AIU	DTRreq	DTRrsp
Data Write	Message class used to transmit data to DMI or DII. AIU → DMI or DII.	DTWreq	DTWrsp

4.3 Message Formats

CCMP messages are a string of bits, formed out concatenation of a well-defined sequence of multiple one or multi-bit fields, that flows from an initiator Ncore 3 unit to a target Ncore 3 unit.

Two basic formats are used for CCMP messages:

A Data Message (DM)

In addition to command and/or control information, a data message carries data that may be read from or written into storage locations.¹¹

A Non-Data Message (NDM)

A non-data message carries only command and/or control information.

Below are shown formats of NDM and DM:

```
NDM    =      <Concerto-C Message Header> + <Non-Data Payload>
DM     =      <Concerto-C Message Header> + <Non-Data Payload> +
              <Data Payload> + <Auxiliary Payload>
```

¹¹ For the sake of precision, data, here specifically refers to the information in storage that is accessed via Read or Write operation

4.4 Concerto-C Message Headers

All Concerto-C messages have a common defined sequence of fields at their start. Although the width of the fields can change from one Ncore instance to the next, the order of fields is fixed. This sequence forms the **Common CCMP Message Header** for all Concerto-C messages. The common header is carried intact end-to-end from the initiator to the target of the message.

The **Initiator Unit** may transmit an additional sequence of fields along with a message. These are placed following the common header. These field together are called the **Common CCMP Message Header Extension**. The following paragraphs define the fields in *common message header* and the *common header extension*.

4.4.1 Common Message Header fields

Figure 4-1 below shows the TX and RX headers for a CCMP message. Target Id, Initiator Id, CMType, Message Id, and HProtection are fields that are part of the *Common CCMP Message Header*.

TX Header	RX Header
Target ID	Target ID ← Network Identifier of the intended receiver
Initiator ID	Initiator ID ← Network Identifier of the sender
CMType	CMType ← Type of Concerto-C Message
Message ID	Message ID ← Unique Message Identifier at the sender
HProtection	HProtection ← Protection for the above 4 fields
TTier	← Traffic Tier
Steering	← Steering attributes for the message
Priority	← Priority Level for the message
QL	← QoS Label

Figure 4-1: Concerto-C Message Headers

There are other fields in the TX header that are not intended to be carried end-to-end, but are included in the header. It is via these fields that the Ncore unit at the interface makes specific service requests of the transport service. The purpose of these fields is specified below.

4.4.1.1 Target ID

Target Id is the Fabric Unit ID¹² (FUnitID) of the target Ncore unit of the message.

4.4.1.2 Initiator ID

Initiator Id is the FUnitID of the Ncore unit that is initiating this message.

4.4.1.3 CMType

CMType indicates the type of this present message.

¹² Refer to the Ncore 3 System Architecture Specification, Programming Interface chapter

4.4.1.4 Message ID

Message Id refers to a unique identifier the sender of this message assigns to it. It forms the Reference Message Id in the response to this current message, if any.

4.4.1.5 HProtection

This is the header protection field that covers the other fields as well as itself for error detection. This helps in identifying errors that may occur during the transportation of the message from the initiator to the target. The target of the message checks for transport errors using this field before *consuming* the message.

4.4.2 Transmit Header Extension

The following section describes the fields in the transmit header extension and their use.

These fields are used as a mechanism to indicate the type of **message transport service** that is being requested from the *transport interconnection facility* of the coherency system. The extended header fields are extracted by the *terminal block* which is transport fabric's interface at the initiator of the message.

The Header Extension fields do not get delivered to the *Target unit* and are not carried end-to-end. from Initiator Unit to Target Unit.

Fields TTier, Steering, Priority, and QL form the *transmit header extension*.

4.4.2.1 TTier

This field stands for “Traffic Tier”. This field is used for deadlock avoidance purpose. Each type of Concerto-C message is associated with an architected TTier value. Multiple types of messages can have the same TTier value.

The transport network maps a TTier value to a virtual channel for each of the links over which the message is carried through the network. For a given message type, the virtual channel chosen can be different at different links. However, the following rule must be satisfied to ensure freedom for deadlocks.

Deadlock Prevention Rule: A message carrying a value V for TTier must not be permanently blocked by a message carrying a value greater than V. Satisfaction of this rule in the interconnect is mandatory to prevent deadlocks. Minimum value for this field is 0. Thus, messages with TTier = 0 must not be blocked by any message with TTier > 0.

This field is not intended to be treated as a Quality-of-Service attribute.

4.4.2.2 Steering

This field influences the determination of the route taken by the message to reach its target.

4.4.2.3 Priority

When two messages are competing to pass through a common path element, the higher priority message is given preference over a lower priority message in the instantaneous choice made when deciding which next message to forward. There can be other considerations that may momentarily override this choice.

4.4.2.4 QL

QoS Label. This field is used to enable bandwidth provisioning and requirements.

The Concerto-C protocol supports several operation types that can be categorized as coherent or non-coherent. The former cause changes in the state of the caches in the initiating agent and any snooped agents. These state changes also cause corresponding state changes in the system directory. The non-coherent operations do not involve system directory and snooping.

Operation types provided by CCMP are described in [Table 4-2](#)[Table 4-2](#) below.

Note that for coherent operations, the actions taken are with respect to all the agents in the coherency domain and the state of the cacheline refers to either at the requester agent or all agents across the coherency domain.

Each CCMP operation effected in CCMP via a well-defined sequence of messages. The first of these messages is the Command request message (CMDreq) that carries all the relevant and necessary semantic content of the operation.

The [Table 4-2](#)[Table 4-2](#) shows the CMDreq that is issued for each of the operation. CMDreq message contents are discussed in detail in section 4.5.

Appendix 1 shows mappings of transactions from various native interfaces and protocols to CCMP operations and CMDreqs.

4.5 CMDreq and CMDrsp Messages

A CMDreq is the first message that gets generated to service an operation received from a native agent via a native request transaction. A CMDreq causes other necessary messages to be generated in the Ncore 3 system to effectuate the semantics of the operation.

4.5.1 CMDreq Messages

Each unique type of native transaction invokes a unique CCMP operation, which in turn causes a unique type of CMDreq to be generated. [Table 4-2](#) shows the CMDreq message generated for each CCMP operation. The table also shows the specific encoding for each CMDreq.

Table 4-2: CCMP CMDreqs generated to start the CCMP Operations

Operation Mnemonic	CMDReq Mnemonic	MsgType[7:0]
ReadNITC	RdNITC	0x07
ReadClean	RdCln	0x01
ReadValid	RdVld	0x03
ReadUnique	RdUnq	0x04
ReadNotShDirty	RdNShDty	0x02
ReadNC	RdNC	0x0B
CleanUnique	ClnUnq	0x05
CleanValid or CleanShared	ClnVld	0x08
CleanSharedPersist	ClnShPsist	0x28
CleanInvalid	ClnInv	0x09
MakeInvalid	MkInv	0x0A
MakeUnique	MkUnq	0x06
Evict	Evict	0x17
WriteUniquePtl	WrUnqPtl	0x10
WriteUniqueFull	WrUnqFull	0x11
WriteCleanPtl	WrClnPtl	0x19
WriteCleanFull	WrClnFull	0x15
WriteBackPtl	WrBkPtl	0x18
WriteBackFull	WrBkFull	0x14
WriteEvict	WrEvict	0x16
WriteNCPtl	WrNCPtl	0x20
WriteNCFull	WrNCFull	0x21
WriteStashPtl	WrStshPtl	0x23
WriteStashFull	WrStshFull	0x22
LoadCchShared	LdCchShd	0x24
LoadCchUnique	LdCchUnq	0x25
ReadNITCCleanInvalid	RdNITCCI	0x26
ReadNITCMakeInvalid	RdNITCMI	0x27
Read Atomic	RdAtm	0x13
Write Atomic	WrAtm	0x12
Compare And Swap Atomic	CompAtm	0x2A
Swap Atomic	SwapAtm	0x29

Table 4-2: CCMP CMDreqs generated to start the CCMP Operations

Operation Mnemonic	CMDReq Mnemonic	MsgType[7:0]
Prefetch	Pref	0x2B
Update Directory Invalid ^a	UpdInv	0x70
Update Directory Shared Clean ^a	UpdSCln	0x71
DVMOp	DVMMsg	0x0F
Note:	a. For UpdInv and UpdSCln CMDreqs see Section 4.14.	

4.5.2 CMDreq message fields

[Table 4-3](#)[Table 4-3](#) shows the fields in a CMDreq message.

Table 4-3: CMDreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:				CCMP Message Header	
TargetId	wTargetId	6-12	O/I	Target Identifier	Local: Address decode / Transaction Type
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier	Local: FUnitId
CMTYPE	wCMTYPE	8	O/I	Type of Message	Local. See Table 4-2 Table 4-2 .
Messageld	wMessageld	6-12	O/I	Messageld	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTYPE, and Messageld.	Local
CMHE:				CCMP Message Header Extension	
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:				CCMP Message Body	
CMStatus	wCmStatus	8	I	Input-only for CMDreq. Used to deliver Transport encountered errors to the Target unit.	Local: RX Demux
Addr	wAddr	32-52	O/I	Address	NI ^a
VZ	wVZ	1	O/I	Visibility	NI
CA	wCA	1	O/I	Cacheable	NI
AC	wAC	1	O/I	Allocate Hint	NI
CH	wCH	1	O/I	Coherent Access	NI
ST	wST	1	O/I	Storage Type	NI
EN	wEN	1	O/I	Endianness	NI
ES	wES	1	O/I	Exclusive / Self-snoop	NI
NS	wNS	1	O/I	Non-secure Access	NI
PR	wPR	1	O/I	Privileged Access	NI
OR	wOR	2	O/I	Order[1:0]	NI / Local: AIU Imposed
LK	wLK	2	O/I	Lock[1:0]	NI

Table 4-3: CMDreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
RL	wRL	2	O/I	Response Level[1:0]	Local
TM	wTM	1	O/I	Trace Me	NI / Local
MPF1	wMpf1	6-12	O/I	Multi-function field #1	NI
MPF2	wMpf2	9-22	O/I	Multi-function field #2	NI
Size	wSize	3	O/I	Access size.	NI
IntfSize	wIntfSize	3	O/I	Interface Size	Local
DId	wDId	2-6	O/I	Destination Id	Local: Address Decode
TOF	wTof	3	O/I	Transaction Ordering Framework	Local
QoS	wQoS	4	O/I	Quality of Service Label	NI
AUX	wReqAux	0-8	O/I	Auxiliary field, propagate user bits information from CHI, AXI and ACE	NI
MPROT	wReqProt	0-10	O/I	Protection bits for Request Message	NI / Local
Note:					
a. Native Interface					

4.5.3 CMDreq field descriptions

4.5.3.1 CMStatus

The CMStatus[7:0] field is shown only as an I in the direction column in Table 4-3, implying that the field is not transmitted along with the rest of the CMDreq message at the initiator, but is an input-only field at the target of the message.

The CMStatus[] field is used to report the result of a processing step of the operation, including detection of errors, to an Ncore unit. However, CMDreq being the start of the operation, there are no results to report.

If any type of error is detected in the native request transaction, the initiator AIU must not issue a corresponding CMDreq message into the Ncore system to commence the processing of the native operation. The operation must be locally terminated, and error reported back to the native agent.

A CMDreq message may encounter a transport error along its path to the target. This error must be reported to the target via the CMStatus[] field constructed at the transport interface of the target Ncore unit.

[Table 4-4](#) specifies the contents of CMStatus field as a function of the message type that is carrying it.

Table 4-4: CMStatus[7:0] definition

Status Type	CMStatus[7:6]	CMStatus[5:0]
Success (No error)	0b00	<p>SNPrsp: For non-DVM snoops: {RV RS DC DT[1:0] Snarf} (See Section 4.6.7.2.)</p> <p>For DVM snoops: {0 0 0 0 0 0}</p> <p>STRreq: For non-Exclusive Read operations: {Reserved}</p> <p>For Exclusive Read operations: {6:1 Reserved Ex-Okay}</p> <p>For Stashing operations: {6:1 Reserved Snarf}</p> <p>For Exclusive CleanUnique {6:1 Reserved Ex-Okay}</p> <p>For CMO operations: {6:1 Reserved Completion}</p> <p>Other: {0 0 0 0 0 0}</p> <p>STRrsp: {0 0 0 0 0 0} TR[1:0]: 00: Invalid; 01: Shared; 10: Owned (SD, UC, or UD)</p>
<u>Snoop Data Error</u> <u>Reserved</u>	0b01	<u>SNPrsp with Intervention Data Error: For non-DVM snoops:</u> <u>{RV RS 0 DT[1] 0 Snarf, Reserved}</u>

Table 4-4: CMStatus[7:0] definition

Status Type	CMStatus[7:6]	CMStatus[5:0]				
Error	0b10	<p>CCMP-reported errors These codes may be supplied within a <i>response</i> message by one Ncore 3 unit to another Ncore 3 unit to report a transport-reported error it encountered within, or received with, an earlier <i>request</i> message.</p> <table border="1"> <tr> <td>[5] = 0</td><td> <p>CCMP Protocol Errors:</p> <ul style="list-style-type: none"> [4:3]: Reserved [2:0]: 000 - 001: Reserved. 010: Non-data, non-address error. 011: Data error. 100: Address error. 101: Target Signaled Error. 110: Time-out Error. 111: Coherency Error. <p>Note: DVM snoop response error from an agent is signaled as [2:0] = 3'b101 (Target signaled error.)</p> </td></tr> <tr> <td>[5] = 1</td><td> <p>Transport Errors:</p> <ul style="list-style-type: none"> [4:3]: Reserved [2:0]: 000: Transport delivery error. (E.g.: Wrong target reached.) 001 - 100: Reserved. 101: Transport non-data error. 110: Transport data error. 111: Reserved. </td></tr> </table>	[5] = 0	<p>CCMP Protocol Errors:</p> <ul style="list-style-type: none"> [4:3]: Reserved [2:0]: 000 - 001: Reserved. 010: Non-data, non-address error. 011: Data error. 100: Address error. 101: Target Signaled Error. 110: Time-out Error. 111: Coherency Error. <p>Note: DVM snoop response error from an agent is signaled as [2:0] = 3'b101 (Target signaled error.)</p>	[5] = 1	<p>Transport Errors:</p> <ul style="list-style-type: none"> [4:3]: Reserved [2:0]: 000: Transport delivery error. (E.g.: Wrong target reached.) 001 - 100: Reserved. 101: Transport non-data error. 110: Transport data error. 111: Reserved.
[5] = 0	<p>CCMP Protocol Errors:</p> <ul style="list-style-type: none"> [4:3]: Reserved [2:0]: 000 - 001: Reserved. 010: Non-data, non-address error. 011: Data error. 100: Address error. 101: Target Signaled Error. 110: Time-out Error. 111: Coherency Error. <p>Note: DVM snoop response error from an agent is signaled as [2:0] = 3'b101 (Target signaled error.)</p>					
[5] = 1	<p>Transport Errors:</p> <ul style="list-style-type: none"> [4:3]: Reserved [2:0]: 000: Transport delivery error. (E.g.: Wrong target reached.) 001 - 100: Reserved. 101: Transport non-data error. 110: Transport data error. 111: Reserved. 					
Error	0b11	<p>Transport-reported errors^a This type of error may be <i>injected</i> into a CCMP message being delivered to an Ncore unit. In this context, the error is detected by a transport fabric element. Such an injection into a CCMP message occurs at an RX Terminal Block of the transport interconnection with which the recipient Ncore is connected.</p> <table border="1"> <tr> <td>[5:3] = 000</td><td> <p>[2:0]:</p> <ul style="list-style-type: none"> 000: Reserved. 001: Routing error. 010: Access violation 011: Path disconnection. 100: Privilege violation. 101: RX Terminal Block injected error 110 - 111: Reserved. </td></tr> </table>	[5:3] = 000	<p>[2:0]:</p> <ul style="list-style-type: none"> 000: Reserved. 001: Routing error. 010: Access violation 011: Path disconnection. 100: Privilege violation. 101: RX Terminal Block injected error 110 - 111: Reserved. 		
[5:3] = 000	<p>[2:0]:</p> <ul style="list-style-type: none"> 000: Reserved. 001: Routing error. 010: Access violation 011: Path disconnection. 100: Privilege violation. 101: RX Terminal Block injected error 110 - 111: Reserved. 					
Note:						
a. Refer to Ncore 3 System Architecture Specification Chapter on Error Management.						

4.5.3.2 Addr

This field carries in full and unmodified the address field of the native operation.

This field is propagated unmodified and in full in all subsequent CCMP messages caused by this CMDreq, except in the case of DVM operations.

This field is also propagated to all the native transactions resulting from this CMDreq. In these cases, the only modifications permitted is truncation or setting to zero of some of the least significant bits of the field, as may be necessary to comply with the architectural requirements of the respective native interfaces.

For DVM operations, the contents of the DVM snoop operations are dictated by the architectures of the native interfaces over which these snoops are issued and thus may require customized constitution of the respective address fields.

4.5.3.3 Operation attributes

The type of an operation describes the general function that is to be carried out. The exact semantics of an operation are refined via additional information that accompanies it. This additional information is divided into fields which are referred to as **attributes** of the operation. This section specifies the operation attributes supported in CCMP. Not coincidentally, these also correspond to fields carried by the CMDreq messages.

Operation attributes are carried by the CMDreq message in the following fields: VZ, AC, CA, CH, ST, EN, ES, NS, PR, OR, TK, RL, and TM. These are described in this section.

The following set of tables shows the storage attributes carried by Concerto-C messages. They specify the legal values of these attributes and describe the semantics associated with each value. Mnemonics of these fields in CCMP messages are also shown.

Mappings of fields in the various native protocols to CCMP attributes are specified in Appendix 1.

4.5.3.3.1 Basic storage characteristics

Table 4-5 below shows attributes used to describe basic attributes of the storage space referred to by the address.

Table 4-5: Basic Storage Access Attributes

Mnemonic	Name	Specification and Notes
ST	Storage Type	0: System or Peripheral memory. Well-behaved storage. Also called <i>normal</i> memory. Includes DDR-based System Memory and Peripheral memory. Used with storage type is “Device=0” in CHI, and when AxCACHE[1] = 1 for AXI4/ACE. 1: Device-based, non-well-behaved storage. Used when storage type is “Device=1” in CHI, and when AxCACHE[1] = 0 ^a in AXI4/ACE.
CA	Cacheability	0: The address shall not be cached. System caches are not required to inspect their contents to check if there is an address match. 1: The address is cacheable. System caches shall inspect their contents to check if there is an address match.
CH	Coherency requirement	0: non-coherent access. Coherency is not enforced for these CMDreqs. 1: coherent access. Coherency must be enforced for these CMDreqs. Also see Section 4.5.3.3.1.2 below.

Note:

- a. For AXI4 architecture, AxCACHE[3:2] always equals 0b00 when AxCACHE[1] = 0b0, AxCACHE[3:1] == 0b000 is an equivalent check

4.5.3.3.1.1 ST bit usage

As described above, ST = 0 represents access to well-behaved system storage. This primarily includes system memory, which can be maintained coherent via CCMP. However, it may also include peripheral memory such as device-internal well-behaved memory buffers, which is not maintained coherent via CCMP. Additionally, it may also include on-chip SRAM or even off-chip DDR memory which is not maintain coherently in the system.

ST = 1 represents an access to non-well-behaved storage (where access has side-effects) in the system, examples being configuration and status registers in system devices. ST = 1 is also used to represent **non-modifiable** accesses made on an AXI, ACE, ACE-Lite, or ACE5-Lite native interface.

4.5.3.3.1.2 CH and destination-based handling of CMDreqs

Setting of the bit is guided by the type of operation being carried out and whether the causative operation is coherent or non-coherent (see Table 3-1, Column C or NC).

Coherent accesses may cause snoops to be generated depending on the state of the cacheline. Non-coherent accesses do not result in snoops.

- *CMDreqs* with CH = 1 always refer to accesses to coherent space. These are sent to DCE for execution of coherency semantics.
- *CMDreqs* with CH = 0 may refer to accesses to coherent or non-coherent space and may therefore be sent to DCE, DMI, or DII.
- For CHI agents, *CMDreqs WrBkPtl, WrBkFull, WrClnPtl, WrClnFull, WrEvict, and Evict*,¹³ are issued with CH = 0, although they refer to coherently accessed storage space. These are sent to DCE.
- For ACE agents and Proxy caches, operations *WriteBackPtl, WriteBackFull, WriteCleanPtl*, and *WriteCleanFull*, are handled in a special manner as described in Section 6.3.4.3.1. For them, *CMDreqs WrNCPtl* and *WrNCFull* are utilized (see below). These may be issued to DMI or DII based on the address field in the original operation.
- *CMDreqs UpdInv, and UpdSCIn* may be used in the context of ACE and Proxy Cache to update the snoop filter (see Section 6.3.4.3.1). Although these refer to coherently accessed storage space, they are issued with CH = 0. These are sent to DCE.
- NC-CMDreqs *RdNC, WrNCPtl* and *WrNCFull* are issued with CH = 0 and also refer to non-coherently accessed storage space. These may be sent directly to DMI or DII.
- Cache management commands, *CMDreqs ClnVld, ClnShPsist, ClnInv, and MkInv* may be issued with CH = 0 or CH = 1. If they are issued with CH = 1, they refer to coherently accessed space and are sent to DCE. If they are issued with CH = 0, they refer to non-coherently accessed space and are sent to DMI or DII depending on the address accessed.

¹³ Note that in CHI, WrBk, WrCln, WrEvict, and Evict requests refer only to coherent storage and are issued over the CHI interface with SnpAttr = 1. However, since these operations are not snooped to other agents, their corresponding CMDreqs are issued to DCE with CH = 0, indicating no snooping is required

4.5.3.3.2 Cache Allocation Attributes

The following table shows cache allocation attributes.

Table 4-6: Access Ordering Attributes

Mnemonic	Name	Specification and Notes
AC	Allocation hint	0: Cacheline need not be allocated in the system cache. 1: Hint to allocate in System cache.
LK[1:0]	Line-Locking	00: No-op 01: Lock 10: Unlock 11: Reserved

4.5.3.3.3 Data format attributes

Table 4-7: Access Ordering Attributes

Mnemonic	Name	Specification and Notes
EN	Endianness	0: Little Endian access 1: Big Endian Access

4.5.3.3.4 Visibility and Ordering Attributes

Table 4-8 shows access ordering attributes

Table 4-8: Access Visibility and Ordering Attributes

Mnemonic	Name	Specification and Notes
VZ	Visibility	0: Coherency Domain visibility. 1: System visibility. Semantics are a function of the ST attribute: Set to 0 when EWA = 1 for CHI accesses, and for Bufferable (AxCACHE[0] == 1) accesses in AXI/ACE. Set to 1 for Device Writes (when ST = Device), Writethrough Writes, and "Persistent storage" Writes. Set to 1 when EWA = 0 for CHI accesses, and for non- Bufferable (AxCACHE[0] == 0) accesses in AXI/ACE. Also see Section 4.5.3.3.4.1.
OR[1:0]	Ordering	00: No ordering required for this access. 01: Write ordering at DII - keeps writes ordered, but allows posted writes to pass reads ¹⁴ . 10: This access must be strongly ordered with respect to a previous access from the same source to the same location. Also, writes from the same source must be observed in the same order by other agents in the system. 11: This access must be strongly ordered with respect to a previous access from the same source to the same endpoint device. Also ensure that conditions of 10 are met. Also see Section 4.5.3.3.4.2.

¹⁴ Please refer to system architecture on how this policy is being used and its application scope

4.5.3.3.4.1 VZ

When VZ=0, it is requested that the message semantics are made visible globally for the purpose of coherent storage access interactions.

When VZ=1, it is requested that the message semantics are made visible globally for all possible coherent or non-coherent storage access interactions.

4.5.3.3.4.2 OR[]¹⁵

An access is considered to be an **ordered access** if OR[] for it is set to either 10 or 11. Mutual ordering of two accesses from the same initiator is required if and only if both accesses are *ordered*.

Mutual ordering between *ordered* accesses from *different* initiators is not implied by the OR[] field.

Two accesses from a given initiator may have different ordering requirements per their respective OR[] field value. The actual extent of ordering enforced between such accesses is a function of the OR[] fields of the two accesses and their mutual order. Actual ordering enforced between two accesses are defined in Table 4-9.

Table 4-9: Actual ordering enforced between two accesses

First Access OR[]	Second Access OR[]	Actual Ordering Enforced
No Order	No Order	No Order
No Order	Request Order	No Order
No Order	Endpoint Order	No Order
Request Order	No Order	No Order
Request Order	Request Order	Request Order
Request Order	Endpoint Order	Request Order
Endpoint Order	No Order	No Order
Endpoint Order	Request Order	Request Order
Endpoint Order	Endpoint Order	Endpoint Order

Mutual ordering between two *ordered* Writes from the same initiator is required under OR[] = 10 or 11. Such an order must be enforced with respect to all observers in the system to the extent.

4.5.3.3.4.3 OR[] settings for CMD Requests

OR[] field is set to non-zero value only for the following CMD requests:

- RdNC
- RdNITC, RdNITCCI, RdNITCMI,
- WrNCPtl, WrNCFull,
- WrUnqPtl, WrUnqFull,
- WrStshPtl, WrStshFull,
- RdAtm, WrAtm, CompAtm, and SwapAtm.

¹⁵ This field is intended to cover the semantic of the Order field in the CHI™ interconnection architecture from ARM Inc.

For the above CMD requests, the OR[] is set to a non-zero value of 10 or 11.¹⁶

CMD request going to DCE will have OR[] bits set as ‘b00. This is a DCE assumes normal cacheable and bufferable memory.

Architecture Note

The ordering of Writes is facilitated via STRreq messages sent to initiator devices by target devices in response to CMDreq messages. A STRreq message from the target device signals the initiator that it may deliver Write data to it. (See section 4.11.)

An *atomic* Read is defined as being ***globally performed*** when no Write in the system to the same location can affect the value returned by the Read. An atomic Write is said to be *globally performed* when any Read in the system to the same cacheline receives the value deposited by the Write.

The target devices within the Ncore 3 platform represent ***Points of Serialization (POS)***, for storage accesses made via them, with *serialization* being enforced on a per cacheline basis. Accesses are taken through a POS one at a time. An access is defined as being *globally performed* with all the accesses made via the Ncore 3 platform once it passes through a POS.

Issuance of STRreq represents a *global performance event* for Write accesses. After the STRreq message is issued in response to a Write CMDreq, the target device has the responsibility to ensure that any Read accessing the location addressed by the Write but arriving after the issuance of STRreq, receives the value stored by the Write.

4.5.3.3.5 Access Security Attributes

Table 4-10: Access Security Attributes

Mnemonic	Name	Specification and Notes
NS	Non-secure	0: Secure access 1: Non-secure access
PR	Privileged access	0: Application access 1: Privileged access (Operating System or Hypervisor)

¹⁶ Mandated by CHI™ architecture requirement on Order[] field

4.5.3.3.6 Operation Semantic Extensions

The following table shows access attributes that indicate semantic extensions to operation.

EX and SS fields are needed for mutually exclusive sets of CMDreqs, and thus use a common field called “**ES**.” Table 4-11 describes the EX and SS bits individually.

Table 4-11: Operation Semantics Extensions

Mnemonic	Name	Specification and Notes
EX	Exclusive access	0: This is not an exclusive access. 1: This is an exclusive access.
SS	Self Snoop	0: No self-snoop. 1: Send a snoop for this operation to the requester also. Note: EX and SS fields are needed for mutually exclusive sets of CMDreqs, and thus use a common field called “ ES .”

Operation Semantic Extension also includes the specification of atomic update of the location accessed by the operation. The type of atomic update is specified via ArgV[5:0] field (see Section 4.5.3.4). Table 4-12 below shows the permitted encodings for the atomic updates.

Table 4-12: Access Semantic Extensions

Mnemonic	Name	Specification and Notes
ARGV[5:0]	Argument Vector	This attribute is used to specify argument(s) for semantic extensions, for example, OP-codes, of operations. Its interpretation is operation specific. In CCMP 1.0 this attribute is 6 bits long. In CCMP it specifies the atomic operation to perform for Read Atomic and Write Atomic operations. The codes are compatible with ARM CHI specifications, and are placed in ARGV[2:0]. 000: Add to location 001: Clear location 010: Exclusive OR with location 011: Logical OR with location 100: Replace the location with the maximum of the initial data in the location and the operand being carried by the transaction, treating both values as signed numbers 101: Replace the location with the minimum of the initial data in the location and the operand being carried by the transaction, treating both values as signed numbers 110: Replace the location with the maximum of the initial data in the location and the operand being carried by the transaction, treating both values as unsigned numbers 111: Replace the location with the minimum of the initial data in the location and the operand being carried by the transaction, treating both values as signed numbers

4.5.3.3.7 Trace and Debug Attributes

The following table shows access attributes that indicate semantic extensions to operation.

Table 4-13: Trace and Debug Attribute

Mnemonic	Name	Specification and Notes
TM	Trace Me	0: No-op 1: Trace this access.

4.5.3.3.8 Protocol response level attributes

Table 4-14: Access Ordering Attributes

Mnemonic	Name	Specification and Notes
RL[1:0]	Response Level Requested	<p>00: No response expected.</p> <p>01: Transport flow-control response expected.</p> <p>10: Protocol-level completion requested at the immediate destination of the message before a response is generated.</p> <p>11: Transitive Protocol-level completion before response is generated to present message.</p>

All CMDreq messages are sent with RL[1:0] = 0b01, which only returns the flow-control credit.

Add RL[1:0] usage by all message classes and any special circumstances (refer to flow diagrams and answers to Oski).

A response message must be issued only after the processing level requirement indicated in the RL[1:0] has been met.

4.5.3.3.8.1 Semantics of the RL[] field

- **00:** The sender of the message expects no response for the present message. All response messages (e.g. XYZ.rsp) have RL[] field set to 00.
- **01:** The sender of the message expects a Transport-level flow-control response. Such a response indicates that the intended destination unit has received the message.
- **10:** Protocol-level completion requested at the immediate destination of the message before a response is generated.
- **11:** A message may trigger one or more messages to be generated at the message's destination unit. These generated messages are considered children messages to the original message. The original message is referred to as the parent message with respect to these children messages. In general, this can repeat recursively a finite number of times, forming a tree of messages. Transitive Protocol-level completion request requires that the RL[] = 11 be transitively sent along with such children messages also. Transitive completion implies that the completion of the children message must be received before the parent message can be considered as having completed and a response is generated for it.

Implementation Note

Since RL[] field is always expected to be set to 0b00 for all the response messages, those messages need not carry this field explicitly in their bodies.

If any message with RL[] = 11 results at a unit in transactions on an external interface, then those transactions must be completed on the external interface to the extent required by the external interface architecture before the message can be responded to by that unit.

Table 4-15 shows the RL[] settings for various messages.

Table 4-15: RL[] settings for various CCMP messages

RL[] = 00	RL[] = 01	RL[] = 10	RL[] = 11
All responses (e.g. XYZrsp)	<ul style="list-style-type: none"> All CMD requests, except CMO commands^a All SNP requests All RBR and RBU requests All HNT requests All STR requests All DTRs under Owner-Sharer filter, except ones indicated under RL[] = 11 column All MRDRd<x> All TUN requests 	<ul style="list-style-type: none"> All DTW requests other than ones indicated in RL[] = 11 column All UPD requests All CMO commands^a MRDFlush, MRDClean, MRDInvalid All DTR requests under Null Filter CleanShared(Persist)^a CleanInvalid^a MakeInvalid^a ACE CleanUnique^a ACE MakeUnique^a 	<ul style="list-style-type: none"> DTWMrgMRD requests associated with WriteUniquePtlStash requests DTR requests associated with WriteUniqueFull-Stash requests from the stashing requesting AIU to Target AIU DTR requests caused by DTWMrgMRDs with RL[] = 11 DTR requests caused by Mrd with RL[] = 11 MrdRd* requests caused by stashing command
Notes:			
a. Changed for CONC-7391: DMI is not sending late response for Cmd = CMO			

4.5.3.3.8.2 Protocol Level Completion

“Protocol-level” in the context of RL[1:0] refers to the extent to which visibility semantics are implied by the VZ bit (see Section 4.5.3.3.4). Specifically, value of the Visibility attribute, VZ, carried by the message determines when the response message can be generated.

When VZ=0, the response must be generated only after guaranteeing that the message semantics are visible globally for the purpose of coherent interactions.

When VZ=1, the response must be generated only after guaranteeing that the message semantics are visible globally for all possible devices in the entire system, including those beyond the CCMP domain, that might be able to observe the effect of the present message.

Nv implies that the value of the attribute is derived from the Native Protocol operation that the CMD represents.

4.5.3.3.9 Attribute applicability for Command messages

Table 4-16: Attribute applicability per type of CMD

CCMP Operation	VZ	AC	CA	CH ^j	ST	EN ^a	ES	NS	PR ^b	OR	LK ^c	RL	TR
ReadNC	Nv ^m	Nv ^m	Nv ^m	0	Nv	0	EX ^d	Nv	Nv	Nv	00	01	Nv
ReadValid / ReadShared	0	Nv	1	1	0	0	EX ^d	Nv	Nv	00	00	01	Nv
ReadClean	0	Nv	1	1	0	0	EX ^d	Nv	Nv	00	00	01	Nv
ReadNITC	0/Nv	Nv	1	1	0	0	0	Nv	Nv	Nv	00	01	Nv
ReadUnique	0	Nv	1	1	0	0	0	Nv	Nv	00	00	01	Nv
ReadNotShDirty	0	Nv	1	1	0	0	EX ^d	Nv	Nv	00	00	01	Nv

Table 4-16: Attribute applicability per type of CMD

CCMP Operation	VZ	AC	CA	CH ^j	ST	EN ^a	ES	NS	PR ^b	OR	LK ^c	RL	TR	
CleanValid														
CleanShared	1 ^e	Nv	Nv	Nv	Nv	0	0	Nv	Nv	00	00	10	Nv	
CleanShared-Persist	1 ^e	Nv	Nv	Nv	Nv	0	0	Nv	Nv	00	00	10	Nv	
CleanInvalid	1 ^e	Nv	Nv	Nv	Nv	0	0	Nv	Nv	00	00	10	Nv	
MakeInvalid	1 ^e	Nv	Nv	Nv	Nv	0	0	Nv	Nv	00	00	10	Nv	
ReadNITCCI	0/Nv	Nv	1	1	0	0	0	Nv	Nv	Nv	00	01	Nv	
ReadNITCMI	0/Nv	0	1	1	0	0	0	Nv	Nv	Nv	00	01	Nv	
CleanUnique	0 ^k	Nv	1	1	0	0		EX ^d	Nv	Nv	00	00	01/10 ⁿ	
MakeUnique	0	Nv	1	1	0	0	0	Nv	Nv	00	00	01/10 ⁿ	Nv	
WriteUniquePtl	0/Nv	Nv	1	1	0	0	0	Nv	Nv	Nv	00	01	Nv	
WriteUniqueFull	0/Nv	Nv	1	1	0	0	0	Nv	Nv	Nv	00	01	Nv	
WriteCleanPtl	0 ^k	Nv	1	0	0	0	0	Nv	Nv	00	00	01	Nv	
WriteCleanFull	0 ^k	Nv	1	0	0	0	0	Nv	Nv	00	00	01	Nv	
WriteBackPtl	0	Nv	1 ^l	0	0	0	0	Nv	Nv	00	00	01	Nv	
WriteBackFull	0	Nv	1 ^l	0	0	0	0	Nv	Nv	00	00	01	Nv	
WriteEvictFull	0	1	1	0	0	0	0	Nv	Nv	00	00	01	Nv	
Evict	0	0	1	0	0	0	0	Nv	Nv	00	00	01	Nv	
WriteNCPtl	Nv ^m	Nv ^m	Nv ^m	0	Nv	0		EX ^d	Nv	Nv	Nv	00	01	Nv
WriteNCFull	Nv ^m	Nv ^m	Nv ^m	0	Nv	0		EX ^d	Nv	Nv	Nv	00	01	Nv
WriteStashPtl	0/Nv	Nv	1	1	0	0	0	Nv	Nv	Nv	00	01	Nv	
WriteStashFull	0/Nv	Nv	1	1	0	0	0	Nv	Nv	Nv	00	01	Nv	
LoadCchShared	0/Nv	Nv	1	1	0	0	0	Nv	Nv	00	00	01	Nv	
LoadCchUnique	0/Nv	Nv	1	1	0	0	0	Nv	Nv	00	00	01	Nv	
Read Atomic	0 ^f	1 ^g	1 ^g	Nv	0 ^h	0 ^a		SS ^d	Nv	Nv	Nv	00	01	Nv
Write Atomic	0 ^f	1 ^g	1 ^g	Nv	0 ^h	0 ^a		SS ^d	Nv	Nv	Nv	00	01	Nv
Compare and Swap Atomic	0 ^f	1 ^g	1 ^g	Nv	0 ^h	0 ^a		SS ^d	Nv	Nv	Nv	00	01	Nv
Swap Atomic	0 ^f	1 ^g	1 ^g	Nv	0 ^h	0 ^a		SS ^d	Nv	Nv	Nv	00	01	Nv
Prefetch	X	1	X	0	0	0	0	Nv	Nv	00	00	01	0	
DVMOp	0	0	0	0	0	0	0	0	0	00	00	01	Nv	
UpdInv	0	X	1	0	0	0	0	Nv ⁱ	0	00	00	01	0	
UpdSCln	0	X	1	0	0	0	0	Nv ⁱ	0	00	00	01	0	

Notes:

- a. In CCMP 1.0, only Little-Endian format is supported and should be set to 0.
- b. PR field is relevant when native interface is ACE, ACE-Lite, ACE5-Lite, or AXI. For CHI, this field is set to 0.
- c. LK[1:0] is intended for future use. Currently it is set to 00.
- d. EX: Can be set to 1 to indicate Exclusive access. Availability of support for Exclusives for ReadNC and WriteNCPt/Full operations is implementation dependent. SS: Can be set to 1 by an agent to demand Self-snoop.
- e. VZ bit is asserted (set to 1) by the AIU for CMOs CleanShared, CleanSharedPersist, and CleanInvalid. It indicates that the operation must enforce updates to the coherency granule thus far are made visible in granule's home location. This guarantees visibility to system agents that might use extra-CCMP domain paths to access the coherency granule contents. This may be achieved by propagating the CMO below the CCMP domain if a cache exists below if the interface below DMI and DII permit such commands. The CMO MakeInvalid is enforced in all the caches in the CCMP domain. It is propagated to caches below DMI and DII if the interfaces below them support an appropriate command.
- f. In CHI, Atomic operations may be specified with EWA = 0 or 1, however, the CCMP architecture supports only EWA = 1. The AIU overrides the native EWA value and sets VZ = 0 in the CMDreq. This also applies to Atomic operations arriving over an ACE5-Lite interface and corresponds to only supporting AxCACHE[0]=1.
- g. In CHI, Atomic operations may be specified with Cacheability = 0 or 1. However, the CCMP architecture supports only CA = 1. The AIU therefore overrides the native CA value and sets CA = 1 in the CMDreq. Also, the AC field in the CMDreq is set to 1 to permit allocation into a system cache, if one is present. This also applies to Atomic operations arriving over an ACE5-Lite interface.
- h. In CHI, the Device value can be set to 0 or 1 for Atomic operations. However, the CCMP architecture supports Atomic operations only on system memory locations. Hence the AIU sets ST = 0 in the CMDreq.
- i. If the operation is on behalf of a Proxy Cache, it supplies this field for its CopyBack Writes.
- j. CH bit to be considered as don't care for Ncore 3.x
- k. For ACE WriteClean can be 0 or 1 and CleanUnique is 1
- l. For ACE it is Nv
- m. For Non-coherent exclusive transactions set as 1
- n. For ACE

Outstanding JIRA 11152: May require update to VZ column!

In regards to the [NCOR-199: \[Mobileye\] Getting early write response \(forces flow\)](#), if we decide to make the RTL change where VZ is going to be 1 if axcache = 2 or axcache=0 when a transaction hits normal memory with NC bit set to 0, we need to update the VZ table.

Note: If the above type of transactions are illegal as mentioned in the arch spec, we don't have to make any RTL changes. If the above transaction is recommended instead, we should probably update the VZ for the below commands.

4.5.3.4 Multi-Purpose Field 1 (MPF1)

This field is used to carry CMDreq specific information. Table 4-17 shows the contents for various types of CMDreq types. For the ones not mentioned, the field is treated as don't care by the CCMP.

All values are issued right aligned within the MPF1 field, except the valid bit (when specified) must be the MSB of the field

Table 4-17: MPF1 semantics for CMDreq

CMDreq type	Information carried
WrStshPtl, WrStshFull LdCchShd, LdCchUnq	{Valid Stash Target} If Valid = 1, indicates Stash Target is the unique AIUID of the AIU that hosts the native agent that is the target of this stashing CMDreq. ^a
RdAtm, WrAtm, CompAtm, SwapAtm	{ArgV[5:0]} General purpose argument vector for CMDreq messages. Argument for Atomic transactions. See Section 4.5.3.3.6.
RdNC and WrNC issued by an AXI, ACE, ACE-Lite, or ACE5-Lite native agent	{BT[1:0] ASize[2:0] ALength[2:0]} BT[1:0]: Burst Type on native interface. 0b10 = WRAP; 0b01 = INCR; 0b00 = FIXED. ASize[2:0] specifies width of transfer on AXI, which equals $2^{(ASize[2:0])}$ bytes. Note: The size of transfer must be equal or smaller than the width of the native interface at the requesting AIU in bytes as specified by IntfSize[1:0] in the present CMDreq. If (number of bytes implied by ASize[]) < (number of bytes implied by IntfSize[]), the CMDreq represents a narrow transfer . ALength[2:0] indicates the number of beats specified in the AXI access. Number of beats = ALength[2:0] + 1. Note: Maximum number of beats = 8. Note: Concerto-C accesses must never cross a cache-line boundary. Total number of bytes accessed must not exceed the cache line size. Note: Native agent being AXI-, ACE-, ACE-Lite-, or ACE5-Lite-based can be detected by inspecting the TOF[] field of the CMDreq (see Section 4.5.3.9). Also see Section 4.5.3.4.1.
WriteUniquePtl and WriteUniqueFull	Used by only ACE agent to specify AWUNIQUE signal as the LSB bit. This is used by DCE to maintain the DCE snoop filter
All other:	Don't care.
Note:	
a. This field may alternatively carry implementation specific identifier of the stashing target.	

4.5.3.4.1 Handling Transfers at DMI and DII

4.5.3.4.1.1 ReadNC and WriteNC CMDreq with ST = 1

ReadNC and WriteNC CMDreqs issued with ST = 1 must be translated into corresponding operations with INCR Burst type with appropriate ALEN and ASIZE settings on the AXI native interface at the DMI or DII.

The resulting operation over AXI should faithfully represent the *actual amount of data accessed* as specified in Section 4.5.3.6.

4.5.3.4.1.2 ReadNC and WriteNC CMDreq with ST = 0

Nominally, the Size field in the CMDreq is used to derive the amount of the data to be accessed by the corresponding operations on the AXI native interface.

It is recommended, but not required, that ReadNC or WriteNC CMDreq indicating an INCR burst type should preserve the burst type when translating into a corresponding operation over the AXI native interface at the DMI or DII.

It is also recommended, but not required, that the resulting operation over AXI should faithfully represent the *actual amount of data accessed*, derived from the MPF1 field, instead of the Size field.

4.5.3.4.1.3 Handling Narrow Transfers at theDII

A RdNC or WrNC, with TOF = AXI or ACE, ST = 1, and with ASIZE[] < IntfSize[] in terms of implied number of bytes, represents a **non-modifiable, narrow transfer** access. The DII must faithfully reproduce such requests on its native AXI interface by utilizing the contents {BT[], ASIZE[], ALen[]} within the MPF1 field.

The only exception is when the native interface of the DII has data bus of width (say DII_DBus_Width) < ASIZE[]. In this case, the DII issues a request on its AXI bus with ASIZE[] equal to the width of its own data bus and the AxLEN[] set to

$$(((ALen[] + 1) * (ASIZE[] / (DII_DBus_Width))) - 1)$$

to access the same amount of data as the original request.

4.5.3.5 Multi-Purpose Field 2 (MPF2)

This field is used to carry CMDreq specific information. Table 4-18 shows the contents for different types of CMDreq types. For the ones not mentioned, the field is treated as don't care by the CCMP.

All values are issued right aligned within the MPF2 field, except valid bit (when specified) must be MSB of the field.

Table 4-18: MPF2 semantics for CMDreq

CMDreq type	Information carried
WrStshPtl, WrStshFull LdCchShd, LdCchUnq	{Valid Stash LPId Target[7:0]} If Valid = 1, indicates Stash Target is the unique processor within a native clustered agent hosted by the target AIU.
All Other:	{Valid FlowId[(Width of MPF2 - 2) : 0]}. If Valid = 1, indicates a unique flow within the initiator AIU. It may represent a unique Processor Id or Thread Id within a processor cluster, an IO Stream Id, or an AXI-ID. For CHI specific, FlowId = {3'b0, LPID[4:0] Or PGroupID[7:0] Or StashGroupID[7:0] Or TagGroupID[7:0]}

4.5.3.6 Size[2:0]

Nominal access size in bytes. The *nominal number of bytes accessed* = $2^{(\text{Size}[2:0])}$.

Thus, if $\text{Size}[2:0] = 0b011$, the number of bytes accessed equals $2^3 = 8$. All accesses in CCMP are Atomic Accesses (see Section 2.2). $\text{Size}[2:0] = 0b111$ is reserved.

4.5.3.6.1 Determining actual data accessed

The amount of **actual data accessed** by a CMDreq may be different than the nominal number of bytes accessed by it, as indicated above.

For a CMDreq indicating an INCR burst, the *actual data accessed* starts at the byte with address given by the Addr field, and its amount is indicated via the MPF1 field (see Section 4.5.3.4).

For a CMDreq not indicating an INCR burst the following rules must be applied to determine the *actual data accessed*:

For an access to memory storage ($\text{ST} = 0$ in CMDreq, see Table 4-5), the **actual data accessed** is the entire *container aligned block* of $2^{(\text{Size}[2:0])}$ bytes (see Section 2.2.1) that contains the address equal to the Addr field of the operation (see Section 4.5.3.2).

For an access to device storage ($\text{ST} = 1$ in CMDreq, see Table 4-5), the *actual data accessed* is the block of data that starts at the byte with address equal to the Addr field of the operation and includes all the higher addressed bytes within the *container aligned block* of size $2^{(\text{Size}[2:0])}$ bytes of the said address – up to and including the *ending byte* of the said *container aligned block* (see Section 2.2.1).

Note:

For non-modifiable narrow transactions with WRAP burst MPF1 field size must be used.

4.5.3.7 IntfSize[2:0]

Interface size in double-words at the initiator of the CMDreq. The interface size = 2^{IntfSize} double-words. In the case of CMDreq, it represents the width of the native data bus at the initiator AIU. 0: 1 DW, 1: 2 DWs; 2: 4 DWs; 3: 8 DWs; 4: 16 DWs; 5 - 7: Reserved.

IntfSize is used to produce good latency and bandwidth performance, by helping to order and format data to be transmitted to the requester so that there is no need for reorder buffers at the receiver, which improves area profile of the design.

The latency of the data delivery on the said interface is minimized since the data is preformatted to create a legal transfer thereon and thus can be transmitted without having to be reordered. This also improves bandwidth on the native interface as no idle cycles are created on the said interface due to need to reorder.

4.5.3.8 DId[]

Destination Id refers to either a target DMI-Id or DII-Id, based on the address map.

A coherent CMDreq identifies a DMI-Id, since only a DMI can be a target of a coherent access.

Application Note

Alternatively, this field may carry an implementation-specific identifier

This field ensures that Address map look-up need be performed only in the Initiator AIU and not repeated at a DCE or a snooper AIU to identify the home storage unit that is being accessed by the CMDreq.

For non-coherent CMD-reqs, the destinations themselves receive the CMDreq, and this field becomes a don't care.

Application Note

When used to transmit NUnitId of the destination unit, it is recommended that DId[MSB] indicates the type of target, where MSB represents the most significant bit number of the DId[MSB : 0] field:

- DId[MSB] = 0: The destination is a DMI.
 - DId[MSB] = 1: The destination is a DII.
-

If the transmitted DId[] value already is unique to destination unit, the above DId[MSB] value-based method may not be necessary to uniquely identify it.

4.5.3.9 TOF[2:0]

Transaction Ordering Framework to apply to the current CMDreq. The field refers to the native interface protocol over which the transaction was first issued into the system by the requester. It is set by AIUs based on the type of their native interface.

Examples are Concerto-C, CHI, AXI, PCI, etc. The value of this field modifies semantics of the Order[1:0] field.

Current values defined:

- **000:** Concerto-C ordering model. This model requires has the following ordering requirements:
 - Order of two native operations to a given cacheline address must be maintained throughout the lifetime of the operations. The system must maintain the necessary order among all the component CCMP messages that those two operations may engender to achieve the same.
 - Two messages of a given type from a given initiator Ncore unit to another given target Ncore unit are delivered to the target unit in the same order they were issued by the initiator unit.
- **001:** CHI ordering model. This model specifically covers the CHI A/B ordering models.
- **010:** ACE ordering model. This covers ACE-Lite and ACE5-Lite interfaces.
- **011:** AXI ordering model.
- **100:** PCI ordering model.
- **101 - 111:** Reserved.

Ncore 3 Requirement

AIUs must insert the appropriate value in TOF[2:0] field based on the type of their native interface

4.5.3.10 QoS

Quality of Service label field.

4.5.3.11 AUX

Auxiliary field carried by CMDreq. This field is used to carry User-defined information with a maximum size of 32 bits. It carries the REQ_RSVDC of the request channel from CHI interface for a CHI-AIU, or AXI/ACE AxUSER bits from an IOAIU.

4.5.3.12 NDProt

Protection bits for the non-data payload in the CMDreq message. The protection can be in the form of Parity or Error Correcting codes.

4.5.4 CMD Responses

A CMDrsp message does not transmit any protocol information; however, a CMDrsp message implies that a command message has been accepted by the DCE.

CMDrsp returns the flow-control credit to the issuer of the original CMDreq initiator.

4.5.5 CMDrsp messages

Two types of CMDrsp messages distinguish between the types of credit returned:

CCMDrsp: This type of CMD response message returns one credit for a coherent CMDreq. CCMDreq is sent by the DCE to an initiator AIU.

NCCMDrsp: This type of CMD response message returns one credit for a non-coherent CMDreq (NCCMDreq). NCCMDrsp are sent by a DMI, DII, or DVE to an initiator AIU.

NCCMDrsps are reserved for RdNC, WrNCptl, WrNCfull, and DVMOp CMDreq requests.

Table 4-19: Two types of CMDrsps

CMDReq Mnemonic	MsgType[7:0]
CCMDrsp	0xF0
NCCMDrsp	0xF1

4.5.6 CMDrsp message fields

Table 4-20 shows the fields in a CMDrsp message.

Table 4-20: CMDreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:				CCMP Message Header	
TargetId	wTargetId	6-12	O/I	Target Identifier	CMDreq.InitiatorId
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier	Local: FUnitId
CMTypE	wCMTypE	8	O/I	Type of Message	Local. See Table 4- 19.
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtect- tion	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTypE, and MessageId.	Local
CMHE:				CCMP Message Header Extension	
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:				CCMP Message Body	
RMessagId	wRMessagId	6-10	O/I	MessageId.	CMDreq.MessageId
CMStatus	wCmStatus	8	O/I		
{Pad}	wCMDRspPad		O/I		
NDPROT	wCMDRspPROT	0-10	O/I	Protection bits for CMDrsp Message	

4.5.7 CMDrsp field descriptions

4.5.7.1 RMessagId

Message Id of the CMDreq whose reception is being acknowledged.

4.5.7.2 CMStatus[7:0]

Refer to Table 4-4.

4.5.7.3 NDPROT

Refer to Section 4.5.3.12.

4.6 SNPreq and SNPrsp Messages

4.6.1 SNPreq messages

The scope of a coherent operation spans the entire coherency domain. Thus, a coherent operation issued by one agent can in general affect the state of its and every other cache¹⁷ in the coherency domain. To achieve this functionality, the coherent operation must also be presented to other agents in the system.

Upon the receiving a coherent CMDreq, the DCE queries its directory to identify other AIUs in the Ncore 3 system that may host native agents or proxy caches that might hold the cache states that require that they participate in the two actions above for the operation. The DCE then multi-casts a message, called **SNPreq**, to all selected AIUs. The AIUs so that they in turn may present the appropriate semantic contents of the original CMDreq to the agents or proxy caches.

The presentation and thereby viewing of these SNPreq messages or their derivatives by agents and proxy caches is called **snooping**.

The AIUs to which SNPreqs are sent are called **snoopers**. The AIUs translate the SNPreq messages into **snoop transactions** and issue them on the native interface to the coherent agents.

Table 4-21 below lists the SNPreq messages defined in the CCMP.

Table 4-21: SNPreq Messages

SNPreq Message (Mnemonic)	Operation Description
Snoop Clean with Data Reply (SnpClnDtr)	<p>Request for a clean copy of cacheline to the initiating AIU and/or transition to a compatible state.</p> <p>A snooped agent that has a cacheline copy in Dirty state, i.e., SD or UD, must transfer data to requester, and may optionally clean the data to memory. A snooped agent that has a copy in a UC state is permitted to transfer data.</p> <p>Final cacheline states in snooper: I, SC, SD.</p>
Snoop Valid with Data Reply (SnpVldDtr)	<p>Request a copy of cacheline in any valid state to the requesting AIU and/or transition to a compatible state.</p> <p>A snooped agent that has a cacheline copy in a Dirty state, i.e., SD or UD must transfer data. A snooped agent that has a copy in a UC state is permitted to transfer data.</p> <p>Final cacheline states in snooper: I, SC, SD.</p>
Snoop Invalid with Data Reply (SnpInvDtr)	<p>Request a copy of cacheline <i>in any valid state</i> to the requesting AIU and invalidate the cacheline state in snooper's cache.</p> <p>Final cacheline states in snooper: I.</p>
Snoop Clean with Data Write (SnpClnDtw)	<p>Request for the cacheline in dirty state to be cleaned to memory.</p> <p>A snooped agent that has a cacheline copy in a Dirty state, i.e., SD or UD must clean the data to memory.</p> <p>Final Cacheline States: Snooper: I, SC, UCE, UC.</p>
Snoop Invalid with Data Write (SnpInvDtw)	<p>Request for the cacheline in dirty state to be cleaned to memory and invalidate the copy of the cacheline.</p> <p>A snooped agent that has a cacheline copy in a Dirty state, i.e., SD or UD must clean the data to memory.</p> <p>Final Cacheline States: Snooper: I.</p>

¹⁷ These include coherent native agents with caches hosted by AIUs as well as proxy caches within the Ncore 3 system

Table 4-21: SNPreq Messages

SNPreq Message (Mnemonic)	Operation Description
Snoop Invalid (SnpInv)	<p>Request for the copy of the cacheline to be invalidated in situ in the cache. Dirty cacheline is NOT cleaned to memory prior to invalidation.</p> <p>Final Cacheline States: Snooper: I</p>
Snoop with No Intention To Cache (SnpNITCDtr)	<p>Request for a snapshot of current cacheline data. Requester does not intend to hold a coherent copy.</p> <p>Since the snapshot does not result in a new copy, no change to cacheline state is <i>required</i> by the snoop.</p> <p>A snooped agent that has a cacheline copy in a Dirty state, i.e., SD or UD must transfer data to requester, and may optionally clean the data to memory. A snooped agent that has a copy in a UC state is permitted to transfer data.</p> <p>Final cacheline states in snooper: No change required. I or any valid state subject to CCMP rules and invariants.</p>
Snoop with No SD Intervention (SnpNoSDIntDtr)	<p>Request a copy of cacheline in any valid state except SD to the requesting AIU and/or transition to a compatible state.</p> <p>A snooped agent that has a cacheline copy in a Dirty state, i.e., SD or UD must transfer data. A snooped agent that has a copy in a UC state is permitted to transfer data. Since SD state cannot be transferred to the requesting agent, cleaning of cacheline data to memory might be necessary.</p> <p>Final cacheline states in snooper: I, SC, SD.</p>
Snoop Invalidate and Stash (SnpInvStsh)	<p>Request for the copy of the cacheline to be invalidated in situ in the cache, as well as an invitation to receive an updated copy of the cacheline.</p> <p>To accept the invitation, the snooper AIU must assert the Snarf signal in its SNPrsp message.</p> <p>Note 1: Only <u>Both</u> CHI-B and CHI-E processors are expected to support this snoop. Other types must be presented with <i>Snoop Invalid</i>.</p> <p>Note 2: Only one snooper may be presented with this snoop.</p> <p>Final cacheline states in snooper: UD if stashing invitation is accepted; I otherwise.</p>
Snoop Unique Stash (SnpUnqStsh)	<p>Request for the copy of the cacheline to be invalidated, as well as an invitation to receive an updated copy of the cacheline.</p> <p>A snooped agent that has a cacheline copy in a Dirty state, i.e., SD or UD must first clean data to memory. To accept the invitation, the snooper AIU must assert the Snarf signal in its SNPrsp message.</p> <p>Note 1: <u>Both</u> CHI-B and CHI-E processors are expected to support this snoop. Only one snooper may be presented with the snoop equivalent to the SnpUnqStsh (see Table 4-23).</p> <p>Note 2: Per CHI-B/E specification, non-target CHI agents are required to be presented with <i>Snoop Unique</i> on their native interface under <i>SnpUnqStsh</i>. This architecture allows <i>SnpCleanInvalid</i> to be used instead. The ACE agents must be snooped with <i>CleanInvalid</i>. The subsequent actions of the AIU hosting a non-target agent correspond to those incurred for <i>SnpInvDtw</i> (see above in this table). I.e., any <i>snoop data response</i> from the agent is cleaned to memory via an appropriate Dtw request.</p> <p>Note 3: Final cacheline states in snooper: UD if stashing invitation is accepted; I otherwise.</p>
Snoop with No SD Intervention (SnpNoSDIntDtr)	<p>Request a copy of cacheline in any valid state except SD to the requesting AIU and/or transition to a compatible state.</p> <p>A snooped agent that has a cacheline copy in a Dirty state, i.e., SD or UD must transfer data. A snooped agent that has a copy in a UC state is permitted to transfer data. Since SD state cannot be transferred to the requesting agent, cleaning of cacheline data to memory might be necessary.</p> <p>Final cacheline states in snooper: I, SC, SD.</p>

Table 4-21: SNPreq Messages

SNPreq Message (Mnemonic)	Operation Description
Snoop Invalidate and Stash (SnpInvStsh)	<p>Request for the copy of the cacheline to be invalidated in situ in the cache, as well as an invitation to receive an updated copy of the cacheline.</p> <p>To accept the invitation, the snooper AIU must assert the Snarf signal in its SNPrsp message.</p> <p>Note 1: Only CHI-B processors are expected to support this snoop. Other types must be presented with Snoop Invalid.</p> <p>Note 2: Only one snooper may be presented with this snoop.</p> <p>Final cacheline states in snooper: UD if stashing invitation is accepted; I otherwise.</p>
Snoop Stash Shared (SnpStshShd)	<p>Invitation that the agent receives a <i>shared</i> copy of the cache line from system memory.</p> <p>To accept the invitation, the snooper AIU must assert the Snarf signal in its SNPrsp message.</p> <p>Note: Only Both CHI-B/E processors are expected to support this snoop. Other types must be presented with Snoop Clean Invalid.</p> <p>Note: At most one agent may be snooped with an invitation to stash.</p> <p>Final cacheline states in snooper: SC if stashing invitation is accepted; No change otherwise.</p>
Snoop Stash Unique (SnpStshUnq)	<p>Invitation that the agent receives a <i>unique</i> copy of the cache line from system memory or if it already has a copy, to upgrade its copy if was in a lower ownership level (I, SC, SD).</p> <p>To accept the invitation, the snooper AIU must assert the Snarf signal in its SNPrsp message.</p> <p>Note 1: Both Only CHI-B/E processors are expected to support this snoop. Other types must be presented with Snoop Clean Invalid.</p> <p>Note 2: At most one agent may be snooped with an invitation to stash.</p> <p>Final cacheline states in snooper: UC or UD if stashing invitation is accepted, no change otherwise. (Note: UD may be reached if the target agent is already in SD state and then receives data in UC state.)</p>
Snoop DVM Message (SnpDVMMMsg)	<p>Request to process DVM command.</p> <p>Final Cacheline States:</p> <p>Requester: NA. Snooper: NA.</p>
Snoop No Intention to Cache and Clean Invalidate (SnpNITCCIDtr)	<p>Request to the AIU for a snapshot of current cacheline data, with the Requester not intending to hold a coherent copy, followed by cleaning and invalidating the copy of the cacheline.</p> <p>The AIU presents Snoop Unique to the processors. After forwarding the data to requester AIU, if <i>intervention data</i> is Dirty, AIU also cleans it to memory.</p> <p>A snooped agent that has a cacheline copy in a Dirty state, i.e., SD or UD must transfer data to requester, and must also clean the data to memory. A snooped agent that has a copy in a UC state is permitted to transfer data. All snoopers must eventually invalidate their copies.</p> <p>Final cacheline states in snooper: I.</p>
Snoop No Intention To Cache and Make Invalidate (SnpNITCMIDtr)	<p>Request to the AIU for a snapshot of current cacheline data, with the Requester not intending to hold a coherent copy, followed by invalidating the copy of the cacheline.</p> <p>For CHI:</p> <p>The AIU presents Snoop Unique to the processors. After forwarding the data to requester AIU, the <i>intervention data</i> may be dropped, even if Dirty. A snooped agent that has a cacheline copy in a Dirty state, i.e., SD or UD must transfer data to requester. A snooped agent that has a copy in a UC state is permitted to transfer data. All snoopers must eventually invalidate their copies.</p> <p>For ACE:</p> <p>The AIU presents Snoop Clean Invalid to the processors. This forces dirty data to be written back, sending a copy to the requester after DCE has collected all outstanding snoop responses.</p> <p>Final cacheline states in snooper: I.</p>

4.6.2 Mapping of CMD requests to SNP requests

Table 4-22 below shows mapping of CMDreqs to SNPreqs. Note that multiple CMDreqs can map to the same SNPReq.

Table 4-22: Mapping of CMDreqs to SNPreqs and their codes

CMDreq Mnemonics	SNPReq Mnemonic	CCMP MsgType[7:0]
CmdRdCln	SnpClnDtr	0x41
CmdRdVld	SnpVldDtr	0x43
CmdRdUnq	SnpInvDtr	0x44
CmdClnVld, CmdClnShPsist	SnpClnDtw	0x48
CmdClnInv, CmdClnUnq, CmdWrUnqPtl, CmdWrAtm, CmdRdAtm, CmdSwapAtm, CmdCompAtm	SnpInvDtw	0x45
CmdMkInv, CmdMkUnq, CmdWrUnqFull	SnpInv	0x46
CmdRdNITC	SnpNITCDtr	0x42
CmdRdNShDty	SnpNoSDIntDtr	0x4A
CmdWrStshFull	SnpInvStsh	0x4B
CmdWrStshPtl	SnpUnqStsh	0x4C
CmdLdCchShd	SnpStshShd	0x4D
CmdLdCchUnq	SnpStshUnq	0x4E
CmdDVMMMsg	SnpDVMMMsg	0x4F
CmdRdNITCClnInv	SnpNITCCIDtr	0x50
CmdRdNITCMkInv	SnpNITCMIDtr	0x51
CmdWrEvict, CmdWrClnFull, CmdWrClnPtl, CmdWrBkFull, CmdWrBkPtl	These transactions do not cause snoops.	

4.6.3 Mapping of SNPreq to Native Snoop transactions

Table 4-23 below shows the mapping of SNPreq messages to various native protocol snoop transactions.

The ACE, ACE-Lite, ACE5-Lite column shows the channel — Read (R) or Write (W) — on which the requested transaction is expected to be issued. It also shows the legal values of AxBAR, AxDOMAIN, and AxSNOOP fields that may be used in the request transaction. Thus, for example {01/10} for ReadOnce indicates that ARDOMAIN field may be legally set to either 0b01 or 0b10 for the ReadOnce transaction. “:” is used to separate the fields. “|” is used to indicate concatenation of bits, which is necessitated by the increase in the size of the field in ACE5-Lite.

A proxy cache within an AIU also acts as a caching agent. Depending on implementation, it may be presented with CCMP, CHI, or ACE type snoops.

Table 4-23: Mapping of CCMP snoop messages to CHI-E/B and ACE snoop transactions

CCMP Snoop	Mappings to Native Snoop Transactions				
	CHI-B	CHI-E	CHI-E / CHI-B	ACE	
Mnemonic	Mnemonic	Mnemonic	Code [4:0]	Mnemonic	ACSNOOP[3:0]
SnpClnDtr	SnpClean	SnpClean	0x02	ReadClean	0b0010
SnpVldDtr	SnpShared	SnpShared	0x01	ReadShared	0b0001
SnpInvDtr	SnpUnique	SnpUnique	0x07	ReadUnique	0b0111
SnpClnDtw	SnpCleanShared	SnpCleanShared	0x08	CleanShared	0b1000
SnpInvDtw	SnpCleanInvalid	SnpCleanInvalid	0x09	CleanInvalid	0b1001
SnpInv	SnpMakeInvalid	SnpMakeInvalid	0x0A	MakeInvalid	0b1101
SnpNITCDtr	SnpOnce	SnpOnce	0x03	ReadOnce	0b0000
SnpNoSDIntDtr	SnpNotSharedDirty	SnpShared	0x04	ReadNotSharedDirty	0b0011
SnpInvStsh	SnpMakeInvalid-Stash <i>N/A: Stashing not supported</i>	SnpMakeInvalidStash <i>N/A: Stashing not supported</i>	CHI-B: 0x06 CHI-E: 0x06-	N/A: Stashing not supported	-
SnpUnqStsh	SnpUniqueStash	SnpUniqueStash <i>N/A: Stashing not supported</i>	CHI-B: 0x05 CHI-E: 0x05-	N/A: Stashing not supported	-
SnpStshShd	SnpStashShared	SnpStashShared <i>N/A: Stashing not supported</i>	CHI-B: 0x0C CHI-E: 0x0C-	N/A: Stashing not supported	-
SnpStshUnq	SnpStashUnique	SnpStashUnique <i>N/A: Stashing not supported</i>	CHI-B: 0x0B CHI-E: 0x0B-	N/A: Stashing not supported	-
SnpDVMMsg	SnpDvmOp	SnpDvmOp	0x0D	DVMOp ^a	0b1111
SnpNITCIDtr	SnpUnique	SnpUnique	0x07	CleanInvalid ^b	0b1001b
SnpNITCMIDtr	SnpUnique	SnpUnique	0x07	CleanInvalid ^b	0b1001b
Note:					
a. Also sent to AIUs with ACE-Lite and ACE5-Lite native interfaces with SMMU-TCU attached b. ACE does not implement SnpUnique , the agent interface must remap the message					

4.6.3.1 Adjustments for Stashing Snoops

The stashing commands identify the intended target AIU into whose agent the cacheline is desired to be installed. The DCE is aware whether the intended target AIU's agent supports stashing.

Ncore 3 Requirement

If the target does not support stashing, then in the case of CmdLdCchShd and CmdLdCchUnq commands, no snoops are generated, and the corresponding transactions are completed normally and without error indication

In all other cases, the DCE issues identical snoops to all the necessary AIUs (see Section 4.6.3.2.2). The snoops indicate the intended target AIU (see Section 4.6.5.7).

The CHI architecture requires that a distinction be made between the snoops sent to target agents and non-target agents. This distinction is made by the agent's AIU.

Ncore 3 Requirement

The AIU is aware of its own NUID¹⁸ as well as the agent's ID that must be used for the purpose of stashing snoops¹⁹

The AIU compares the stashing target ID in the snoop with its own to determine if its agent is the intended target of the snoop.

A snooping AIU adjusts the actual snoop sent to the native agent as a function of:

- The type of snoop
- The type of native agent
- Whether its agent is the target of the stashing snoop

Table 4-24 shows the distinction between the CCMP snoop types exercised by AIUs depending on whether their agent is a target or not a target for stashing. Such an agent could be embedded inside the AIU – as in the case of a Proxy Cache capable of stashing – or a native agent.

Table 4-24: Snoops exercised by Target and non-Target AIUs

Stashing CCMP snoop issued by the DCE	CCMP snoop type exercised by the AIU of the target agent	CCMP snoop type exercised by the AIU of the non-target agent
SnpInvStsh	SnpInvStsh	SnpInv
SnpUnqStsh	SnpUnqStsh	SnpInvDtw
SnpStshShd	SnpStshShd	SnpClnDtw
SnpStshUnq	SnpStshUnq	SnpInvDtw

The AIU translates these adjusted CCMP snoop message types to native snoop transactions per [Table 4-23](#), [Table 4-24](#), [Table 4-25](#). [Table 4-25](#) below shows the actual native snoop chosen by the AIU as a function of these considerations for the different native interfaces.

If the stashing snoop is ultimately issued to the native agent, the agent's stashing identifier is included in that snoop. For CHI, this identifier is LPID.

¹⁸ See Ncore 3 System Architecture Specification

¹⁹ It may be possible in the implementation to set the agent's ID for stashing to be identical to the NUID of the AIU.

Table 4-25: Adjustments of snoops for stashing

CCMP Stashing Snoop	Adjustments to CCMP Snoop Transactions		
	CHI-B	CHI-E	ACE
Mnemonic	Mnemonic	Mnemonic	Mnemonic
SnpInvStsh	To CHI-B Target ^a : SnpMakeInvalidStash To non-Targets ^b : SnpInv	To CHI-E Target^e: SnpMakeInvalidStash To non-Targets^b: SnpInv To CHI-E Target and all non-Targets: SnpInv^b	To ACE Target and all non-Targets: SnpInv ^b
SnpUnqStsh	To CHI-B Targets ^b : SnpUniqueStash To non-Targets ^b : CHI-B snoop: SnpUnique. AIU operation: SnpInvDtw equivalent ^c .	To CHI-E Target^e: SnpUniqueStash To CHI-E Target and all non-Targets^b: To non-Targets^b: CHI-E snoop: SnpUnique. AIU operation: SnpInvDtw equivalent^c CHI-E snoop: SnpUnique. AIU operation: SnpInvDtw equivalent^e	To ACE Target and all non-Targets ^b : ACE snoop: ReadUnique. AIU operation: SnpInvDtw equivalent ^c .
SnpStshShd	To CHI-B Target ^b : SnpStashShared If CHI-B Target and Snarf=1, to non-Targets ^b : CHI-B snoop: SnpShared AIU operation: SnpClnDtw equivalent ^c .	To CHI-E Target^e: SnpStashShared If CHI-E Target and Snarf=1, to non-Targets^b: CHI-E snoop: SnpShared AIU operation: SnpClnDtw equivalent^c If CHI-E Target: All snoops suppressed^d If CHI-B Target and Snarf=1, to non Targets^b: CHI-E snoop: SnpShared AIU operation: SnpClnDtw equivalent^e	If ACE Target: All snoops suppressed ^d . If CHI-B/E Target and Snarf=1, to non-Targets ^b : ACE snoop: ReadShared AIU operation: SnpClnDtw equivalent ^c .

Table 4-25: Adjustments of snoops for stashing

SnpStshUnq	To CHI-B Target ^b : SnpStashUnique If CHI-B Target and Snarf=1, to non-Targets ^b : CHI-B snoop: SnpUnique. AIU operation: SnpInvDtw equivalent ^c .	<u>To CHI-E Target^e:</u> <u>SnpStashUnique</u> <u>If CHI-E Target and</u> <u>Snarf=1, to non-Targets^b:</u> <u>CHI-E snoop: SnpUnique.</u> <u>AIU operation:</u> <u>SnpInvDtw equivalent^c, If CHI-E</u> <u>Target:</u> <u>All snoops suppressed^d.</u> <u>If CHI-B Target and</u> <u>Snarf=1, to non Targets^b:</u> <u>CHI-E snoop: SnpUnique.</u> <u>AIU operation:</u> <u>SnpInvDtw equivalent^e.</u>	If ACE Target and non-Targets: All snoops suppressed ^d . If CHI-B/E Target and Snarf=1, to non-Targets ^b : ACE snoop: ReadUnique. AIU operation: SnpInvDtw equivalent ^c .
Note:			

a. The AIU is expected to be aware of whether it has a CHI-B or CHI-E native interface, as well as the Cache Identifiers assigned to the caches belonging to the native agents it hosts and adjust the snoop codes appropriately as indicated in this table.

- When the CHI-B AIU's outstanding transaction credits for stashing are not available, the CHI-B AIU sets DoNoTDataPull bit in the snoop request and forwards it to CHI SNP channel for Read Stash (Independent Stash) and Write Stash (Write with Hint).
- When the CHI-B AIU's outstanding transaction credits for stashing are available, the CHI-B AIU forwards the snoop request to CHI SNP channel

b. Stashing transactions are not supported in CHI-E or ACE. The WriteUniqueStash operation must therefore be handled as a WriteUnique operation.

c. SnpUnique and SnpShared snoops on CHI is required by that architecture. However, the snooping AIU behaves per SnpInvDtw or SnpClnDtw per context, cleaning the cacheline to memory, if necessary, and NOT issuing a DTRreq.

d. DCE is expected to be aware if a given C-AIU is ACE, CHI-E (non-Stash-capable) or CHI-B/E (Stash-capable). If the target is ACE or CHI-E (non-Stash-capable), the snoop process is suppressed within the DCE since CHI-E it does not support StashOnce transactions.

e. The AIU is expected to be aware of whether it has a CHI-E native interface, as well as the Cache Identifiers assigned to the caches belonging to the native agents it hosts and adjust the snoop codes appropriately as indicated in this table.

- When the CHI-E AIU's outstanding transaction credits for stashing are not available, the CHI-E AIU declines it right away by terminating the snoop stash request with SNPrsp CMstatus=8'h0 as per described in [CONC-13156] for Read Stash (Independent Stash), and downgrade the stash snoop to non-stash snoop request and forwards it to the CHI SNP channel for Write Stash (Write with Hint).
- When the CHI-E AIU's outstanding transaction credits for stashing are available, the CHI-E AIU forwards the snoop request to CHI SNP channel

4.6.3.2 Issuing snoops to AIUs

4.6.3.2.1 Owners and Sharers

The directory tracks Owner and Sharer states of a cacheline. It also tracks Reservation states among the AIUs.

An AIU is marked in the directory as an *owner* of a cacheline when, if as a result of an operation executed by its agent or via a stash operation executed by an external agent, it receives for the cacheline a *UD*, *UC*, or *SD* state.

A *sharer* AIU, on the other hand, can only be in the *SC* state. Note that an AIU in *I* state but with an active Reservation is also treated as being in *sharer* state (see Section 2.4.2.1).

[Table 4-26](#) shows the legal numbers of Owner and Sharer states among AIUs of the system and the possible states within those AIUs.

Table 4-26: Legal distribution of states among AIUs

# of Owners	# of Sharers	Possible Owner States	Possible Sharer States
1	0	UD, UC, SD	-
1	≥ 1	SD	SC
0	≥ 1	-	SC

When there is a single AIU in the system that contains the copy of a cacheline, with no active Reservations for the cacheline in other AIUs, the AIU is referred to as having a ***Unique Presence*** of the cacheline. Note that in such a unique AIU, the cacheline can be in either *owner* or *sharer* state.

4.6.3.2.2 Choosing AIUs to snoop

The number of AIUs snooped and which ones, is a function of the snoop type. Table 4-27 shows this information by each snoop type.

Note that in Table 4-27, *All Valid* includes those AIUs whose agents may hold an *I* state with an active Reservation.

Table 4-27: Snooper choices for per type of SNPreq

SNPReq Mnemonic	AIUs to snoop
SnpClnDtr	Owner (if present)
SnpVldDtr	Owner (if present)
SnpInvDtr	All Valid
SnpClnDtw	Owner (if present)
SnpInvDtw	All Valid
SnpInv	All Valid
SnpNITCDtr	Owner (if present)
SnpNoSDIntDtr	Owner (if present)
SnpInvStsh	Target AIU + All Valid
SnpUnqStsh	Target AIU + All Valid
SnpStshShd	Target AIU + Owner (if Owner present & Snarf == 1)

Table 4-27: Snooper choices for per type of SNPReq

SNPReq Mnemonic	AIUs to snoop
SnpStshUnq	Target AIU + All Valid (if Snarf == 1)
SnpDVMMMsg	All C-AIUs + All NC-AIUs with SMMU-TCUs
SnpNITCCIDtr	All Valid
SnpNITCMIDtr	All Valid

Architecture Note

In the case of SnpClnDtr, SnpVldDtr, SnpNITCDtr, SnpNITCIDtr, SnpNITCMIDtr, and SnpNoSDInt:

- If an agent is in the *Owner* state, it must be sent the snoop.
- In CHI-B, an agent in *Shared* state can be induced to respond to the snoop with a data transaction if RetToSrc signal is asserted. However, CCMP 1.0 does not make use of this signal.
- An ACE agent in *Shared* state is permitted to transfer data, although it is not required to. Implementations may use this fact to transfer data to a requester by issuing a snoop to such an agent if and only if no agent is in *Owner* state.

In the case of SnpClnDtw:

- If an agent is in the *Owner* state, it must be sent the snoop.
 - It is not required that agents in *Shared* state be sent snoops.
-

4.6.4 SNPreq message fields

[Table 4-28](#) shows the fields in an SNPreq message.

Table 4-28: CCMP SNPreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:		CCMP Message Header			
TargetId	wTargetId	6-12	O/I	Target Identifier	Local: Snooper FUnitId
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier	Local: DCE FUnitId
CMTypE	wCMTypE	8	O/I	Type of Message	Local. See Table 4-22.
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTypE, and MessageId.	Local
CMHE:		CCMP Message Header Extension			
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:		CCMP Message Body			
CMStatus	wCmStatus	8	I	Input-only for SNPreq. Used to deliver Transport encountered errors to the Target unit.	
Addr	wAddr	32-52	O/I	Address	CMDreq.Addr
VZ	wVZ	1	O/I	Visibility	CMDreq.VZ
CA	wCA	1	O/I	Cacheable	CMDreq.CA
AC	wAC	1	O/I	Allocate Hint	CMDreq.AC
NS	wNS	1	O/I	Non-secure Access	CMDreq.NS
PR	wPR	1	O/I	Privileged Access	CMDreq.PR
UP	wUP	1	O/I	Unique Presence	Local
RL	wRL	2	O/I	Response Level[1:0]	Local
EO	wEO	1	O/I	Exclusive Okay	Local
TM	wTM	1	O/I	Trace Me	CMDreq.TM
MPF1	wMpf1	6-12	O/I	Multi-purpose field #1	If Stash: CMDreq.MPF1 If Non-stash: CMDreq.FUnitId
MPF2	wMpf2	6-12	O/I	Multi-purpose field #2	If Stash: CMDreq.MPF2 If Non-stash: CMDreq.MessageId

Table 4-28: CCMP SNPreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
MPF3	wMPF3	8-10	O/I	Multi-purpose field #3. If UP[1:0] == 0b00, 0b01, or 0b10, Don't care. If UP[1:0] == 0b11, Identifier of the AIU designated to send SC-intervention to the requesting AIU.	Local
IntfSize	wIntfSize	3	O/I	Interface Size	CMDreq.IntfSize
DId	wDId	2-6	O/I	Destination Id	CMDreq.DId. (Don't care for DVM snoops.)
TOF	wTOF	3	O/I	Transaction Ordering Framework.	CMDreq.TOF
QoS	wQoS	4	O/I	Quality of Service.	CMDreq.QoS
RBID	wRBID	6-10	6-10	Request Buffer Id assigned for this snoop.	Local
AUX	wReqAux	0-8	O/I	Auxiliary field	CMDreq.AUX ^a
MPROT	wReqProt	0-10	O/I	Protection bits for Snoop Message	Local
Note:					
a. Both DVM snoops receive the same AUX field contents.					

4.6.5 SNPreq field descriptions

4.6.5.1 CMStatus

See Section 4.5.3.1.

4.6.5.2 Addr

See Section 4.5.3.2.

The Addr field in SNPreq is copied from the CMDreq it represents.

4.6.5.3 Operation attributes

Attributes are associated with snoops as well.

Operation attributes are carried by the SNPreq message in the following fields: VZ, AC, CA, NS, PR, UP, RL, and TM.

For all the above attributes except UP, see Section 4.5.3.3.

4.6.5.4 UP[1:0]

The DCE consults the directory for the address in CMDreq and detects if there is a single snooper AIU for the current CMDreq. If so, it uses this attribute to indicate so to the snooper AIU. Note that this evaluation must also consider states of Reservation in agents (see Section 2.4.2.1).

This field is utilized to improve coherent Read performance in CCMP.

In general, the snoop request semantics, the native protocol of the snooper, and the snoop response received must be taken into account in producing the CCMP snoop response messages in acting in accordance with the UP[1:0] value in the SNPreq.

Unless the snoop response from the agent definitively indicates absence of other copies in the system, it should be assumed that other copies exist. For example, for an ACE interface, a snoop response of WasUnique = 1 there was exactly 1 copy of the cacheline present in the agent on the present ACE interface. If, IsShared = 0 also in the snoop response, that indicates that there are no valid copies of the cacheline left in the system after the present snoop.

Table 4-29: Snoop Attributes

Mnemonic	Name	Specification and Notes
UP[1:0]	Unique Presence / Unique Provider	<p>Set for snoops.</p> <p>0b00: Unique Presence / Unique Provider / Unique Permission mechanisms are NOT invoked.</p> <p>0b01: Unique Presence: If set to 0b01, snooper has the only copy of the cacheline in the system (unique presence) and thus the only AIU to receive the present snoop. Under this circumstance, the snooper AIU may be able to upgrade the snoop response for a Read type command. (See Read Acceleration - ???)</p> <p>0b10: Unique Provider: If set to 0b10, it indicates that the AIU receiving the snoop is allowed to provide data response to the requester. This setting is used when only one AIU is being snooped. The setting indicates that <i>additional shared copies of the cacheline exist in the system</i>, which are not being snooped.</p> <p>0b11: Unique Permission: If set to 0b11, it indicates that only the AIU identified in the MPF3 field is authorized to provide data response to the requester. This becomes necessary when the snoop is to be sent to multiple snoopers and only one of them is to be authorized to provide a DTR response.</p>

4.6.5.4.1 UP[1:0] = 0b00

This is the default value for the attribute. This value carries no specific semantic meaning to be used during the snoop process and in producing snoop response.

It can be used for SNPreqs corresponding to any CCMP CMDreq.

In the case of Read snoops, UP[10] = 0b00 allows the snooper AIU to send a DTRreq to the requesting AIU if the snooper agent provides a data transfer to its AIU.

This value is used when an Ncore system does not implement a directory (i.e. a **Null** filter).

4.6.5.4.2 UP[1:0] = 0b01

This setting of the field is used for Read type snoops.

This value indicates a **Unique Presence** of the cacheline at the snooper agent. The value is used when snooping a *sole* or *singleton* agent in the system that has a copy of the cacheline. Since it is a singleton copy, the snoop with UP[1:0] = 0b01 is sent only to the agent that is holding that copy.

Unique presence may correspond to any of the valid states of the cacheline in an agent: UD, UC, SD, or SC.

Note that the *Unique Presence* setting must NOT be used when the snoop is sent to a single snoopee which is not a singleton copy of the cacheline.

Since there is a singleton copy, the snoop is only issued to the AIU holding it.

Indicating a singleton copy to the AIU offers an opportunity to *upgrade* the state of the cacheline delivered to

the requesting agent via a DTR. The value 0b01 is used to indicate to an AIU that its agent has the sole copy of the cacheline.

The state of the cacheline provided via a DTR is a function of the type of snoop and the actual snoop response provided by the agent. The state of the cacheline may be upgraded with the additional knowledge that no other agent in the system has a copy of the said cacheline. For example, state of the cacheline can be upgraded from SC to UC for a SnpClean snoop if the snooper indicates that as a result of the snoop, it has invalidated the copy of the cacheline.

4.6.5.4.3 UP[1:0] = 0b10

This setting of the field is used for Read type snoops.

UP[1:0] value of 0b10 is used when the *snoop is being sent to only one of the AIUs*. This value bestows a **Unique Provider** status upon one of the AIUs carrying the copy, allowing it to provide a DTRreq in response to the snoop.

It is chosen when:

- More than one agent – and thus AIUs – have a copy of the cacheline. That is, UP[1:0] = 0b01 cannot be asserted for the snoop.
- The type of snoop permits it to be presented to exactly one on the AIUs that carries the copy.

The snoop must be sent with the following preference order:

- If an owner exists, the snoop *must* be sent to it. This is to ensure Dirty to Clean state transition at the owner, if required by the given snoop.
- Only if an owner doesn't exist, the snoop can be sent to one of the sharers.

The cacheline state carried by the DTR under Unique Provider can be the highest one that can be delivered that is compatible with:

- The actual response from the local agent. The local agent might indicate that it is *passing* the *Dirty* state of the cacheline or might not
- Potential highest state of the cacheline that might be retained among agent caches as a result of the current snoop

Table 4-30 shows the highest state retained local and peer agents as a result of SNPreqs for Read commands under the condition that the local snooper issues an *intervention* data transfer. The table shows columns for two possibilities: *Pass Dirty* in the local response is negated or asserted.

Table 4-30: Highest Cacheline state retained in an agent cache

Read Command	Snoop Request	Local Response: Data Transfer + Pass Dirty = 0		Local Response: Data Transfer + Pass Dirty = 1	
		Highest Agent cacheline state	Highest Cacheline State in DTRreq	Highest Agent Cacheline State	Highest Cacheline State in DTRreq
ReadNITC	SnpNITCDtr	SD	I	SC	I
ReadNITC Clean Invalid	SnpNITCCIDtr	I	I	I	I
ReadNITC Make Invalid	SnpNITCMIDtr	I	I	I	I
ReadClean	SnpClnDtr	SC	SC	SC	SC
ReadShared	SnpVldDtr	SD	SC	SC	SC
ReadNotSharedDirty	SnpNoSDInt	SD	SC	SC	SC ^a
ReadUnique	SnpInvDtr	I	UC	I	UD
Note:					
a. SD would be compatible, but “Not Shared-Dirty” prohibits it.					

Architecture Note

Note that setting the UP[1:0] = 0b01 or 0b10 does not guarantee that the agent will issue Data intervention and its AIU will thus issue a DTR. The DCE must still examine the snoop response and supply the data to the requester if the AIU has not.

4.6.5.4.4 UP[1:0] = 0b11

This setting of the field is used for Read type snoops.

It is chosen when there are multiple agents, and thus AIUs, with a copy of cacheline, *all of which must be snooped*, but only one should be permitted to issue a DTRreq to the requesting AIU.

This value is used to indicate a **Unique Permission** provided to one AIU from among multiple potential providers of a DTRreq in response to a snoop.

When UP[1:0] set to 0b11, the MPF3 field in the SNPreq identifies the AIUid of the AIU chosen to provide the DTR. Only the AIU whose Id matches the AIUid in the MPF3 field issues the necessary DTRreq.

The AIUUD in MPF3 must be set with the following preference order:

1. If an owner exists, the AIUID in the snoop *must* identify it. This is to ensure *Dirty* to *Clean* state transition at the owner, if required by the given snoop.
2. Only if an owner doesn't exist, the snoop can carry the AIUID of one of the sharers.

Application Note

An example of use of Unique Permission setting is a ReadUnique snoop in presence of multiple SC copies of the cacheline under the ACE architecture. All the agents with a copy of the cacheline must be snooped. But since all the snoopers could potentially send a DTR in response to the snoop, the DCE can choose one of the agents and designate it as the provider of a DTR. It does so by sending the *Unique Permission* setting in UP[1:0] and providing the Id of the selected AIU in the MPF3 field.

After receiving a data intervention from the agent, the designated AIU can send a DtrRdUCln to the requesting AIU.

4.6.5.5 RL[1:0]

The RL[1:0] field in SNPreq message is set to 0b10, which requires that the response be at the coherency protocol level.

The CMStatus[7:0] field in SNPrsp thus contains the result of snooping the agent and thus reflects the snoop response received from that agent.

4.6.5.6 EO

Exclusive Okay.

The DCE sets or resets this bit for a snoop message of an Exclusive Read CMD request (see Section 4.5.3.3.6).

EO is set when the Exclusive Monitor in the DCE determines that the Exclusive Read has performed successfully at the **point of serialization**. If the Exclusive Read is not performed successfully, the EO bit is reset.

If the DCE has indicated Exclusive Read success, it must track the Exclusive success status of the coherency granule within the requesting agent until that status is reset due to the successful completion of its Exclusive sequence, it had been interrupted because another agent completed its Exclusive sequence, or another agent had unconditionally updated the said granule.

If DCE cannot track the Exclusive status of an agent, it must set the EO bit to zero in SNPreq, MRDreq, and RBRreq messages it might use in connection with said Exclusive Read CMDreq.

Exclusive Okay flag is returned to the agent along with its data to indicate the success of its Exclusive Read transaction.

The EO bit in the SNP request is transferred to the EO bit in the DTRreq issued to the requesting AIU, if the snooper issues such a DTRreq message (see Section 4.7.4).

For non-Exclusive accesses, the EO bit is don't care and must be set to 0.

4.6.5.7 Multi-Purpose Field-1 (MPF1)

This field is used to carry SNPreq specific information. Table 4-31 shows the contents for various types of SNPreq types. For the ones not mentioned, the field is treated as don't care by the CCMP.

All values are issued right aligned within the MPF1 field, except valid bit (when specified) must be MSB of the field.

Table 4-31: MPF1 semantics for SNPreq

SNPreq type	Information carried
SnpInvStsh, SnpUnqStsh, SnpStshShd, SnpStshUnq	{Valid Stash Target} The Valid is a single bit field which is the most significant bit of the MPF2 field. The LPlid is zero-extended to occupy the less significant bits of the MPF2 field. If Valid = 1, indicates Stash Target is the unique AIUID of the AIU that hosts the native agent that is the target of this stashing. If Valid = 0, the AIU receiving the snoop concludes a mismatch regardless of the actual AIUID value.
SnpNoSDInt, SnpClnDtr, SnpVldDtr, SnpInvDtr, SnpNITCDtr, SnpNITCCIDtr, SnpNITCMIDtr	TargetId[] (Section 4.4.2.1) for the DTRreq that will be used to forward the data to the requester AIU.
SnpDVMOp	Carries VMIDExt field for CHI DVMOp snoops.
All other:	Don't care.

4.6.5.7.1 Use in SnpInvStsh, SnpUnqStsh, SnpStshShd, and SnpStshUnq

The field may be copied from MPF1 in CMDreq for which this is the snoop or filled in by the DCE based on the directory state knowledge that may indicate which AIU is the target of the stash snoop.

If the Valid bit is set, this field indicates which AIU is the *Target* or candidate for stashing. If the Valid bit is not set, no AIU is the candidate for stashing. AIUs must adjust the native snoop transactions per Table 4-23 based on whether they are a target or not.

4.6.5.7.2 Use in SnpNoSDInt, SnpClnDtr, SnpVldDtr, SnpInvDtr, SnpNITCDtr, SnpNITCCIDtr, SnpNITCMIDtr

This is copied from the InitiatorId[] in the CMDreq header (see Section 4.4.2.2) for which this is the snoop. This value is used as TargetId[] in the header of the data response message (DTRreq) carrying intervention data, if it is delivered by the native agent.

4.6.5.8 Multi-Purpose Field-2 (MPF2)

This field is used to carry SNPreq specific information. Table 4-32 shows the contents for various types of SNPreq types. For the ones not mentioned, the field is treated as don't care by the CCMP.

All values are issued right aligned within the MPF2 field, except valid bit (when specified) must be the MSB of the field.

Table 4-32: MPF2 semantics for SNPreq

SNPreq type	Information carried
SnpInvStsh, SnpUnqStsh, SnpStshShd, SnpStshUnq	{Valid Stash LPId} The Valid is a single bit field which is the most significant bit of the MPF2 field. The LPId is zero-extended to occupy the less significant bits of the MPF2 field. If Valid = 1, indicates Stash Target is the unique Logical Processor within the native agent that is the target of this stashing. If Valid = 0, it is implementation dependent whether the target agent accepts the stashing request. The field is copied from MPF1 in CMDreq for which this is the snoop.
SnpNoSDInt, SnpClnDtr, SnpVldDtr, SnpInvDtr, SnpNITCDtr, SnpNITCIDtr, SnpNITCMIDtr	RMessageld[] for the DTRreq that will forward the data to the requester AIU. This is copied from the Messageld[] in the CMDreq header (see Section 4.4.2.4) for which this is the snoop.
SnpDVMOp	Carries a globally unique identifier for the DVMOp that is being snooped. If a DVMOp snoop is issued in multiple portions, all snoop portions belonging to the DVMOp must carry the same value in this field.
All other:	Don't care.

4.6.5.8.1 Use in SnpInvStsh, SnpUnqStsh, SnpStshShd, SnpStshUnq

The field may be copied from MPF2 in CMDreq for which this is the snoop or filled in by the DCE based on the directory state knowledge that may indicate which AIU is the target of the stash snoop and the Logical Processor Id configured within the DCE for the target AIU's native coherent agent.

If the Valid bit is set, this field indicates which Logical Processor is the *Target* or candidate for stashing. If the Valid bit is not set, no Logical Processor is the candidate for stashing.

AIUs must adjust the native snoop transactions per Table 4-23 based on whether a valid Logical processor is identified or not.

4.6.5.8.2 Use in SnpNoSDInt, SnpClnDtr, SnpVldDtr, SnpInvDtr, SnpNITCDtr, SnpNITCIDtr, SnpNITCMIDtr

This is copied from the Messageld[] in the CMDreq header (see Section 4.4.2.2) for which this is the snoop. This value is used as RMessageld[] field in the data response message (DTRreq) carrying intervention data, if it is delivered by the native agent.

4.6.5.9 Multi-Purpose Field-3 (MPF3)

This field is used to carry SNPreq specific information.

Table 4-33: MPF3 semantics for SNPreq

SNPreq type	Information carried
SnpNoSDIntDtr, SnpClnDtr, SnpVldDtr, SnpInvDtr, SnpNITCDtr, SnpNITCCIDtr, SnpNITCMIDtr	When UP[1:0] = 0b11, this field contains the AIUID of the AIU that is thereby given a <i>Unique Permission</i> to send a DTRreq to the requesting AIU. See Section 4.6.5.4.4.
SnpDVM	If a DVM snoop is issued in multiple portions, each portion is uniquely numbered in order. MPF3 carries the ordinal identifier for the portion of a given DVMOp snoop. 0: Portion 1; 1: Portion 2; etc.
All other:	Don't care.

4.6.5.9.1 AIU-ID

Unique Ncore unit identifier assigned to an AIU. See Section 1.3.4 for details.

4.6.5.10 Size

See Section 4.5.3.6.

4.6.5.11 IntfSize

See Section 4.5.3.7.

This field refers to the native interface of the AIU which issued the CMDreq that is the of this SNPreq. It is copied from the CMDreq.

4.6.5.12 Did (Destination ID)

See Section 4.5.3.8.

This field is copied from the Did field in the corresponding CMDreq.

4.6.5.13 TOF

See Section 4.5.3.9.

4.6.5.14 QoS

See Section 4.5.3.10

4.6.5.15 RBID

See Section 4.12.4.1.

4.6.5.16 AUX

See Section 4.5.3.11.

4.6.5.17 MProt

See Section 4.5.3.12.

4.6.6 SNPrsp Messages

After processing the *snoop transactions*, the agents' caches or proxy caches issue snoop responses, of which data transfers to requesting AIU or DMI, if necessary, may be a part.

Each *snooped* AIU summarizes these responses in a message, called **SNPrsp**, back to DCE.

A snoop may engender issuance of DTRreq and/or DTWreq messages from the snooper AIU. A SNPrsp message is issued by the snooping AIU only after all these messages have been completed at their targets and the corresponding response messages have been received by the AIU.

The DCE gets one SNPrsp message from each snooper AIU to which the SNPreq is sent. The DCE processes the SNPrsps to update the directory to reflect the states of the cacheline at the various AIUs as well as to take subsequent actions, including sending additional messages to additional Ncore 3 units, as might be necessary.

Table 4-34: CCMP SNPrsp message shows the fields in an SNPrsp message.

Table 4-34: CCMP SNPrsp message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:					
TargetId	wTargetId	6-12	O/I	Target Identifier	SNPreq.InitiatorId
Initiator	wInitiatorId	6-12	O/I	Initiator Identifier	Local: FUnitId
CMTType	wCMTType	8	O/I	Type of Message	Local. 0xF2.
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTType, and MessageId.	Local
CMHE:				CCMP Message Header Extension	
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:				CCMP Message Body	
RMessagId	wMsgId	6-10	O/I	Reference Message Id	SNPreq.MessageId
CMStatus	wCmStatus	8	O/I	Output and Input for SNPrsp. Used to deliver snoop response and error report to DCE.	Local
MPF1	wMPF1	6-10	O/I	Multi-purpose Field 1	Local
IntfSize	wIntfSize	1	O/I	Interface Size	Local
QoS	wQoS	4	O/I	Quality of Service.	SNPreq.QoS
AUX	wReqAux	0-8	O/I	Auxiliary field	Local
MPROT	wRspProt	0-10	O/I	Protection bits for Snoop Response Message	Local

4.6.7 SNPrsp field descriptions

4.6.7.1 RMessageld

The Messageld of the SNPreq to which this message is the snoop response. It is copied from the Messageld field in the SNPreq's header.

4.6.7.2 CMStatus

See Section 4.5.3.1.

Section 4.5.3.1 indicates the definition of the field for SNPrsp to be following:

- CMStatus[7:6] = 0b00 indicates a successful snoop.
- CMStatus[5:0] = {RV | RS | DC | DT[1:0] | Snarf} is the actual response issued by the snooper AIU.

Table 4-35 below shows the definitions of the bits CMStatus[5:0]

Table 4-35: Definition of CM Status[5:0] field for SNPrsp messages

Bit	Mnemonic	Definition
5	RV	RV = 1: AIU is host to a <i>valid</i> copy of the cacheline. RV = 0: AIU did not, or, as a result of this snoop, no longer hosts a valid copy of the cache line (i.e., it is in / state).
4	RS	This bit is <i>don't care</i> if RV == 0 and must be set to 0. Otherwise, if RV == 1: RS = 1: The copy hosted by the AIU is in <i>Shared</i> state (SC state). RS = 0: The copy hosted by the AIU is in an <i>Owner</i> state.
3	DC	This bit is <i>don't care</i> if DT[1] == 0 and must be set to 0. Otherwise, if DT[1] == 1: DC = 1: The snooper AIU is delivering a cacheline to the requesting AIU in an <i>Owner</i> state. The line is being transferred directly or via the DMI. DC = 0: The snooper AIU is transferring a cacheline to the requesting AIU in <i>Shared</i> state. At most one SNPrsp for a given SNPreq may have DC set to 1. If DC == 1 in this SNPrsp, it must have DT[1] set to 1.
2	DT[1]	DT[1] = 1: The snooper AIU has sent, or has arranged to send via the DMI, data response to the requesting AIU. The data response transfer uses a DTRreq message. DT[1] = 0: The snooper AIU neither has sent directly, nor has arranged to send via the DMI, a data response to the requesting AIU.
1	DT[0]	DT[0] = 1: The snooper AIU is sending a data transfer to the DMI as part of the snoop response. This transfer to DMI may be a DTWMrqMRDreq which is part of an indirect delivery of a DTRreq to the requester, or it may be a DTWreq to clean data to memory. DT[0] = 0: The snooper AIU is NOT sending a data transfer to the DMI as part of the snoop response. At most one SNPrsp for a given SNPreq may have DT[0] set to 1. If DT[0] == 1 in this SNPrsp, no other SNPrsp for the same SNPreq may have DT[1] set to 1.
0	Snarf	Snarf = 1: The snooper agent has accepted the invitation to receive the relevant cacheline data conveyed by this stashing SNPreq. Snarf = 0: The snooper agent has declined the invitation to receive the relevant cacheline data conveyed by this stashing SNPreq. For SNPrsp for a non-stashing SNPreq, this bit is <i>don't care</i> , but must be set to 0.

Architecture Note

RV, RS:

- $\{RV \mid RS\} == \{0 \mid 1\}$ is an illegal combination. $\{0 \mid 0\}$ indicates that the snooper did not or no more holds a copy of the cacheline. $\{1 \mid 0\}$ implies that an *owner* state has been retained by the snooper. $\{1 \mid 1\}$ implies that the *shared (SC)* state has been retained by the snooper
-

DT[1:0]:

- **00:** Snooper is not transferring data
 - **01:** Snooper AIU has transferred a full or partial cacheline worth of data to DMI. This code can occur in response to snoops for CMOs or WriteUniquePtl, for example.
 - **10:** Snooper AIU has directly transferred a full cacheline to the requester. This occurs during snoops for Reads.
 - **11:** Snooper AIU has arranged to transfer a full cacheline to the requester via the DMI. The snooper AIU does this when the snooping agent supplies a partial cacheline worth of intervention data that must be merged with the cacheline in memory before sending it to the requester. The transfer to DMI is in the form of a DTWMrgMRDreq.
-

DC:

- DC is utilized only in responses to snoops for Read requests. It is set if the data is delivered to the requester in an owner state.
 - If DC == 1 in a snoop response, DT[1] must also be set to 1.
 - DC == 1 implies that the snooper has given up its owner state as a result of the snoop and transferred to the requester. The $\{RV \mid RS\}$ in this snoop response can be $\{0 \mid 0\}$ or $\{1 \mid 1\}$ but not $\{1 \mid 0\}$, since there can be at most one owner of the cacheline in the system.
 - DC == 0 and DT[1:0] == 11 in a Read snoop response indicates that an SC copy of the cacheline has been sent to the requester and the dirty cacheline has been cleaned to memory as part of the snoop process.
 - DC == 1 and DT[1:0] == 11 can indicate:
 - The snooper AIU is using a DTWMrgMRD to deliver data to the requester, or
 - A UC state has been delivered to the requester and the dirty data supplied by the snooper is being cleaned to memory via a DTW. This can happen for a ReadClean snoop.
-

Snarf:

- This bit is relevant only for stashing snoops. When set, it indicates that the snooper will accept the stashing data.
-

4.6.7.3 Mapping of CHI snoop responses to CMStatus[5:0]

Table 4-36 below shows translation of each possible snoop response that may be received over the CHI interface into CMStatus[5:0] field.

Generation of CMStatus is a function of both the SNPreq message and the CHI Snoop Response type received for it. Consequently, a given CHI Snoop Response type can result in more than one CMStatus[5:0] value, as shown in Table 4-36.

Table 4-36: Snoop response translation from CHI to CMStatus[5:0]

CHI				CHI Snoop Response Mnemonic	CMStatus [5:0]					
SRSP		WDAT		RV	RS	DC	DT [1]	DT [0]	Snarf	
Opcode [3:0]	Resp [2:0]	Opcode [2:0]	Resp [2:0]							
0x1	0b000	-	-	SnpResp_I	0	0	0	0	0	
0x1	0b000	-	-	SnpResp_I_Read	0	0	0	0	0	
0x1	0b001	-	-	SnpResp_SC	1	1	0	0	0	
0x1	0b001	-	-	SnpResp_SC_Read	1	1	0	0	1	
0x1	0b011	-	-	SnpResp_SD	1	0	0	0	0	
0x1	0b011	-	-	SnpResp_SD_Read	1	0	0	0	1	
0x1	0b010	-	-	SnpResp_UC	1	0	0	0	0	
0x1	0b010	-	-	SnpResp_UC_Read	1	0	0	0	1	
0x1	010-??	-	-	SnpResp_UD	1	0	0	0	0	
-	-	0x1	0b000	SnpRespData_I	0	0	1	1	0	
-	-				0	0	0	0	0	
-	-	0x1	0b001	SnpRespData_SC	1	1	0	1	0	
-	-	0x1	0b011	SnpRespData_SD	1	0	0	1	0	
-	-	0x1	0b010	SnpRespData_UC	1	0	0	1	0	
-	-	0x1	0b010	SnpRespData_UD	1	0	0	1	0	
-	-	0x5	0b110	SnpRespDataPtl_UD	1	0	0	1	1	
-	-	0x1	0b100	SnpRespData_I_PD	0	0	0	1	1	
-	-				0	0	1	1	0	
-	-				0	0	1	1	0	
-	-				0	0	0	0	1	
-	-				0	0	0	0	1	
-	-	0x1	0b101	SnpRespData_SC_PD	1	1	0	0	1	
-	-				1	1	0	1	0	
-	-				1	1	1	1	0	
-	-	0x1	0b110	SnpRespData_UC_PD	1	0	1	0	1	
-	-	0x5	0b100	SnpRespDataPtl_I_PD	0	0	0	1	1	
-	-				0	0	1	1	0	
-	-				0	0	0	0	1	
-	-				0	0	0	0	1	
-	-				0	0	0	0	1	

In Table 4-36:

- CHI responses ending in _Read correspond to DataPull signal being asserted in the CHI-B Snoop Response transactions in response to *Stashing* snoops. The *Snarf* bit in CMStatus[5:0] is asserted whenever DataPull is asserted.

CHI-E and ACE do not support Stashing operations or the DataPull signal.

- DT[1] set to 1 indicates that the snooper AIU has also generated a DTRreq message to the requesting AIU.
- DT[0] set to 1 indicates that the snooper AIU has also generated a DTWreq message to the DMI.

4.6.7.4 MPF1

In SNPrsp message, this field is used to carry the Message Id to be used by a DTRreq to be sent by the stashing CMDreq requester to the target AIU of the WriteStashFull/Ptl CMDreq. The Message Id is valid only when CMStatus[0] = Snarf = 1 in the CMStatus field included with the SNPrsp message.

When Snarf = 1, the Message Id is forwarded by the DCE to the requester of the WriteStashFull/Ptl CMDreq via STRReq.

4.6.7.5 IntfSize

See Section 4.5.3.7 for general information about the field. This particular field is utilized for stashing operations.

This field specifies the width of the data bus of the native interface at the snooper AIU if it issues this SNPrsp with Snarf == 1. It is sent in anticipation of receiving a DTRreq message with data to inform the potential initiator of the DTRreq.

If Snarf == 1, this IntfSize value is forwarded to the initiator of the related CMDreq via an STRreq message.

4.6.7.6 QoS

See Section 4.5.3.10

4.6.7.7 AUX

See Section 4.5.3.11.

4.6.7.8 MProt

See Section 4.5.3.12.

4.7 DTRreq and DTRrsp Messages

These messages are delivering data to AIUs and thence to the native agents via native data transactions, or to proxy caches. They are used in processing Read and Stashing operations.

4.7.1 DTRreq messages

DTRreq messages are used to respond to Read requests made by AIUs with data. DTRreqs carry ***data payload*** in addition to ***non-data information***. They are sent by DMIs or DIs, which are Ncore interfaces to system storage, or even by other AIUs.

The DTRreq encodings (and the corresponding mnemonics) indicate the coherency state the data is expected to be installed in, in the requesting agent's cache.

Table 4-37 below lists the DTRreq messages defined in the CCMP.

Table 4-37: DTRreq Messages and their codes

DTRreq Message (Mnemonic)	Message Code[7:0]	Operation Description
Data Reply Invalid (DtrDataInv)	0x80	Data Reply with cacheline state of Invalid. Size of delivered data: <i>Atomic Size</i> . Cacheline state expected in requesting agent: I.
Data Reply Shared Clean (DtrDataSCln)	0x81	Data Reply with cacheline state of Shared Clean. Size of delivered data: Cacheline. Cacheline state expected in requesting agent: SC.
Data Reply Shared Dirty (DtrDataSDty)	0x82	Data Reply with cacheline state of Shared Dirty. Size of delivered data: Cacheline. Cacheline state expected in requesting agent: SD.
Data Reply Unique Clean (DtrDataUCln)	0x83	Data Reply with cacheline state of Unique Clean. Size of delivered data: Cacheline. Cacheline state expected in requesting agent: UC.
Data Reply Unique Dirty (DtrDataUDty)	0x84	Data Reply with cacheline state of Unique Dirty. Size of delivered data: Cacheline. Cacheline state expected in requesting agent: UD.

4.7.2 Possible DTR requests per Read CMDreqs

Table 4-38 below shows DTRreqs that might be received for the various CMDreqs. Note that multiple types DTRreqs can be received for a given CMDreq and vice versa.

Table 4-38: CMDreqs and the possible DTRreqs they may receive

CMDreq Mnemonics	Possible DTRreqs Received
CmdRdCln	DtrDataScln, DtrDataUcln
CmdRdVld	DtrDataScln, DtrDataSDty, DtrDataUcln, DtrDataUDty
CmdRdUnq	DtrDataUcln, DtrDataUDty
CmdRdAtm, CmdSwapAtm, CmdCompAtm	DtrDataInv
ReadNITC	DtrDataInv
CmdRdNShDty	DtrDataScln, DtrDataUcln, DtrDataUDty
CmdWrStshFull (Snarf = 1)	DtrDataUDty
CmdWrStshPtl (Snarf = 1)	DtrDataUcln, DtrDataUDty
CmdLdCchShd (Snarf = 1)	DtrDataScln
CmdLdCchUnq (Snarf = 1)	DtrDataUcln
ReadNITCCI	DtrDataInv
ReadNITCMI	DtrDataInv
ReadNC	DtrDataInv

4.7.3 Mapping of DTReq to Native transactions

Table 4-39 below shows the mapping of DTReq messages to CHI-E/B and ACE native protocol data response transaction codes.

Table 4-39: Mapping of DTReq to CHI-E/B and ACE data transaction codes

CCMP DTReq Type	Native Data Transaction Mappings			
	CHI-E/B			ACE
Mnemonic	Mnemonic	RDAT Opcode[2:0]	RDAT Resp[2:0]	RRESP[3:0] ^a
DtrDataInv	CompData_I	0b100	0b000	0b00ab
DtrDataSCln	CompData_SC	0b100	0b001	0b10ab
DtrDataSDty	CompData_SD_PD	0b100	0b111	0b11ab
DtrDataUCln	CompData_UC	0b100	0b010	0b00ab
DtrDataUDty	CompData_UD_PD	0b100	0b110	0b01ab

Note:

a. RRESP-Field:

- RRESP[3] stands for IsShared
- RRESP[2] stands for PassDirty
- “ab” = RRESP[1:0]

RRESP [1]	RRESP [0]	
0	0	OKAY
0	1	EXOKAY
1	0	SLVERR
1	1	DECERR

4.7.4 DTRreq message fields

[Table 4-40](#) shows the fields in an DTRreq message.

Table 4-40: CCMP DTRreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:				CCMP Message Header	
TargetId	wTargetId	6-12	O/I	Target Identifier	If from AIU: { If Intervention: SNPreq.MPF1 If Stashing: FUnitId(STRreq.MPF1.StashNId) } If from DMI: { If Coherent: MRDreq.MPF1 or If non-coherent CMDreq.InitiatorId } If from DII: CMDreq.InitiatorId
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier	Local: FUnitId
CMTYPE	wCMTYPE	8	O/I	Type of Message	Local See Table 4- 37.
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId,CMTYPE, and MessageId.	Local
CMHE:				CCMP Message Header Extension	
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.

Table 4-40: CCMP DTRreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMB:		CCMP Message Body			
Non-data payload:					
RMes-sageld	wMsgId	6-12	O/I	Reference Message Id	If from AIU: { If Secondary: SNPreq.MPF2 If Stashing (Primary): STRreq.MPF2 } If from DMI: { If Coherent: MRDreq.MPF2 or If non-coherent CMDreq.MessageId } If from DII: CMDreq.MessageId
CMStatus	wCmStatus	8	O/I	Output / Input for DTRreq.	Local
RL	wRL	2	O/I	Response Level[1:0]	Local
EO	wEO	1	O/I	Exclusive Okay	If Intervention DTR from AIU {SNPreq.EO} If DTR from DMI in response to DtwMrgMRD {RB.EO} If DTR from DMI in response to MRD {MRD.EO}
TM	wTM	1	O/I	Trace Me	If from AIU: SNPreq.TM If from DMI: { If Coherent: MRDreq.TM or If non-coherent CMDreq.TM} If from DII: CMDreq.TM
MPF1	wMpf1	6-12	O/I	Multi-purpose field #1	Local
QoS	wQoS	4	O/I	Quality of Service.	SNPreq.QoS, DTWreq.QoS, MRDreq.QoS, or RB.QoS
AUX	wReqAux	0-8	O/I	Auxiliary field. (Optional.) Can be used to carry USER bits.	Local
MPROT	wReqProt	0-10	O/I	Protection bits for DTRreq Message Non-data payload	Local
Data Payload - Per Beat Contents:					
Data	nDwsPerBeat * 64	64 128 256 512	O/I	Data payload. 1, 2, 4, or 8 Doublewords (DW) per beat (DwsPerBeat). Maximum data transferred by a DTRreq = 1 cacheline.	Local (SMC or other storage) or NI
BE	nDwsPerBeat * 8	8 16 32 64	O/I	Per DW: 8 Byte Enables (1 per byte of data). BE = 1: Byte is valid BE = 0: Byte is invalid	Local or NI
Data Protection	nDwsPerBeat * 8	8 16 32 64	O/I	Per DW: Parity or ECC (Optional)	Local
DWId	nDwsPerBeat * 3	3 6 12 24	O/I	DW Identifier.	Local

Table 4-40: CCMP DTRreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
DBad	nDwsPerBeat	1 2 4 8	O/I	Bad data indicator. (Same as "Poison.")	Local (SMC or other storage) or NI
AUX	wAUXPerDW * nDwsPerBeat		O/I	Auxiliary data. (Optional.) Can be used to carry USER bits.	Local (SMC or other storage) or NI

4.7.5 DTRreq field descriptions

4.7.5.1 Non-Data Payload

This portion of the message contains non-data information.

4.7.5.1.1 RMessageld

The Messageld of the Read CMDreq to which this message is the data response. It is copied from the Messageld field in the CMDreq's header.

Note that {<CMH> | RMessageld} is repeated per each Data Payload beat.

4.7.5.1.2 CMStatus

See Section 4.5.3.1.

4.7.5.1.3 Operation attributes

Attributes are associated with snoops as well.

Operation attributes are carried by the DTRreq message in the following fields: RL and TM. For all of the above attributes except RL, see Section 4.5.3.3.

4.7.5.1.3.1 RL[1:0]

The RL[1:0] field in DTRreq message may be set to 0b01, which merely represents an acknowledgment of an receipt of the DTRreq message. The CMStatus[7:0] field indicates a successful or erroneous receipt.

The RL[1:0] field in DTRreq message may be set to 0b10, which represents an protocol-level acknowledgment of the receipt of the data by the native agent. The CMStatus[7:0] field indicates a successful or erroneous receipt.

4.7.5.1.3.2 EO

Exclusive Okay.

This flag is forwarded to the requesting AIU along with DTRreq issued to it. The AIU forwards the Exclusive Okay signal to the requesting agent along with data.

Depending on the cause of the DTR, this bit is derived from different sources as documented in Table 4-40. Also see Section 4.6.5.6.

4.7.5.1.3.3 TM

See Section 4.5.3.3.7.

4.7.5.1.4 Multi-Purpose Field-1 (MPF1)

This field is intended for future use for Long non-coherent Reads.

For non-coherent long reads, it identifies the starting *doubleword* for multi-DTR response. All values are issued right aligned within the MPF1 field.

4.7.5.2 QoS

See Section 4.5.3.10

4.7.5.3 Data Payload

This portion of the message contains the data being delivered to the Read requester. Data is delivered in one or more **beats**.

Each beat of delivery contains a *beat* of actual data plus companying auxiliary information applicable to that beat of data.

Each beat of transfer has the same formatting, which is described below. At most a cacheline of data is transferred by a DTRreq.

4.7.5.3.1 Data

This is the **actual data content** of the message.

Each beat of data is comprised of 1, 2, 4, or 8 ordered *doublewords* of data, starting from the lowest addressed byte to the highest addressed byte. Lowest addressed byte is right aligned.

For formatting of data in the Concerto-C domain see Section 8.4.1.

4.7.5.3.2 BE

Byte Enables. One bit is allocated to each byte of data being transferred in the data beat. When a byte enable is asserted, the corresponding byte is validated.

Non-valid bytes must NOT be consumed in computations. BEs must be carried unmodified end-to-end.

Architecture Note

The assertion of BEs for DTRreq messages is implementation specific.

4.7.5.3.3 Data Protection

Data protection field contains bits on a per doubleword basis and for all the doublewords in the beat. Typical applications might use parity or ECC.

4.7.5.3.4 DWId

This field contains ordered list of identities of doublewords carried by the beat.

4.7.5.3.5 DBad

One bit is dedicated per doubleword being transmitted in the beat.

When set, it indicates **Bad** or **Poisoned** doubleword. DBad signal must be carried end-to-end.

A bad doubleword must NOT be consumed in a computation. If data is stored at an interim point, the accompanying DBad signal must be preserved.

4.7.5.3.6 AUX

Additional auxiliary information can be associated on a per doubleword basis. Needed information can be distributed on a per doubleword basis at the source and recovered at the destination.

4.7.6 DTRrsp Messages

After receiving and/or processing the DTRreq, the AIU issues a DTRrsp response back to sending Ncore unit of the DTRreq.

[Table 4-41](#) shows the fields in an DTRrsp message.

Table 4-41: CCMP DTRrsp message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:					CCMP Message Header
TargetId	wTargetId	6-12	O/I	Target Identifier	DTRreq.InitiatorId
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier	Local: FUnitId
CMTYPE	wCMTYPE	8	O/I	Type of Message	Local. 0xF4.
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtect-	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTYPE, and MessageId.	Local
CMHE:					CCMP Message Header Extension
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:					CCMP Message Body
RMessageld	wMsgId	6-10	O/I	Reference Message Id	DTRreq.MessageId
CMStatus	wCmStatus	8	O/I	Output and Input for DTRrsp. Used to deliver snoop response and error report to DCE.	Local
AUX	wReqAux	0-8	O/I	Auxiliary field	Local
MPROT	wRspProt	0-10	O/I	Protection bits for DTR Response Message	Local

4.7.7 DTRrsp field descriptions

4.7.7.1 RMessageld

This refers to the DTRreq to which this message is the response. It is copied from the Messageld field in the DTRreq's header.

4.7.7.2 CMStatus

See Section 4.5.3.1.

4.7.7.3 AUX

See Section 4.5.3.11.

4.7.7.4 MPROT

See Section 4.5.3.12.

4.8 DTWreq and DTWrsp Messages

These messages are associated with delivering data to the system storage for Write or cache management operations.

4.8.1 DTWreq messages

DTWreq messages are used to deliver data to storage in response to Write and cache management CMD requests made by AIUs. DTWreqs carry ***data payload*** in addition to ***non-data information***. They are sent to DMIs, DIs, or the DVE.

Primary purpose of DTWs is to update system storage. However, in the case of DTWs aimed at DMI, the DTWs may result in depositing data into the System Memory Cache (SMC), instead. In the latter case, the SMC maintains the Dirty state for the cacheline if it contains newer data than system memory.

All DTWs are performed *atomically* by their targets.

There are two categories of DTWs:

- Those that are purely Writes to system memory locations. (DTWs)
- Those that are Writes to system memory but in addition, also cause data to be delivered to a Read requester. (DTWMrgMRD)

The two categories are described below.

4.8.1.1 DTW Types

The DTWreq encodings (and the corresponding mnemonics) indicate the coherency state the data is being delivered.

DMIs and DII must process DTWreq targeting a given address in strict order of their arrival.

Table 4-42 below lists the DTWreq messages defined in the CCMP.

Table 4-42: DTWreq Messages and their codes

DTWreq Message (Mnemonic)	Message Code[7:0]	Operation Description
Data Write Null Data (DtwNullData)	0x90	<p>Data Write with no valid data. This DTWreq is sent to DMI or DII. This message is used in case of a WriteBack operation issued by an agent, whose cacheline is lost because of an intervening snoop, which results in the agent issuing a CopyBackWrData_L transaction. This message is also used when the agent issues a WriteDataCancel transaction to cancel a WriteUniquePtl, WriteUniquePtlStash, or WriteNoSnpPtl operation.</p> <p>Note: DtwNullData can be used for WriteUniquePtlStash only when the Snarf bit is negated in CMStatus[] field in the STRreq message. See also DtwMrgMRDUDty in Table 4-43.</p> <p>Size of delivered data: Number of bytes delivered in the data payload is equal to the Size field in the original CMDreq. The data bytes must all be set to zero. No BEs accompanying the data are asserted. Therefore, the data bytes delivered must not modify the bytes of the cacheline in SMC, system memory, or peripheral storage.</p> <p>DTRreq Issued: None</p>
Data Write with Clean (DtwDataCln)	0x91	<p>Data Write with cacheline state of Clean. This DTWreq is sent to DMI or DII. It occurs in the case of a WriteEvict operation issued by an agent. The cacheline delivered is in clean state.</p> <p>Size of delivered data: Cacheline.</p> <p>DTRreq Issued: None</p>
Data Write Partial Dirty (DtwDataPtlDty)	0x92	<p>Data Write with partial cacheline with Dirty state. This DTWreq is sent to DMI or DII. It occurs in the case of a WriteBackPtl, WriteCleanPtl, WriteUniquePtl or WriteNonCoherentPtl operation issued by an agent. The partial cacheline is in Dirty state.</p> <p>Size of delivered data: Atomic Size.</p> <p>DTRreq Issued: None</p>
Data Write Full Dirty (DtwDataFullDty)	0x93	<p>Data Write with full cacheline with Dirty state. This DTWreq is sent to DMI or DII. It occurs in the case of a WriteBackFull, WriteCleanFull, WriteUniqueFull or WriteNonCoherentFull operation issued by an agent. The cacheline is in Dirty state.</p> <p>Size of delivered data: Cacheline.</p> <p>DTRreq Issued: None</p>

4.8.1.2 DTWMrgMRD Types

The DTWMrgMRDreq encodings (and the corresponding mnemonics) indicate the coherency state the data is to be delivered to the requester.

DMIs must process DTWs and DTWMrgMRDreq targeting a given address in strict order of their arrival.

A DTWMrgMRD message processing involves the following steps:

1. Reading the cacheline
2. Merging the DTWMrgMRD data with the cacheline data. Only bytes for which Byte Enables are asserted are used to replace those in the cacheline read in step 1.
3. Sending the result to a Read requester,
4. Updating the memory (or SMC's copy thereof).

Each DTWMrgMRD is performed *atomically* by its DMI target.

DTWMrgMRDs carry *Dirty* data. Therefore, after data from a DTWMrgMRD message is merged with cacheline data, the state of the cacheline becomes *Dirty*. If such a cacheline is not written to memory and instead kept in the SMC, it must be marked there as being *Dirty*. for DTWMrgMRDs other than DTWMrgMRD-UDty.

Architecture Note

It is recommended that after a DTWMrgMRD-UDty is performed, the cacheline is not held in the SMC

Table 4-43 below lists the DTWMrgMRDreq messages defined in the CCMP.

Table 4-43: DTWMrgMRDreq Messages and their codes

DTWreq Message (Mnemonic)	Message Code[7:0]	Operation Description
Data Write Merge with MRDreq in Invalid state (DtwMrgMRDInv)	0x98	<p>Data Write Partial, Merge with cacheline, and issue the data in Invalid state. This DTWreq type is only issued to a DMI. This occurs due to Partial Intervention from a UDP state in the agent in response to a SnpNITCDtr, SnpNITCCIDtr, or SnpNITCMIDtr snoop. The DMI is expected to merge the partial cacheline sent in the message to the copy in system memory (or from SMC) and then issue a DTR message to the requester or target AIU to install the cacheline in Invalid state.</p> <p>Size of delivered data: Atomic Size.^a</p> <p>DTRreq evoked from DMI: DtrDataInv.</p>
Data Write Merge with MRDreq in Shared Clean state (DtwMrgMRDSCIn)	0x99	Reserved.
Data Write Merge with MRDreq in Shared Dirty state (DtwMrgMRDSDty)	0x9A	Reserved.

Table 4-43: DTWMrgMRDreq Messages and their codes

DTWreq Message (Mnemonic)	Message Code[7:0]	Operation Description
Data Write Merge with MRDreq in Unique Clean state (DtwMrgMRDUCIn)	0x9B	<p>Data Write partial, Merge with cacheline, and issue the data in Unique Clean state.</p> <p>This DTWreq type is only issued to a DMI.</p> <p>This occurs due to Partial Intervention from a UDP state in the agent in response to a Read snoop.</p> <p>The DMI is expected to merge the partial cacheline sent in the message to the copy in system memory (or from SMC) and then issue a DTR message to the requester or target AIU to install the cacheline in Unique Clean state.</p> <p>Size of delivered data: Atomic Size.^a</p> <p>DTRreq evoked from DMI: DtrDataUCIn.</p>
Data Write Merge with MRDreq in Unique Dirty state (DtwMrgMRDUDty)	0x9C	<p>Data Write partial, Merge with cacheline, and issue the data in Unique Dirty state.</p> <p>This DTWreq type is only issued to a DMI.</p> <p>This occurs due to Partial Intervention from a UDP state in the agent in response to a Read snoop.</p> <p>The DMI is expected to merge the partial cacheline sent in the message to the copy in system memory (or from SMC) and then issue a DTR message to the requester or target AIU to install the cacheline in Unique Dirty state.</p> <p>Note: DtwMrgMRDUDty is also used by the stashing requester when it receives from a CHI agent a WriteDataCancel data transaction for its WriteUniquePtlStash request. It is employed when the Snarf bit is asserted in CMStatus[] field in the STRreq message. Data payload associated with such a DtwMrgMRDUDty message must have all its bytes set to 0 and Byte Enables negated. The negated Byte Enables mask the merging of the data carried by this message with the cacheline in system memory. Also see DtwNullData in Table 4-42.</p> <p>Size of delivered data: Atomic Size.^a</p> <p>DTRreq evoked from DMI: DtrDataUDty.</p>
Note:		
<ul style="list-style-type: none"> a. When DtwMrgMRDs carry the snoop data delivered by the agent, the size of data transferred is that of a coherency granule, or cacheline, fewer than all bytes of which may be valid as marked via BEs. On the other hand, for WriteUniquePtlStash operations, the requesting AIU may issue a DtwMrgMRD carrying data of <i>atomic size</i> that is equal or smaller than a coherency granule. The source of data within a DtwMrgMRD can be discerned from the <i>Primary</i> field in the message (see Section 4.8.3). 		

4.8.2 Ordering Considerations for DTWs

The order of performance of storage accesses is consequential to the correct behavior of the software executing in computing system. Hence, strict ordering rules must be satisfied by the underlying hardware for the performance of CCMP messages that are used to affect these accesses in the CCMP. In particular, the home units must satisfy the following ordering requirements.

4.8.2.1 Global ordering of accesses and messages

In CCMP, all accesses, coherent or non-coherent, to a given coherency granule are strictly serialized. Coherent accesses are serialized by the home DCE, while non-coherent accesses are serialized by the home DMI or DII.

In CCMP, accesses are represented by messages. Messages referencing a given coherency granule cross the PoS in a **strictly serial manner** – that is, one at a time. This imposes a strict order called the **system order** on the said messages for the said coherency granule.

Once a message passes a PoS, it is said to be **globally ordered**. The effects of a globally ordered message

are **visible** to **all** messages to the same coherency granule issued by agents in the system that are ordered after it. That is, the message is said to be **globally performed**. Note that a message performed globally may not actually be observed by another agent.

The system order imposed on messages at their *PoS* must not be violated by other actions of the system.

The ordering of messages at PoS are marked by issuance of certain other messages in the CCMP.

An STRreq issued by the home unit in response to a Write request message indicates that the Write request has passed the PoS. Thus, the order of issuance of STRreqs imposes a system order of global performance of the corresponding Writes in the system.

A DTWrsp issued by a DMI or DII indicates that the corresponding DTWreq has been globally ordered in the system.

4.8.2.2 Order and performance of non-coherent Writes and their DTWs

In general, DTWs corresponding to non-coherent Writes occur asynchronously and at different times than the Write requests. Arrival of DTWs and their performance in the home unit and home device may similarly be asynchronous and separated in time.

However, DTWs must be performed in the home unit and in the home device in the same system order as the non-coherent Write request messages to which they correspond.

4.8.2.3 Order and performance of coherent requests and their DTWs

The home DCE enforces strict serial ordering on coherent accesses to a given coherency granule. Many of these accesses may involve DTWs being issued to the home DMI.

The CCMP event sequencing ensures that these DTWs are issued and received at the said DMI in the system order imposed by the home DCE on those accesses.

The DMI must perform these DTWs in their strict order of reception.

4.8.2.4 Mutual order and performance of Reads and Writes

If a non-coherent Read request follows a non-coherent Write request in the *system order*, the home unit must ensure the Read performs after the said Write.

An MRDreq and DTWreq to the same coherency granule must be performed in their system order. That is, if an MRDreq follows a DTWreq to the same coherency granule in the *system order*, the home unit must ensure that DTRreq issued in response to the MRDreq, is performed after the said DTWreq. Thus, the DTRreq must return the data deposited by the DTWreq – unless another DTWreq has intervened in the system order for the coherency granule.

Any two DTWs (including DTWMrgMRDs) to the same coherency granule must be performed at the home unit and home device in their *system order*.

4.8.3 DTWreq and DTWMrgMRDreq message fields

Table 4-44 shows the fields in a DTWreq or DTWMrgMRDreq message.

Table 4-44: CCMP DTWreq and DTWMrgMRDreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:				CCMP Message Header	
TargetId	wTargetId	6-12	O/I	Target Identifier	If Secondary: FUnitId(SNReq.DId) If Primary:{ Local: FUnitId(CMDreq.DId) }
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier	Local: FUnitId
CMTypE	wCMTypE	8	O/I	Type of Message	Local. See Table 4-42
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTypE, and MessageId.	Local
CMHE:				CCMP Message Header Extension	
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:				CCMP Message Body	
Non-data payload:					
RBID	wRBID	6-10	O/I	RBID.	If Secondary: SNReq.RBID If Primary: STRreq.RBID
CMStatus	wCmStatus	8	O/I	Output / Input for DTRreq.	Local
RL	wRL	2	O/I	Response Level[1:0]	Local
TM	wTM	1	O/I	Trace Me	If Secondary: SNReq.TM If Primary: CMDreq.TM
Primary	wPrimary	1	O/I	Primary data.	Local

Table 4-44: CCMP DTWreq and DTWMrgMRDreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
MPF1	wMpf1	6-12	O/I	Multi-purpose field #1	If Secondary: { If DTWMrgMRDreq: SNPreq.MPF1 if DTWreq: 0x00 = Don't care } If Primary: { If (Stashing AND STRreq.CMStatus.Snarf == 1); STRreq.MPF1 If (non-Stashing OR (Stashing AND STRreq.CMStatus.Snarf == 0)); 0x00 = Don't care }
MPF2	wMpf2	6-12	O/I	Multi-purpose field #2	If Secondary: { If DTWMrgMRDreq: SNPreq.MPF2 if DTWreq: 0x00 = Don't care } If Primary: { If (Stashing AND STRreq.CMStatus.Snarf == 1); STRreq.MPF2 If (non-Stashing OR (Stashing AND STRreq.CMStatus.Snarf == 0)); CMDreq.MessageId }
IntfSize	wIntfSize	3	O/I	Interface Size	Used for DTWMrgMRDreq. If issued by snooper AIU: SNPreq.IntfSize If issued by WriteStashPtl requester AIU: STRreq.Intf-Size
QoS	wQoS	4	O/I	Quality of Service.	SNPreq.QoS, Native.QoS
AUX	wReqAux	0-8	O/I	Auxiliary field. (Optional.) Can be used to carry USER bits.	Local
MPROT	wReqProt	0-10	O/I	Protection bits for DTWreq Message Non-data payload	Local
Data Payload - Per Beat Contents:					
Data	nDwsPerBeat * 64	64 128 256 512	O/I	Data payload. 1, 2, 4, or 8 Doublewords (DW) per beat (DwsPerBeat). Maximum data transferred by a DTRreq = 1 cacheline.	Local (ProxyCache) or NI
BE	nDwsPerBeat * 8	8 16 32 64	O/I	Per DW: 8 Byte Enables (1 per byte of data). BE = 1: Byte is valid BE = 0: Byte is invalid	Local (ProxyCache) or NI
Data Protection	nDwsPerBeat * 8	8 16 32 64	O/I	Per DW: Parity or ECC (Optional)	Local
DWId	nDwsPerBeat * 3	3 6 12 24	O/I	DW Identifier.	Local

Table 4-44: CCMP DTWreq and DTWMrgMRDreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
DBad	nDwsPerBeat	1 2 4 8	O/I	Bad data indicator (Same as “Poison.”)	Local (ProxyCache) or NI
AUX	wAUXPerDW * nDwsPerBeat		O/I	Auxiliary data. (Optional.) Can be used to carry USER bits.	Local or NI

4.8.4 DTWreq and DTWMrgMRDreq field descriptions

4.8.4.1 Non-Data Payload

This portion of the message contains non-data information.

4.8.4.2 RBID

See Section 4.12.4.1.

Note that {<CMH> | RBID} is repeated per each Data Payload beat.

4.8.4.3 CMStatus

See Section 4.5.3.1.

4.8.4.3.1 Operation attributes

Attributes are associated with snoops as well.

Operation attributes are carried by the DTRreq message in the following fields: RL and TM. For all the above attributes except RL, see Section 4.5.3.3.

4.8.4.3.1.1 RL[1:0]

See Section 4.5.3.3.8.1 for the general semantics of the RL[] field.

The RL[1:0] field in DTWreq message may be set to 0b01, which merely represents a request for an acknowledgment of the reception of the DTWreq message. Such an acknowledgment is sent via a DTWrsp message.

The RL[1:0] field in DTWreq message may be set to 0b10, which represents a protocol-level completion of the request at the immediate receiver of the DTWreq. If such a completion requires acknowledgment of the receipt of the data by the native agent, that is accomplished before sending the DTWrsp message.

When the 0b11 setting is used for DTWMrgMRD requests, the DMI also uses the same setting for the DTRreqs that result from these DTWMrgMRD requests.

4.8.4.3.1.2 Primary

0: Secondary Data. This is data provided by an agent via an intervention data response to a snoop. Since intervention data represents the latest contents of the coherency granule accessed by software, secondary data is always treated as *Dirty*.

1: Primary Data. This is data originating at the agent performing a Write operation. Primary data can be *Clean* or *Dirty*.

Both secondary and primary data may carry one or more beats of data, collectively referred to as **data payload**. The data payload may contain zero to CG-size number of valid data bytes (see Section 4.8.4.5).

Application Note

In CHI and ACE, the payload size of the secondary data (intervention data) is always of the size of a coherency granule, which is 64 bytes

4.8.4.3.2 Multi-Purpose Field-1 (MPF1)

This field is used to carry DTWreq specific information. Table 4-45 shows the contents for various types of DTWreq types. For the ones not mentioned, the field is treated as don't care by the CCMP.

All values are issued right aligned within the MPF1 field, except valid bit (when specified) must be MSB of the field.

Table 4-45: MPF1 semantics for DTWreq

Related request type	Information carried
DTWMrgMRD for: CmdUnqStsh (Snarf = 1)	Stash Target Id Stash Target Id is the unique AIUID of the AIU that hosts the native agent that is the target of this stashing. Implementation note: Implementations may choose to send the FUID of the AIU instead of its AIUID. This Stash Target value is received via the MPF1 field of the STRreq. In case the original WriteUniqueStash request did not have Target identified, DCE may still supply one in the STRreq. See Section 4.11.3.5.
CMD Long Write (Future)	Starting DW for multi-DW DTW.
DTWMrgMRD in response to: SnpNoSDInt, SnpClnDtr, SnpVldDtr, SnpInvDtr, SnpNITCDtr, SnpNITCIDtr, SnpNITCMIDtr	TargetId[] (Section 4.4.2.1) for the DTRreq that will be used to forward the data to the requester AIU.
All other:	Don't care.

4.8.4.3.3 Multi-Purpose Field-2 (MPF2)

This field is used to carry DTWreq specific information. Table 4-46 shows the contents for various types of DTWreq types. For the ones not mentioned, the field is treated as don't care by the CCMP.

All values are issued right aligned within the MPF2 field, except valid bit (when specified) must be MSB of the field.

Table 4-46: MPF2 semantics for DTWreq

Related request type	Information carried
CmdUnqStsh (Snarf = 1)	Stash Target Reference Message Id Stash Target Reference Message Id is the unique RMessageld supplied by the AIU that hosts the native agent that is the target of this stashing. The RMessageld is delivered via SnpRsp message. This Messageld is delivered to the stash Cmd requester via the MPF2 field in the STRreq. See Section 4.11.3.6.
DTWMrgMRD in response to: SnpNoSDInt, SnpClnDtr, SnpVldDtr, SnpInvDtr, SnpNITCDtr, SnpNITCCIDtr, SnpNITCMIDtr	RMessageld[]. (Section 4.5.7.1) for the DTRreq that will be used to forward the data to the requester AIU indicated by MPF1 of this request.
All other:	Don't care.

4.8.4.4 IntfSize

See Section 4.5.3.7 for general information about the field.

This field is used in DTWMrgMRDreq messages. It is not useful for DTWreqs.

If issued by a snooping AIU, the field is copied by from the same field in the SNPreq.

If issued by a stashing requester AIU that issued a WriteStshPtl and received a Snarf = 1 in its STRreq, it is copied from the same field in STRreq.

When DMI receives a DTWMrgMRDreq, it uses this field to optimize the format of data transmitted to requester. (See Chapter 8.)

4.8.4.5 QoS

See Section 4.5.3.10

4.8.4.6 AUX

See Section 4.5.3.11.

4.8.4.7 MPROT

See Section 4.5.3.12.

4.8.4.8 Data Payload

This portion of the message contains the data being delivered to the DTWreq target. Data is delivered in one or more **beats**.

Each beat of delivery contains a *beat* of actual data plus companying auxiliary information applicable to that beat of data.

Each beat of transfer has the same formating, which is described below. At most a cacheline of data is transferred by a DTRreq.

4.8.4.8.1 Data

This is the actual **data content** of the message.

For formatting of data in the Concerto-C domain see Section 8.4.2.

Each beat of data is comprised of 1, 2, 4, or 8 ordered *doublewords* of data, starting from the lowest addressed byte to the highest addressed byte. Lowest addressed byte is right aligned. (See data arrangement.)

4.8.4.8.2 BE

Byte Enables. One bit is allocated to each byte of data being transferred in the data beat. When a byte enable is asserted, the corresponding byte is validated.

Non-valid bytes must NOT be consumed in computations. BEs must be carried unmodified end-to-end.

Architecture Note

For DTWreq messages, BEs must be asserted for all the valid bytes of the data being transmitted.

For DTWNullData request message, none of the BEs are set

4.8.4.8.3 Data Protection

Data protection field contains bits on a per doubleword basis and for all the doublewords in the beat. Typical applications might use parity or ECC.

4.8.4.8.4 DWId

This field contains ordered list of identities of doublewords carried by the beat.

4.8.4.8.5 DBad

One bit is dedicated per doubleword being transmitted in the beat.

When set, it indicates **Bad** or **Poisoned** doubleword. DBad signal must be carried end-to-end.

A bad doubleword must NOT be consumed in a computation. If data is stored at an interim point, the accompanying DBad signal must be preserved.

4.8.4.8.6 AUX

Additional auxiliary information can be associated on a per doubleword basis. Needed information can be distributed on a per doubleword basis at the source and recovered at the destination.

4.8.5 DTWrsp Messages

After receiving and/or processing the DTWreq, the AIU issues a DTRrsp response back to sending Ncore unit of the DTWreq.

Table 4-34 shows the fields in an DTWrsp message.

Table 4-47: CCMP DTWrsp message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:					
TargetId	wTargetId	6-12	O/I	Target Identifier	DTWreq.InitiatorId
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier	Local: FUnitId
CMTType	wCMTType	8	O/I	Type of Message	Local: 0xF3.
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTType, and MessageId.	Local
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:					
RMessagId	wMsgId	6-10	O/I	Reference Message Id	DTWreq.MessageId
CMStatus	wCmStatus	8	O/I	Output and Input for DTRrsp. Used to deliver snoop response and error report to DCE.	Local
AUX	wReqAux	0-8	O/I	Auxiliary field	Local
MPROT	wRspProt	0-10	O/I	Protection bits for DTW Response Message	Local

4.8.6 DTWrsp field descriptions

4.8.6.1 RMessagId

This refers to the DTWreq to which this message is the response. It is copied from the MessageId field in the DTWreq's header.

4.8.6.2 CMStatus

See Section 4.5.3.1.

4.8.6.3 AUX

See Section 4.5.3.11.

4.8.6.4 MPROT

See Section 4.5.3.12.

4.9 MRDreqs and MRDrsp

MRD stands for Memory Read Directive.

4.9.1 MRDreq messages

This type of message is sent by DCE to DMI for one of two reasons:

1. In response to a Read command after the DCE has concluded that the data will not be provided to the requester by a snooper. This determination may be based on the SNPrsp messages received from the snoopers or may be based on the inspection of the states of the cacheline in various potential snoopers as reflected in the directory.
2. In order to enforce a *cache maintenance operation (CMO)* at the SMC. After having enforced the semantics of the CMO among system's agents via snoops, the CMOs must also be enforced at the system memory cache to fully perform the CMO.

An MRDreq message is sent by the DCE to a DMI to have a Read operation performed in system memory in response to a Read operation from a native agent. After retrieving the data from memory, the DMI transmits the data to the requesting AIU via a DTRreq message.

A DMI may optionally embed a system level cache referred to as **Memory Cache** which can cache the data retrieved from memory. The data is held in Memory Cache (MC) in *Clean* or *Dirty* states in relation to the contents of the memory itself.

When an MRDreq is received by the DMI, it inspects the MC before issuing a memory access to see if the requisite cacheline is already installed there. If so, that copy of the cacheline is provided to the requester.

Some MRDreqs also contain directives to the Memory Cache.

Table 4-48 shows the different types of MRD requests. It also shows their message codes.

Table 4-48: MRDreq Messages

MRDreq Message (Mnemonic)	Operation Description	Message Type[7:0]
Memory Read Clean (MrdRdCln)	Perform a system memory Read and transmit the data to the requesting AIU as a <i>Clean copy</i> . (This MRD type is deprecated in CCMP.)	0x60
Memory Read with Shared Clean (MrdRdWSCln)	Perform a system memory Read and transmit the data to the requesting AIU as a <i>Shared Clean copy</i> .	0x61
Memory Read with Unique Clean (MrdRdWUCln)	Perform a system memory Read and transmit the data to the requesting AIU as a <i>Unique Clean copy</i> .	0x62
Memory Read with Unique (MrdRdWU)	Perform a system memory Read and transmit the data to the requesting AIU as a <i>Unique copy</i> . If the cacheline data is retrieved from memory, it is delivered in <i>UC</i> state. If the cacheline data is obtained from MC, then if the copy is in <i>Clean</i> state, it is delivered to the requester in <i>UC</i> state. If the copy in MC is in <i>Dirty</i> state, the copy is delivered to the requester in <i>UD</i> state, and the copy is invalidated in situ.	0x63
Memory Read with Invalid (MrdRdWInv)	Perform a system memory Read and transmit the <i>snapshot</i> of the cacheline in <i>Invalid</i> state to the requesting AIU, which is expected NOT to cache it.	0x64

Table 4-48: MRDreq Messages

MRDreq Message (Mnemonic)	Operation Description	Message Type[7:0]
Memory Prefetch (MrdPref)	Perform a system memory prefetch Read if the cacheline doesn't already exist in the SMC. The data is stored into SMC as a clean copy. The fetched cacheline is NOT transmitted to any AIU.	0x65
Memory Clean (MrdClean)	If the cacheline is in Dirty state in the SMC, it is cleaned to memory. This command is used for <i>CleanShared</i> and <i>CleanSharedPersist</i> type of cache management operations.	0x66
Invalidate SMC copy (MrdInvalidate)	Invalidate the copy of the cacheline in situ if it exists in the SMC. This command is used for <i>MakeInvalid</i> type of cache management operation.	0x67
Memory Flush (MrdFlush)	Flush the cacheline to memory, if its copy exists in the MC. This command is used for <i>CleanInvalid</i> type of cache management operation.	0x68

4.9.2 DTRreqs issued in response to MRDreqs

See Section 5.5.1.

4.9.3 MRDreq message fields

Table 4-49 shows the fields in an MRDreq message.

Table 4-49: CCMP MRDreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:					CCMP Message Header
TargetId	wTargetId	6-12	O/I	Target Identifier	FUnitId(CMDreq.D.Id)
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier	Local: DCE FUnitId
CMTType	wCMTType	8	O/I	Type of Message	Local. See Table 4-48
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTType, and MessageId.	Local
CMHE:					CCMP Message Header Extension
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:					CCMP Message Body
CMStatus	wCmStatus	8	I	Input-only for MRDreq. Used to deliver Transport encountered errors to the Target unit.	Local: RX Demux
Addr	wAddr	32-52	O/I	Address	CMDreq.Addr
VZ	wVZ	1	O/I	Visibility	CMDreq.VZ
AC	wAC	1	O/I	Allocate Hint	CMDreq.AC
NS	wNS	1	O/I	Non-secure Access	CMDreq.NS
PR	wPR	1	O/I	Privileged Access	CMDreq.PR
LK	wLK	2	O/I	Lock[1:0]	CMDreq.LK
RL	wRL	2	O/I	Response Level[1:0]	Local
EO	wEO	1	O/I	Exclusive Okay	Local
EO	wEO	1	O/I	Exclusive Okay	Local
TM	wTM	1	O/I	Trace Me	CMDreq.TM
MPF1	wMpf1	6-12	O/I	Multi-purpose field #1	CMDreq.InitiatorId
MPF2	wMpf2	6-12	O/I	Multi-purpose field #2	CMDreq.MessageId
MPF3	wMpf3	6-12	O/I	Multi-purpose field #3	CMDreq.FlowId
Size	wSize	3	O/I	Access size	CMDreq.Size

Table 4-49: CCMP MRDreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
IntfSize	wIntfSize	3	O/I	Interface Size	CMDreq.IntfSize ^a
QoS	wQoS	4	O/I	QoS Label	CMDreq.QoS
AUX	wReqAux	0-8	O/I	Auxiliary field	CMDreq.AUX
MPROT	wReqProt	0-10	O/I	Protection bits for MRDreq Message	Local

Note:

- a. If the initial command is a stash, DCE must use the stash target interface size from snoop response

4.9.4 MRDreq field descriptions

4.9.4.1 CMStatus

See Section 4.5.3.1.

4.9.4.2 Addr

See Section 4.5.3.2.

Addr field in MRDreq is copied from the CMDreq to which it is related.

4.9.4.3 Operation attributes

Attributes are associated with snoops as well.

Operation attributes are carried by the MRDreq message in the following fields: AC, NS, PR, LK, RL, and TM.

For all the above attributes, see Section 4.5.3.3.

4.9.4.3.1 VZ

CCMP supports cache management operations, namely, CleanShared, CleanSharedPersist, CleanInvalid, and MakeInvalid. Processor architectures specify whether the effects of these operations be visible within the coherency domain or beyond the coherency domain and across the entire system. Depending on the architectural requirement, the corresponding CmdReqs are issued with the VZ bit set to the appropriate value.

The VZ bit in other messages are also set based on the visibility semantic needs for those messages.

MrdClean, MrdFlush, and MrdInvalidate are associated with the above cache management operations. Satisfaction of ARM architecture requires that all these message carry VZ=1.

4.9.4.3.2 RL[1:0]

For all MRD requests except for MrdFlush, MrdCln, MrdInv and MrdRdCln related to stash command the RL[1:0] field in MRDreq message is set to 0b01, which requires a *transport-level* acknowledgment of simply the receipt of the MRDreq message. MrdFlush, MrdCln, and MrdInv use RL[1:0] = 0b10 which indicates a protocol level completion of the MRDreq. MrdRdCln related to stash command use RL[1:0] = 0b11 which indicates that one or more messages to be generated at the message's destination unit.

Specifically, RL[1:0] = 0b10 requires that the MrdFlush and MrdClean operations are *performed* all the way to memory device before MRDrsp is generated. MrdInv requires that the cacheline is invalidated from the SMC before MRDrsp is generated.

Both MrdFlush and MrdClean operations at their completion must guarantee that any modified data has been written to system memory. MrdFlush further guarantees at its completion that no valid cacheline copy remains in the SMC. MrdInv also guarantees at its completion that no valid cacheline copy remains in the SMC.

Architecture Note

In CCMP, an MrdClean message is generated for CleanShared, CleanSharedPersist. An MrdFlush message is generated for CleanInvalid operation. A MakeInvalid message is generated for a MakeInvalid operations.

- These operations are caused by an explicit software intent to *clean*, *flush* or *invalidate* the cacheline content from system caches, typically, so that another device might access the data.
- These operations are parts of software action called *software-initiated cache management* and are often utilized to achieve *software managed coherency*.

An MrdFlush or MrdShared engenders in DMI a Write to memory if the cacheline is found in dirty state in the SMC. If the cacheline is in clean state, it is simply invalidated in situ for MrdFlush. In the case of MrdInv, the cacheline is invalidated in situ regardless of the state it is in, in the SMC.

Implementation Note

To ensure that the Write has completed to the system memory, the Write issued must be of *non-buffered* type (with AWCACHE[3:0] = 0b0010), for which the completion response is issued by the target system memory device.

Response to MrdFlush must ensure any memory Write operation to the address carried by the MrdFlush that might already in progress or queued at the DMI is completed and visible in the memory device. To facilitate this, it may be advantageous to issue all Writes from DMI to memory to be non-buffered.

If Writes are in progress when an MrdFlush is received, depending on the implementation, it may be sufficient to generate an additional 0-byte non-buffered Write to ensure that the previous Writes are pushed to the memory device before response is generated for MrdFlush.

4.9.4.3.3 EO

Exclusive Okay.

The DCE sets this bit in an MRD message if the Exclusive Monitor indicates that the Exclusive Read CMDreq has successfully performed at point of serialization and if the MRD is issued to have the DMI generate a DTRreq to the requesting AIU.

This field in the MRDreq is copied by the DMI into the EO field of the DTRreq.

Also, see Section 4.6.5.6.

4.9.4.3.4 TM

See Section 4.5.3.3.7.

4.9.4.4 Multi-Purpose Field #1 (MPF1)

Used to carry TargetId for the DTRreq message, which carries the Read data response to the requester.

This field is copied from InitiatorId in the CCMP header of the CMDreq this message represents.

4.9.4.5 Multi-Purpose Field #2 (MPF2)

Used to carry RMessageld for the DTRreq message, which carries the Read data response to the requester. This field is copied from Messageld in the CCMP header of the CMDreq this message represents.

4.9.4.6 Multi-Purpose Field #3 (MPF3)

Used to carry Flow Id from the original Cmd message. This may be used for cache allocation policy.

4.9.4.7 Size

See Section 4.5.3.6.

This field is copied from the same field in the related CMDreq.

4.9.4.8 IntfSize

See Section 4.5.3.7 for general information about the field. This field in MRDreq is copied from the same field in the CMDreq.

For transactions where the requester and the destination of the returned data are not the same, e.g. Stashes, DCE must provide the interface size of the stash target.

When DMI receives an MRDreq, it uses this field to determine the correct format of data transmitted to requester. (See Chapter 8.)

When DMI receives a DTWMrgMRDreq from a snoop target, it uses this field to determine the correct format of data transmitted to data target. (See Chapter 8.)

4.9.4.9 QoS

See Section 4.5.3.10

4.9.4.10 AUX

See Section 4.5.3.11.

4.9.4.11 MPROT

See Section 4.5.3.12.

4.9.5 MRDrsp Messages

After receiving the MRDreq message, the DMI issues an MRDrsp message back to the DCE to acknowledge the receipt of the former message. It returns a credit to the DCE to send another MRDreq. Table 4-50 shows the fields in an MRDrsp message.

Table 4-50: CCMP MRDrsp message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:					CCMP Message Header
TargetId	wTargetId	6-12	O/I	Target Identifier	MRDreq.InitiatorId
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier	Local: FUnitId
CMTType	wCMTType	8	O/I	Type of Message	Local. 0xF6.
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTType, and MessageId.	Local
CMHE:					CCMP Message Header Extension
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:					CCMP Message Body
RMessagId	wMsgId	6-10	O/I	Reference Message Id	MRDreq.MessageId
CMStatus	wCmStatus	8	O/I	Output and Input for MRDrsp. Used to deliver response and error report to DCE.	Local
AUX	wRspAux	0-8	O/I	Auxiliary field	Local
MPROT	wRspProt	0-10	O/I	Protection bits for MRD Response Message	Local

4.9.6 MRDrsp field descriptions

4.9.6.1 RMessageld

Message Id of the MRDreq whose reception is being acknowledged.

4.9.6.2 CMStatus[7:0]

Refer to Table 4-4.

4.9.6.3 NDPROT

Refer to Section 4.5.3.12.

4.10 HNTreqs and HNTrsps

HNT stands for *Prefetch Hint*.

4.10.1 HNTreq messages

This message is sent after DCE in response to a Prefetch CMDreq.

An HNTreq message is sent by the DCE to a DMI to have a Read operation performed in system memory in response to a Prefetch operation from a native agent. After retrieving the data from memory, the DMI allocates it in the SMC or in a separate ***prefetch buffer*** if one is implemented. The data is held there in a *Clean* state.

When an HNTreq is received by the DMI, it inspects the SMC to see if the requisite cacheline is already installed there, before issuing the memory access.

The size of the memory access is that of a cacheline.

4.10.2 HNTreq message fields

Table 4-51 shows the fields in an HNTreq message.

Table 4-51: CCMP HNTreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:					CCMP Message Header
TargetId	wTargetId	6-12	O/I	Target Identifier	FUnitId(CMDreq.D Id)
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier	Local: DCE FUnitId
CMTType	wCMTType	8	O/I	Type of Message	Local. 0x78.
Messageld	wMessageld	6-12	O/I	Messageld	Local

Table 4-51: CCMP HNTreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMType, and MessageId.	Local
CMHE:				CCMP Message Header Extension	
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:				CCMP Message Body	
CMStatus	wCmStatus	8	I	Input-only for HNTreq. Used to deliver Transport encountered errors to the Target unit.	Local: RX Demux
Addr	wAddr	32-52	O/I	Address	CMDreq.Addr
AC	wAC	1	O/I	Allocate Hint	CMDreq.AC
NS	wNS	1	O/I	Non-secure Access	CMDreq.NS
RL	wRL	2	O/I	Response Level	Local
AUX	wReqAux	0-8	O/I	Auxiliary field	Local
MPROT	wReqProt	0-10	O/I	Protection bits for HNTreq Message	Local

4.10.3 HNTreq field descriptions

4.10.3.1 CMStatus

See Section 4.5.3.1.

4.10.3.2 Addr

See Section 4.5.3.2.

Addr field in HNTreq is copied from the CMDreq to which it is related.

4.10.3.3 Operation attributes

Attributes are associated with snoops as well.

Operation attributes are carried by the HNTreq message in the following fields: AC, and NS. For all the above attributes, see Section 4.5.3.3.

4.10.3.3.1 RL[1:0]

The RL[1:0] field in HNTreq message is set to 0b01, which merely represents an acknowledgment of an receipt of the HNTreq message. The CMStatus[7:0] field indicates a successful or erroneous receipt.

4.10.3.4 IntfSize

See Section 4.5.3.7 for general information about the field. This field in MRDreq is copied from the same field in the CMDreq.

4.10.3.5 AUX

See Section 4.5.3.11.

4.10.3.6 MProt

See Section 4.5.3.12.

4.10.4 HNTrsp Messages

After receiving the HNTreq message, the DMI issues an HNTrsp message back to the DCE to acknowledge the receipt of the former message. It returns a credit to the DCE to send another HNTreq. Table 4-52 shows the fields in an HNTrsp message.

Table 4-52: CCMP HNTrsp message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:					CCMP Message Header
TargetId	wTargetId	6-12	O/I	Target Identifier	HNTreq.InitiatorId
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier	Local: FUnitId
CMTType	wCMTType	8	O/I	Type of Message	Local. 0xF5.
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTType, and MessageId.	Local
CMHE:					CCMP Message Header Extension
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:					CCMP Message Body
RMessageld	wMsgId	6-10	O/I	Reference Message Id	HNTreq.MessageId
CMStatus	wCmStatus	8	O/I	Output and Input for HNTrsp. Used to deliver response and error report to DCE.	Local
AUX	wRspAux	0-8	O/I	Auxiliary field	Local
MPROT	wRspProt	0-10	O/I	Protection bits for HNT Response Message	Local

4.10.5 HNTrsp field descriptions

4.10.5.1 RMessageld

Message Id of the HNTreq whose reception is being acknowledged.

4.10.5.2 CMStatus[7:0]

Refer to Table 4-4.

4.10.5.3 MPROT

Refer to Section 4.5.3.12.

4.11 STRreqs and STRrsp

STR stands for ***State Reply***.

4.11.1 STRreq messages

An STRreq message is sent by a *home* Ncore unit to an AIU to indicate a key event during the processing of a native operation in the CCMP protocol sequence, which starts with a CMDreq message sent by an AIU to the appropriate *home* unit.

- An STRreq is sent by the DCE for a coherent operation to:
 - Indicate the completion of snooping process for coherent operation.
 - Report the *Summary Coherency Response* (see Section 4.11.3.2).
 - Indicate the achievement of global ordering and visibility for the operation.
 - Guarantee of forward progress for the operation.
 - Indicate success or failure of an Exclusive access operation.
 - Indicate if the target agent has accepted the invitation to receive data for a Stashing operation.
 - Indicate a permission to send *Primary data* associated with a Write operation. The data is transmitted in the form of a DTW.
- STRreq message also supplies additional information to facilitate the transmission of the DTW (see Section 4.11.3).
 - An STRreq is sent by a DMI or DII for a non-coherent operation to:
 - Indicate the achievement of global ordering and visibility for the operation.
 - Guarantee of forward progress for the operation.
 - A permission to send Primary data associated with a Write operation. The data is transmitted in the form of a DTW.

STRreq message also supplies the receiving unit's Interface Size information to facilitate the transmission of the DTW.

Note that DVM Message operation also behaves like a Write operation.

Also see Section 5.4.7.

4.11.2 STRreq message fields

STRreq reports the ***Summary Coherency Response*** in the CMStatus[7:0] field.

Table 4-53 shows the fields in an STRreq message.

Table 4-53: CCMP STRreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:					CCMP Message Header
TargetId	wTargetId	6-12	O/I	Target Identifier	CMDreq.InitiatorId
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier	Local: FUnitId
CMTypE	wCMTypE	8	O/I	Type of Message	Local. 0x7A.
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTypE, and MessageId.	Local
CMHE:					CCMP Message Header Extension
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:					CCMP Message Body
RMessagId	wMsgId	6-10	O/I	Reference to the CMDreq to which this is related.	CMDreq.MessageId
CMStatus	wCmStatus	8	O/I	Output and Input for STRreq.	Local
RL	wRL	2	O/I	Response Level	Local
RBID	wRBID	6-10	O/I	RBID.	Local
MPF1	wMpf1	6-12	O/I	Multi-purpose field #1	If Stash Target identified: { CMDReq.MPF1.StashNId } If Stash Target chosen by DCE: { Local:SNPrsp.InitiatorId if SNPrsp.CMStatus.Snarf == 1 }
MPF2	wMpf2	6-12	O/I	Multi-purpose field #2	SNPrsp.MPF1
IntfSize	wIntfSize	3	O/I	Interface Size	SNPrsp.IntfSize for Snooper with SNPrsp.CMStatus.Snarf == 1
QoS	wQoS	4	O/I	Quality of Service.	CMDreq.QoS
AUX	wReqAux	0-8	O/I	Auxiliary field	Local
MPROT	wReqProt	0-10	O/I	Protection bits for STRreq Message	Local

4.11.3 STRreq field descriptions

4.11.3.1 RMessageld

RMessageld is the Messageld of the CMDreq to which this message is correlated.

4.11.3.2 CMStatus

STRreq reports the ***Summary Coherency Response*** in the CMStatus[7:0] field.

Section 4.5.3.1 indicates the definition of the field for STRreq to be following:

- CMStatus[7:6] = 0b00
- CMStatus[5:0] = {5:1 Reserved | Snarf}

Table 4-54 below shows the definitions of the bits CMStatus[4:0] for STRreq.

Table 4-54: Definition of CMStatus[5:0] field for STRreq messages

Bit	Mnemonic	Definition
5:1	Reserved	Reserved
0	Snarf / ExOkay	<p>For Stashing CMDreqs: Snarf = 1: The target snooper agent has accepted the invitation to receive the stashed data. Snarf = 0: The target snooper agent has declined the invitation to receive the stashed data.</p> <p>For Exclusive CMDreqs: ExOkay = 1: The Exclusive CMDreq was successful ExOkay = 0: The Exclusive CMDreq was NOT successful. For other CMDreqs: Don't care.</p>

4.11.3.3 Operation attributes

Attributes are associated with snoops as well.

Attribute RL[1:0] is the only one carried by the STRreq message. For definition, see Section 4.5.3.3.

4.11.3.3.1 RL[1:0]

The RL[1:0] field in STRreq message is set to 0b11.

The STRrsp message is issued only after STRreq is received, but additional semantics are also implied by the message. The response is expected after the operation is completed at the agent.

For a coherent operation, a corresponding STRrsp message must represent a protocol level completion of the native operation and completion acknowledgment, if any, at the native agent. The STRrsp response, when received, allows the DCE to remove hazard for the address contained in the original native operation.

For a non-coherent operation, a corresponding STRrsp message must represent a protocol level completion of the native operation at the native agent.

4.11.3.4 RBID

See Section 4.12.4.1.

4.11.3.5 Multi-Purpose Field #1 (MPF1)

Relevant only in the case of stashing CMDreqs, *CmdWrStshFull* and *CmdWrStshPtl*, if Snarf = 1.

Used to carry **StashAIUID** in the case of Stashing if Snarf = 1. This is the identifier of the AIU that has accepted the invitation to stash data in its cache. This value may be the same as the original stashing Target if it was specified, or may be value chosen by the DCE, if the original stashing request did not specify the stash Target.

If Snarf = 0, this field is don't care.

4.11.3.6 Multi-Purpose Field #2 (MPF2)

Relevant only in the case of stashing CMDreqs, *CmdWrStshFull* and *CmdWrStshPtl*, if Snarf = 1.

Used to carry RMessageld for the DTRreq message, which carries the **stash data response** to the requester.

This field is copied from the stashing *MessageId* in the *MPF1* field of the SNPrsp message from the target that indicated Snarf = 1.

4.11.3.7 IntfSize

See Section 4.5.3.7 for general information about the field. This field in STRreq carries the IntfSize information about:

- The DMI that will be the target of the DTWreq message in the case of Write CMDs, including WriteStash CMDs for which the Stash target AIU indicated Snarf = 0 in its SNPrsp, or
- The Stash target AIU for WriteStash CMDs which issued Snarf = 1 – that is carried by the eponymous field in the CMStatus[] being carried by the current STRreq. (See Section 4.6.7.5.)

4.11.3.8 QoS

See Section 4.5.3.10

4.11.3.9 AUX

See Section 4.5.3.11.

4.11.3.10 MPROT

See Section 4.5.3.12.

4.11.4 STRrsp Messages

STRrsp message indicates the completion of all the necessary transactions by the issuing AIU at its native agent interface.

In the case of a coherent operation, the reception of a STRrsp signals to the DCE the completion of the operation's coherency semantics at the native agent and permits the DCE to release the *coherency hazard block* against the operation's address, allowing the processing of next coherent operation to the same address to be commenced.

After receiving the STRreq message, the AIU issues an STRrsp message back to the DCE to acknowledge the receipt of the former message. It returns a credit to the DCE to send another STRreq.

Table 4-55 shows the fields in an STRrsp message.

Table 4-55: CCMP STRrsp message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:				CCMP Message Header	
TargetId	wTargetId	6-12	O/I	Target Identifier	STRreq.InitiatorId
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier	Local: FUnitId
CMTYPE	wCMTYPE	8	O/I	Type of Message	Local. 0xF7.
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTYPE, and MessageId.	Local
CMHE:				CCMP Message Header Extension	
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:				CCMP Message Body	
RMessageld	wMsgId	6-10	O/I	Reference Message Id	STRreq.MessageId
CMStatus	wCmStatus	8	O/I	Output and Input for STRrsp. Used to deliver response and error report to DCE.	Local
AUX	wRspAux	0-8	O/I	Auxiliary field	Local
MPROT	wRspProt	0-10	O/I	Protection bits for State Response Message	Local

4.11.5 STRrsp field descriptions

4.11.5.1 RMessageld

Message Id of the STRreq whose reception is being acknowledged.

4.11.5.2 CMStatus[7:0]

Refer to Table 4-4.

4.11.5.3 MPROT

Refer to Section 4.5.3.12.

4.12 RBRreqs and RBRrsp

RBR stands for Request Buffer Reservation. The RBRreq protocol is used to manage data buffer allocation and release.

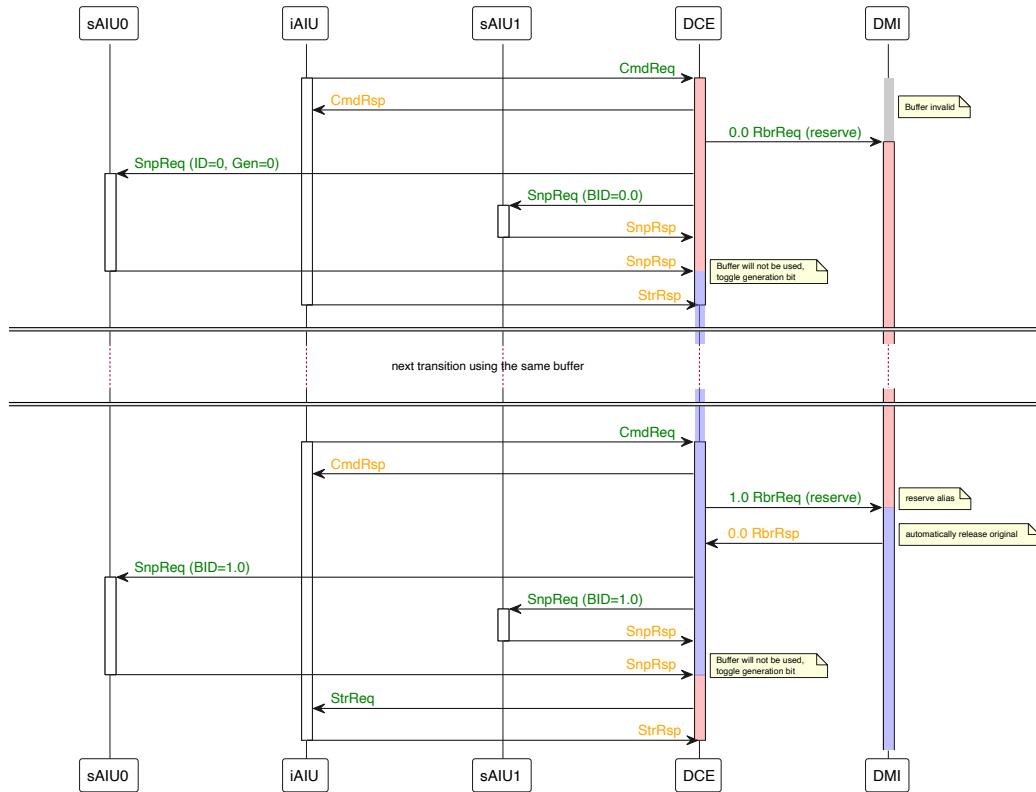


Figure 4-2: Transaction with RBRreq handshake

Architecture Notes

The buffer stays alive until either a new reservation (RBRreq) arrives, or it has been used by data and will be explicitly invalidated/released by an RBRrsp

GenIndex is only allowed to toggle when SnpRsp positively confirms the buffer has not been used!

4.12.1 Request Buffers

The **Request Buffer (RB)** is a hardware data structure used to hold control information from the coherent CMDreq that is needed to process an intervention data transfer to DMI that might result from an SNPreq corresponding to the said CMDreq. They are also utilized to mediate data transfers from requesting AIUs, such as in the case of Write CMDreqs.

Request Buffers are also utilized by DMIs and DIs for mediating data transfers from requesters AIUs to themselves for non-coherent CMDreqs.

An **RBID** identifies a specific Request Buffer.

The fields in the RBRreq message as shown in Table 4-56, except RBID, CMStatus, and MPROT, represent the information held in an RB.

4.12.1.1 Using RBs for data transfers in coherent CMDreqs

4.12.1.1.1 Intervention data transfers

Each DCE is allocated an independent pool of Request Buffers in each DMI it communicates with. The DCE uses these to manage intervention data transfers that might be engendered by the SNPreqs it generates. DCE manages the use of its set of RBIDs on a per DMI basis. An unreserved RB must be available in the right DMI for the DCE to issue an SNPreq that refers to an address that is mapped to that said DMI.

The control information held in the Request Buffer is delivered to it via the RBRreq message, sent to the DMI by the DCE. Address, Size, IntSize, and the various attributes are a part of this information.

The DCE deposits this information using an RBRreq message. In case a data transfer is issued by the snooper AIU to DMI, this identifier is copied into the same named field in that data transfer message DTWreq from SNPreq.

4.12.1.1.2 Using RBs for data transfers from requesters in coherent CMDreqs

For coherent Writes, the DCE also delivers the appropriate RBID to the requester AIU via the STRreq. Upon receiving the STRreq, the requester issues a DTWreq referencing the RBID.

4.12.1.1.3 Coordinating RB Usage

DCE keeps track of the RBs it has been allocated on a per DMI basis and does not issue an RBRreq unless it has an unreserved RB available.

A reserved RB may not get utilized if the snoops for CMDreq do not generate any DTWs. In this case, DCE may **un-reserve** or **release** the RB to be able to reuse it for another CMDreq.

Each physical buffer in DMI has two alias identifiers, called generations. These are distinguished by a single bit, the GenIndex. DCE will track the state of a physical buffer at certain decision points – a buffer may be reassigned under the following conditions:

1. Both generations are available – no transaction using this physical buffer is active. In this case RB[RBID] is considered *invalid*.
2. One of the RB's generations is available, and it has been determined by DCE that the currently assigned generation will not be used – this is known when for the currently outstanding transaction
 - a. all SNPrsp have been received and the consolidated state indicates **no data transfer**
 - b. the consolidated state or the command type required data transfer and RBRrsp has been received

Whenever an RBRrsp has been sent to a DMI while the previous generation of RB is still allocated, i.e. RB is valid, that DMI shall respond with an RBRrsp to *release* the previous generation alias.

Only after the *release* is complete, can the RB be *reused* in snoops for another CMDreq. To ensure no race condition occurs, the RB must be *reused* only after the next decision point, SNPrsp or RBRrsp has passed and corresponding RB of that GenIndex has been marked as *unreserved*.

Architecture Notes

The DCE may choose to issue an RB Reserve request to a DMI only if it determines at least one of the following conditions is true:

1. Snoops must be issued for the CMD and one of those might result in intervention data transfers to the DMI.
2. The CMD involves a data transfer from the requester. For example, WriteCMDs.

Note that condition 1 above depends on the current state of the cacheline in the system currently registered in the DCE's directory.

MakeInvalid and MakeUnique are examples of coherent CMDreqs that, by definition, do not incur data transfers to DMI

Implementations must devise means to ensure that release operations are successful before reusing the RB. This can be achieved by ensuring that the RBRreq to *reserve* and RBRreq to *release* are issued in the correct order, reach the DMI in the intended order and that DMI processes them in the order of arrival

Alternatively, the DCE may simply re-reserve a reserved RB via RBRreq for another CMDreq. But to avoid race conditions, the SNPreqs for that CMDreq must not be issued until the RBRrsp for the re-reservation has been received. Plus, the RBRreq must be issued with RL[1:0] set to 0b10 to ensure protocol-level response (see Section 4.12.4.5.2)

Ncore 3 Requirement

In Ncore 3 RBID value used by the DCE is formed by concatenation of three subfields: DCEID, GenIndex and RBIndex ($\text{RBID} = \text{DCEID} \mid \text{RBIndex}$), where DCEID is the NUID of the DCE for which the RB is reserved.

GenIndex is a single bit and identifies the generation of the current meta data (transaction parameters) used with this buffer.

Each DCE is allocated a block of 2^i Request Buffers. RBIndex is the identity of the buffer within that block.

4.12.1.2 Using RBs for non-coherent CMDreqs

A separate and independent pool of RBs is used by each DMI or DII for the purpose of mediating data transfers corresponding to non-coherent CMDreqs issued to it.

For this purpose, the DMI or DII delivers the appropriate RBID to the requester AIU via the STRreq. Upon receiving the STRreq, the requester issues a DTWreq referencing the RBID.

4.12.1.3 Independence of RBs and Data Buffers

DMIs and DIIs also have data buffers to hold the data delivered to them via DTWreqs. It is expected that number of data buffers is less than that of the RBs, to help lower the area cost. The actual flow of data from AIUs to DMIs and DIIs is controlled by credits associated with DTWreqs.

4.12.2 RBRreq message

This message is sent by DCE to DMI when there is a possibility of intervention in response to an SNPreq and a Request Buffer is available to deposit the relevant information from the CMDreq that might become necessary to process the intervention data transmission in the form of the DTW.

4.12.3 RBRreq message fields

Table 4-56 shows the fields in an RBRreq message.

Table 4-56: CCMP RBRreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:					CCMP Message Header
TargetId	wTargetId	6-12	O/I	Target Identifier	FUnitId(CMDreq.D Id)
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier	Local: DCE FUnitId
CMTType	wCMTType	8	O/I	Type of Message	Local. 0x7C.
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTType, and MessageId.	Local
CMHE:					CCMP Message Header Extension
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:					CCMP Message Body
RBID	wRBID	6-10	O/I	RBID	Local
CMStatus	wCmStatus	8	I	Input-only for RBRreq. Used to deliver Transport encountered errors to the Target unit.	Local: RX Demux
RType ^a	wRType	0	O/I	Request type: 1 = Reserve the RB 0 = Release the RB	Local
Addr	wAddr	32-52	O/I	Address	CMDreq.Addr
Size	wSize	3	O/I	Access size.	CMDreq.Size
VZ	wVZ	1	O/	Visibility	CMDreq.VZ
CA	wCA	1	O/I	Cacheable	CMDreq.AC
AC	wAC	1	O/I	Allocate Hint	CMDreq.AC
NS	wNS	1	O/I	Non-secure Access	CMDreq.NS
PR	wPR	1	O/I	Privilege	CMDreq.PR
LK	wLK	2	O/I	Lock[1:0]	CMDreq.LK
MW	wMW	1	O/I	Merging Write	Local
RL	wRL	2	O/I	Response Level	Local
EO	wEO	1	O/I	Exclusive Okay	Local
MPF1	wMPF1	6-12	O/I	Multi-purpose field #1	CMDreq.MPF2
TOF	wTOF	4	O/I	Transaction Ordering Framework	CMDreq.TOF
QoS	wQoS	4	O/I	QoS Label	CMDreq.QoS

Table 4-56: CCMP RBRreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
AUX	wReqAux	0-8	O/I	Auxiliary field	Local
MPROT	wReqProt	0-10	O/I	Protection bits for RBRreq Message	Local
Note:					
a. RType will no longer be used – Starting with Ncore 3.6 RBRreq will always be used to reserve one generation of RB and it will implicitly release all previously generations of the same physical buffer					

4.12.4 RBRreq field descriptions

4.12.4.1 RBID

RBID is the Request Buffer Identifier. RBID is the concatenation of the physical buffer number (\log_2 of the number of buffers) with the buffer's generation ID bit.

4.12.4.2 CMStatus

See Section 4.5.3.1.

4.12.4.3 RType

Deprecated

4.12.4.4 Addr

See Section 4.5.3.2.

The Addr field in RBRreq is copied from the CMDreq to which it is related.

4.12.4.5 Operation attributes

Attributes are associated with snoops as well. Operation attributes are carried by the RBRreq message in the following fields: VZ, AC, CA, NS, PR, LK, MW, and RL.

For all the above attributes, except MW, see Section 4.5.3.3.

4.12.4.5.1 MW

DMI uses MW bit (when set to 1) to reserve a Merge Buffer, if available.

This bit is set for RBRreq for the CMDreq of WrUnqPtl (CCMP operation of WriteUniquePartial), which may require merging of Primary Write data with Intervention data.

1: Merging Write indicated.

0: Merging Write NOT indicated

When this bit is set, DMI reserves a cacheline sized data buffer and uses it to merge the Primary and Secondary data (DTW).

Per CCMP protocol, Secondary (Intervention) data (Secondary DTW) arrives first, and then Primary Write data (Primary DTW). CCMP protocol ensures that Secondary DTW arrives at the DMI strictly before Primary DTW arrives at the DMI.

When MW = 1, the DMI issues an RBRrsp only after the Primary DTW has been processed. Buffer is deallocated only after the Primary data has been processed.

If the buffer is not allocated, Secondary and then Primary data are handled independently, but strictly in the order of their arrival.

4.12.4.5.2 RL[1:0]

The RL[1:0] field in RBRreq message is set to 0b01 or 0b10 for *reserving* the RB, which merely represents an acknowledgment of a receipt of the RBRreq message.

~~The RL[1:0] field in RBRreq message must be set to 0b10 for releasing the RB, which returns the RBRrsp only after the RBRreq has been processed by the DMI.~~

4.12.4.5.3 EO

Exclusive Okay, see Section 4.6.5.6.

The DCE sets this bit in the RBR message if the Exclusive Monitor indicates that the Exclusive Read CMD has successfully performed at point of serialization and if the DCE needs to send snoops for the exclusive Read.

The bit is provided in anticipation of a DTWMrgMRDreq from the snooper, in response to which the DMI would need to issue a DTRreq.

The EO bit delivered by the RBRreq gets registered in the RB data structure for later use in the DTRreq that might be issued by the DMI to the requesting AIU.

If a DTWMrgMRD is used by the snooping AIU to deliver the data to the requesting AIU, the DTRreq receives the value for its EO field from the RB.

For CMD requests other than exclusive Reads, this bit is don't care and must be set to 0 by the DCE in the RBRreq.

4.12.4.6 TOF

See Section 4.5.3.9.

4.12.4.7 QoS

See Section 4.5.3.10

4.12.4.8 MPF1

Used to carry the Flow ID from the original Cmd message. This may be used for cache allocation policy.

4.12.4.9 AUX

See Section 4.5.3.11.

4.12.4.10 MPROT

See Section 4.5.3.12.

4.12.5 RBRrsp Message

After receiving the RBRreq message, the DMI issues an RBRrsp message back to the DCE to acknowledge the receipt of the former message. It returns a credit to the DCE to send another RBRreq. As explained in Section 4.12.4, the DCE also needs an unreserved Request Buffer before it can issue another RBRreq.

4.12.6 RBRrsp message fields

Table 4-57 shows the fields in an RBRrsp message.

Table 4-57: CCMP RBRrsp message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:					CCMP Message Header
TargetId	wTargetId	6-12	O/I	Target Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	FUnitId(CMDreq.D Id)
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	Local: DCE FUnitId
CMTYPE	wCMTYPE	8	O/I	Type of Message	Local. 0xF9
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTYPE, and MessageId.	Local
CMHE:					CCMP Message Header Extension
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:					CCMP Message Body
RBID	wRBID	6-10	O/I	RBID - the most significant bit will be used to indicate the RB generation	Local
CMStatus	wCmStatus	8	O/I	Output and Input for RBRrsp. Used to deliver response and error report to DCE.	Local
AUX	wRspAux	0-8	O/I	Auxiliary field	Local
MPROT	wRspProt	0-10	O/I	Protection bits for RBR Response Message	Local

4.12.7 RBRrsp field descriptions

4.12.7.1 RBID

RBID is the Request Buffer Identifier. RBID is the concatenation of the physical buffer number (\log_2 of the number of buffers) with the buffer's generation ID bit.

4.12.7.2 RBGen

Generation Index which is released for the physical RB associated with the RBID sent in the RBReq. If RBRrsp is sent in direct response to re-reserving the same physical buffer for a new transaction, this field shall contain the index that was used before. If the response was sent to release a buffer that has been used by data, RBGen shall carry the generation of the RB used for that data.

4.12.7.3 CMStatus[7:0]

Refer to Table 4-4.

4.12.7.4 AUX

See Section 4.5.3.11.

4.12.7.5 MPROT

Refer to Section 4.5.3.12.

4.13 SYSreq and SYSrsp

The purpose of system messages is to convey operating state information or state transitions for agents or asynchronous events within the endpoints of an Ncore domain. In the future they will be used to implement new functionality in the general class of system related activities.

This service is implemented by a new message class, **System**. System messages will be sent from a source agent to one or more destination agents. Ncore doesn't support broadcasting messages as part of the protocol, therefore System commands rely on a redistribution service implemented as a special Function Unit – as of Ncore 3.x, DVE assumes the responsibility to broadcast system messages.

Every system message must receive a response to acknowledge receipt by the destination agent - depending on the protocol implemented by a specific message pair, receipt of a response may initiate additional activity (for example a state transition at the source agent).

4.13.1 SYSreq message

SYSreq message is an implementation of the new system message class. The purpose of these messages is to propagate system related information between agents or functional units within the system.

SYSreq messages are asynchronous, do not use credit management as there shall be only one outstanding transaction per target, it shall be acknowledged by the receiver returning the SYSrsp message. The SYSrsp response implements status and error reporting in the CMStatus:

Command error²⁰:

- The command was not properly formatted
- No support for this operation
- Wrong unit - this unit is not supporting the OpCode or unknown OpCode
- The target unit is busy and cannot perform the requested operation at this time
- Ok - no error occurred; command completed as expected

SysReq OpCode:

- SysReq command uses CMType = 0x7B²¹
- SysRsp response uses CMType = 0xFB²²

²⁰ Some or all of these will not be supported in the initial release Ncore 3.2

²¹ This opcode is available - between 0x7A (STRreq) and 0x7C (RBRreq)

²² This uses one of the few free opcodes within the response message block (0xF0 - 0xFF)

4.13.1.1 SYSreq Commands

As an asynchronous system service, SYSreq commands do not require credits at the target unit, this imposes restrictions on the use of SysReq messages. It requires that a functional unit must always be able to process a single SysReq command and generate a meaningful response using SysRsp. Agents that support DVM Snoops must issue SysReq.Attach to DVE to connect to DVM coherency domain, and SysReq.Detach to disconnect. In the first implementation, NCore 3.2 and Ncore 3.4 will support these operations:

Table 4-58: SYSreq commands

Command	OpCode	Payload	Function
SysReq.NOP	0x00	TimeStamp ²³	No function, beyond carrying the timestamp in both directions. This transaction may be used to measure the roundtrip latency between two agents. If an agent does not implement a timestamp counter, the value used in the response shall be -1 (all ones)
SysReq.Attach	0x01	TimeStamp ²³²² SysCoReq	Attach the requester to the coherency domain. This command is used by a cache coherent initiator (CAIU) to attach itself to the coherency engine (DCE). The command needs to be sent to all interleaved DCE. The OK response signals successful attachment, only after that the agent will be allowed to send coherent requests and participate in snoop protocol
SysReqDetach	0x02	TimeStamp ²³²² SysCoReq	Detach the requester from the coherency domain. This command removes the agent from the set of snooped coherent agents managed by all DCE
SysReq.Event	0x03	TimeStamp ²³²² EventReq	System Event - these events are generated within various agents ²⁴ in the system and convey asynchronous events between event producers and event-consumers. These agents are CPUs, monitors for exclusive transactions, SMMU, Accelerators or other agents. Agents may be producer, consumers or both.

4.13.2 SYSreq message fields

Table 4-59: CCMP SYSreq message fields shows the fields in an SYSreq message.

Table 4-59: CCMP SYSreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:					
TargetId	wTargetId	6-12	O/I	Target Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	FUnitId(CMDreq.D Id)
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	Local: DVE/DCE/AIU FUnitId
CMTYPE	wCMTYPE	8	O/I	Type of Message	Local. 0x7D
MessageId	wMessageId	6-12	O/I	MessageID	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTYPE, and MessageId.	Local
CMHE:					
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.

²³ Timestamp will not be part of the initial implementation in Ncore 3.2

²⁴ In Ncore 3.2 and 3.4, only DCE (exclusive monitors) and AIU (EventInReq) will issue Event messages

Table 4-59: CCMP SYSreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:					CCMP Message Body
RMessageID	wRMessageID	6 - 12	O	MessageID for the command, to be returned with the response	Local
CMStatus	wCmStatus	8	I	Status - using standard command status layout	Local
TimeStamp ²³²²	wTimeStmp	0/32/64	O/I	Timestamp information – will be used to synchronize local timestamp counters to a centralized counter	Local
MPROT	wReqProt	0-10	O/I	Protection bits for SYSreq Message	Local

4.13.3 SYSreq field descriptions

4.13.3.1 RMessageID

Message ID of this SYSreq, to be returned in the SYSrsp as part of the receipt for the transaction.

4.13.3.2 CMStatus[7:0]

The CMStatus[7:0] field is shown only as input in the direction column, implying that the field is not transmitted along with the rest of the SYSreq message at the initiator, but is an input-only field at the target of the message.

The CMStatus[] field is used to report the result of a processing step of the operation, including detection of errors, to an Ncore unit. However, SYSreq being the start of the operation, there are no results to report.

If any type of error is detected in the native request transaction, the initiator AIU must not issue a corresponding SYSreq message into the Ncore system to commence the processing of the native operation. The operation must be locally terminated, and errors reported back to the native agent.

A SYSreq message may encounter a transport error along its path to the target. This error must be reported to the target via the CMStatus[] field constructed at the transport interface of the target Ncore unit.

4.13.3.3 AUX

See Section 4.5.3.11.

4.13.3.4 MPROT

Refer to Section 4.5.3.12.

4.13.4 SYSrsp message

Table 4-60: CCMP SYSrsp message fields shows the fields in an SYSrsp message.

Table 4-60: CCMP SYSrsp message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:				CCMP Message Header	
TargetId	wTargetId	6-12	O/I	Target Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	FUnitId(CMDreq.D Id)
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	Local: DVE/AIU/DCE FUnitId
CMTypE	wCMTypE	8	O/I	Type of Message	Local. 0xFA
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTypE, and MessageId.	Local
CMHE:				CCMP Message Header Extension	
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:				CCMP Message Body	
RMessagId	wMsgId	6-10	O/I	Reference Message ID	SYSrsp.MessageID
CMStatus	wCmStatus	8	O/I	Output and Input for SYSrsp. Used to deliver response and error report to DVE.	Local
AUX	wRspAux	0-8	O/I	Auxiliary field	Local
MPROT	wRspProt	0-10	O/I	Protection bits for SYSrsp Message	Local

4.13.5 SYSrsp field descriptions

4.13.5.1 RMessagId

Message Id of the SYSreq whose reception is being acknowledged.

4.13.5.2 CMStatus[7:0]

The CMStatus[] field is used to report the result of a processing step of the operation, including detection of errors, to an Ncore unit. However, SysReq being the start of the operation, there are no results to report.

If any type of error is detected in the native request transaction, the initiator AIU must not issue a corresponding *SysReq* message into the Ncore system to commence the processing of the native operation. The operation must be locally terminated, and error reported back to the native agent.

A *SysReq* message may encounter a transport error along its path to the target. This error must be reported to the target via the CMStatus[] field constructed at the transport interface of the target Ncore unit.

Table 4-61: CMStatus for SYSrsp

Status Type	CMStatus [7:6]	CMStatus [5:0]
Success (No error)	2b00	SysRsp: System Response carries back the status of the command execution at the target agent
		[5:3] = 3b000 [2:0]: xx0: No Operation performed - this may not be an error! 001: Completion - Execution state machine is still running 011: Final - Execution state machine went to IDLE after sending this response 101 - 111: reserved (TBD)
		SysReq: [Reserved] - there should be no error on the incoming command, except transport problems
Error	2b01	SysRsp: System Response carries back the status of the command execution at the target agent
		[5:3] = 3b000 [2:0]: xx0: No Operation performed - unit signals error 001: Completion - with unknown error (conditions TBD) 011 - 110: Format Error - reserved 111: Unit is busy and cannot perform the requested operation at this time

4.13.5.3 AUX

See Section 4.5.3.11.

4.13.5.4 MPROT

Refer to Section 4.5.3.12.

4.14 UPDreqs and UPDrsp

UPD stands for *Directory Update*.

4.14.1 UPDreq message

This message is sent by an ACE-AIU to DCE to update the directory.

A UPDreq message requires system level guarantees of AIU-to-DCE forward progress to prevent deadlock.

As shown in Table 4-62, two types of UPDreq message are defined, however, UpdInv is used more often than the other.

Table 4-62 : UPDreq Message

UPDreq Message (Mnemonic)	Operation Description
Update Invalid (UpdInv)	<p>Update the agent state in the directory to invalid for the cacheline indicated. If no entry exists in the directory for the cacheline, the operation is treated as don't care. If the entry does exist and it does track the state of the cacheline within the agent, the entry is updated to indicate that the agent is in invalid state for the cacheline.</p> <p>The system <i>must guarantee</i> that an Update Directory Invalid operation, including all its associated protocol messages, can make forward progress under all circumstances.</p> <p>Note: This operation corresponds to the ACE Evict operation. It is also used in implementing WriteBack and WriteEvict transactions issued by an ACE agent.</p> <p>Final Cacheline States: Requester: I. Snooper: NA.</p>
Update Shared Clean (UpdSCln)	<p>Update the agent state in the directory to Shared Clean for the cacheline indicated. The directory entry is updated to indicate that the agent is in Shared Clean state for the cacheline.</p> <p>The system <i>must guarantee</i> that an Update Directory Shared Clean operation, including all its associated protocol messages, can make forward progress under all circumstances.</p> <p>Note: This operation may be used in updating the directory after a memory clean operation.</p> <p>Final Cacheline States: Requester: SC. Snooper: NA.</p>

4.14.1 UPDreq message fields

Table 4-63 shows the fields in an UPDreq message.

Table 4-63: CCMP UPDreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:					
TargetId	wTargetId	6-12	O/I	Target Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	FUnitId(CMDreq.D Id)
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	Local: DCE FUnitId
CMTyp	wCMTyp	8	O/I	Type of Message	Local: 0x70 for UpdInv 0x71 for UpdSCln

Table 4-63: CCMP UPDreq message fields

Name of field	Width Param	Param Values	Direction	Description	Source
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMType, and MessageId.	Local
CMHE:				CCMP Message Header Extension	
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:				CCMP Message Body	
CMStatus	wCmStatus	8	I	Input-only for UPDreq. Used to deliver Transport ncountered errors to the Target unit.	Local: RX Demux
Addr	wAddr	32-52	O/I	Address	NI
NS	wNS	1	O/I	Non-secure Access	NI
RL	wRL	2	O/I	Response Level	Local
QoS	wQoS	4	O/I	Quality of Service.	NI / Local
AUX	wReqAux	0-8	O/I	Auxiliary field	Local
MPROT	wReqProt	0-10	O/I	Protection bits for UPDreq Message	Local

4.14.2 UPDreq field descriptions

4.14.2.1 CMStatus

See Section 4.5.3.1.

4.14.2.2 Addr

See Section 4.5.3.2. The Addr field in UPDreq is copied from the address field of the native transaction in response to which it is generated.

4.14.2.3 Operation attributes

Attributes are associated with snoops as well.

Operation attributes are carried by the UPDreq message in the following fields: NS and RL. For all of the above attributes, see Section 4.5.3.3.

4.14.2.3.1 RL[1:0]

The RL[1:0] field in UPDreq message is set to 0b10, which requests a protocol level acknowledgment after the completion of the directory update operation.

4.14.2.4 QoS

See Section 4.5.3.10

4.14.2.5 AUX

See Section 4.5.3.11.

4.14.2.6 MProt

See Section 4.5.3.12.

4.14.3 UPDrsp Message

After receiving the UPDreq message, the DCE issues an UPDrsp message back to the AIU to acknowledge the receipt of the request message. It returns a credit to the DCE to send another UPDreq.

4.14.4 UPDrsp message fields

Table 4-64 shows the fields in an UPDrsp message.

Table 4-64: CCMP UPDrsp message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:					CCMP Message Header
TargetId	wTargetId	6-12	O/I	Target Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	FUnitId(CMDreq.D Id)
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	Local: DCE FUnitId
CMTYPE	wCMTYPE	8	O/I	Type of Message	Local. 0xF8.
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTYPE, and MessageId.	Local
CMHE:					CCMP Message Header Extension
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:					CCMP Message Body
RMessagId	wMsgId	6-10	O/I	Reference Message Id	UPDreq.MessageId
CMStatus	wCmStatus	8	O/I	Output and Input for UPDrsp. Used to deliver response and error report to DCE.	Local
AUX	wRspAux	0-8	O/I	Auxiliary field	Local
MPROT	wRspProt	0-10	O/I	Protection bits for UPD Response Message	Local

4.14.5 UPDrsp field descriptions

4.14.5.1 RMessageID

Message ID of the UPDreq whose reception is being acknowledged.

4.14.5.2 CMStatus[7:0]

Refer to Table 4-4.

4.14.5.3 AUX

See Section 4.5.3.11.

4.14.5.4 MPROT

Refer to Section 4.5.3.12.

4.15 Miscellaneous Response Messages

CCMP defines certain response-only messages.

4.15.1 CMPrsp Message

CMP stands for Completion.

This message is sent to an AIU to indicate completion of DVM operations.

4.15.2 CMPrsp message fields

Table 4-65 shows the fields in an CMPrsp message.

Table 4-65: CCMP CMDrsp message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:					CCMP Message Header
TargetId	wTargetId	6-12	O/I	Target Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	FUnitId(CMDreq.D Id)
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	Local: DCE FUnitId
CMTypE	wCMTypE	8	O/I	Type of Message	Local. 0xFC
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTypE, and MessageId.	Local
CMHE:					CCMP Message Header Extension
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:					CCMP Message Body
RMessagId	wMsgId	6-10	O/I	Reference Message Id	CMDreq.MessageId
CMStatus	wCmStatus	8	O/I	Output and Input for CMPrsp. Used to deliver response and error report to DCE.	Local
AUX	wRspAux	0-8	O/I	Auxiliary field	Local
MPROT	wRspProt	0-10	O/I	Protection bits for CMP Response Message	Local

4.15.3 CMPrsp field descriptions

4.15.3.1 RMessageld

Message ID of the CMDreq whose reception is being acknowledged.

4.15.3.2 CMStatus[7:0]

Refer to Table 4-4.

4.15.3.3 AUX

See Section 4.5.3.11.

4.15.3.4 MPROT

Refer to Section 4.5.3.12.

4.15.4 CMERsp Message

CME stands for Concerto Message Error.

This message is sent by an Ncore unit to another Ncore unit to indicate an error in protocol processing a Request message from the latter when a Response message for the Request has already been issued. For example, CMERsp can be used to indicate a Write completion error from a DMI or DII for a buffered Write.

4.15.5 CMERsp message fields

Table 4-66 shows the fields in an CMERsp message.

Table 4-66: CCMP CMERsp message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:				CCMP Message Header	
TargetId	wTargetId	6-12	O/I	Target Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	FUnitId(CMDreq.D Id)
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	Local: DCE FUnitId
CMTypE	wCMTypE	8	O/I	Type of Message	Local. 0xFD
Messageld	wMessageld	6-12	O/I	Messageld	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTypE, and Messageld.	Local
CMHE:				CCMP Message Header Extension	
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:				CCMP Message Body	
RMessageld	wMsgId	6-10	O/I	Reference Message Id	Request Message.Messageld
CMStatus	wCmStatus	8	O/I	Output and Input for CMERsp. Used to deliver response and error report to DCE.	Local
ECMType	wMessageld	8	O/I	Type of CCMP Message that encountered the error.	Request Message.CMTypE
AUX	wRspAux	0-8	O/I	Auxiliary field	Local
MPROT	wRspProt	0-10	O/I	Protection bits for CME Response Message	Local

4.15.6 CMErsp field descriptions

4.15.6.1 RMessageld

Message Id of the request message for which the error is being indicated.

4.15.6.2 ECMTType

Type of CCMP message that encountered the error being indicated by the CMStatus[].

4.15.6.3 CMStatus[7:0]

Refer to Table 4-4.

4.15.6.4 AUX

See Section 4.5.3.11.

4.15.6.5 MPROT

Refer to Section 4.5.3.12.

4.15.7 TRErsp Message

TRE stands for Transport Reported Error.

This message is sent by a Legato component unit to an Ncore unit to indicate an error during the transmission of a CCMP message from that unit to another unit.

4.15.8 TRErsp message fields

Table 4-67 shows the fields in an TRErsp message.

Table 4-67: CCMP TRErsp message fields

Name of field	Width Param	Param Values	Direction	Description	Source
CMH:					CCMP Message Header
TargetId	wTargetId	6-12	O/I	Target Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	FUnitId(CMDreq.D Id)
InitiatorId	wInitiatorId	6-12	O/I	Initiator Identifier is always a concatenation: {FUnitId, FPortId} where FPortId is always tied to 1'b0	Local: DCE FUnitId
CMTypE	wCMTypE	8	O/I	Type of Message	Local. 0xFE
MessageId	wMessageId	6-12	O/I	MessageId	Local
HProtection	wHProt	Derived	O/I	Protection for the Concerto Message Header fields TargetId, InitiatorId, CMTypE, and MessageId.	Local
CMHE:					CCMP Message Header Extension
TTier	wTTier	0-4	O/I	Traffic Tier of the message for use by the transport fabric.	Local. A function of Message Type.
Steering	wSteering	0-4	O/I	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message.	Local. A function of Message Type and other fields in the message body.
Priority	wPriority	0-4	O/I	Priority of the message to be used by the transport fabric.	Local. A function of the Message Type and QoS field in the message body.
QL	wQL	0-4	O/I	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronous property, etc. to be applied for this message.	Local. A function of the Message Type and QoS field in the message body.
CMB:					CCMP Message Body
RMessagId	wMsgId	6-10	O/I	Reference Message Id	Message.MessageId
CMStatus	wCmStatus	8	O/I	Output and Input for TRErsp. Used to deliver response an error report to an Ncore unit.	Local
ECMTypE	wMessageId	8	O/I	Type of CCMP Message that encountered the error.	Message.CMTypE
AUX	wRspAux	0-8	O/I	Auxiliary field	Local
MPROT	wRspProt	0-10	O/I	Protection bits for TRE Response Message	Local

4.15.9 TRersp field descriptions

4.15.9.1 RMessageld

Message Id of the request message for which the error is being indicated.

4.15.9.2 ECMTType

Type of CCMP message that encountered the error being indicated by the CMStatus[].

4.15.9.3 CMStatus[7:0]

Refer to Table 4-4.

4.15.9.4 AUX

See Section 4.5.3.11.

4.15.9.5 MPROT

Refer to Section 4.5.3.12.

4.16 Unit Pairs for Messages

The following matrix shows the Concerto-C messages the row unit sends to the column unit.

Table 4-68: Concerto-C Message Incidence Matrix

From\To	C-AIU	NC-AIU	DCE	DMI	DII	DVE
C-AIU	DTRreq DTRrsp CMErsp	DTRreq DTRrsp CMErsp	C_CMDreq SNPrsp STRrsp CMErsp UPDreq	NC_CMDreq DTWreq DTWMrgMRDreq DTRrsp CMErsp	NC_CMDreq DTWreq DTRrsp STRrsp CMErsp	NC_CMDreq DTWreq SNPrsp STRrsp CMErsp SYSreq ^{tt} SYSrsp (event)
NC-AIU	DTRreq ^{oe} DTRrsp CMErsp	DTRreq ^{DP} DTRrsp ^{qf} CMErsp	C_CMDreq SNPrsp STRrsp CMErsp UPDreq	NC_CMDreq DTWreq DTWMrgMRDreq DTRrsp CMErsp	NC_CMDreq DTWreq DTRrsp STRrsp CMErsp	NC_CMDreq DTWreq SNPrsp STRrsp CMErsp
DCE	SNPreq STRreq CMErsp UPDrsp	SNPred STRreq CMErsp UPDrsp ^{ss}		MRDreq HNTreq RBRreq CMErsp		SYSreq (event) SYSrsp (sysco)
DMI	DTRreq STRreq DTWrsp NC_CMDrsp CMErsp	DTRreq STRreq DTWrsp NC_CMDrsp CMErsp	MRDrsp HNTrsp RBRrsp CMErsp			SYSreq (event)
DII	DTRreq STRreq DTWrsp NC_CMDrsp CMErsp	DTRreq STRreq DTWrsp NC_CMDrsp CMErsp				
DVE	STRreq SNPreq DTWrsp NC_CMDrsp CMPrsp CMErsp SYSreq (event) SYSrsp (sysco/event)	Only for ACE-Lite or ACE5-Lite AIU: SNPred CMErsp	SYSreq (sysco) SYSrsp (event)	SYSrsp (event)		
CTF	TRErsp	TRErsp	TRErsp	TRErsp	TRErsp	TRErsp
Note:	<ul style="list-style-type: none"> o. Only if initiator is an ACE5-Lite NC-AIU performing stashing. p. Only if initiator is an ACE5-Lite NC-AIU performing stashing and Target is an NC-AIU with Proxy Cache. q. Only if the Target is an ACE5-Lite NC-AIU and the initiator has Proxy Cache to which Stash-Write has been done. r. Only if NC-AIU has a proxy cache instantiated within it. Proxy cache implements the UPDreq protocol. s. Only if NC-AIU has a proxy cache instantiated within it and it generates UPDreq t. AIU sends both, events and attach requests u. RBUreq/RBUrsp have been deprecated and removed from this matrix 					

Chapter 5: Coherency Processing

This chapter specifies how operations from native agents are processed, which include both coherent and non-coherent operations. In doing so, the chapter identifies causal connections between different CCMP messages that are issued and processed in order to carry out the native operation. The actions the various Ncore units take in response to native transactions are also specified.

Note that in the case of proxy caches, which are Ncore-internal agents, the transactions may not be manifested and instead translated directly into CCMP messages by these caches.

Apart from coherent operations, other native operations such as non-coherent operations and DVM Messages undergo specialized processing steps. These are also covered later in this chapter.

5.1 Processing of operations

The processing of an operation involves many messages and their respective responses. However, the processing of each operation can be broken down into specific set of elemental subprocesses, each of which involves a specific set of messages and their responses. This section describes each of the elemental subprocesses. A later section identifies which sequences of elemental subprocesses are involved in carrying out various categories of operations.

5.1.1 Command Request Process

This subprocess involves the AIU's analysis of the native transaction and determining the home Ncore unit to send a CMDreq message.

In the case of a coherent transaction, the home Ncore unit for the data is a *Distributed Memory Interface (DMI)* but for coherency it is a *Distributed Coherence Engine (DCE)*. Exactly which DMI and DCE units they are is a function of the address carried by the native transaction and the interleaving function that is operational at the address. The CMDreq is sent to the DCE unit first, which coordinates coherency processing for the operation.

- If it is a non-coherent transaction, the home Ncore unit can be a *Distributed Memory Interface (DMI)* or a *Distributed I/O Interface (DII)*. Exactly which unit it is, is a function of the address carried by the native transaction and the interleaving function that is operational at the address.
- If it is a DVM message, the home is the *Distributed Virtual Memory Engine (DVE)*.

The home unit receiving the CMDreq acknowledges its reception via the message CMDrsp.

CMDrsp returns a protocol credit to the requesting AIU and also marks the end of Command request step.

5.1.2 Snoop Process

This step is incurred in the processing of coherent operations.

The DCE consults the directory and determines to which of the non-requesting AIUs to issue snoops.

The DCE then creates a snoop message and multicasts it to the selected AIUs. Note that the same message is sent to every selected AIU, which is responsible for any native interface specific snoop transaction (see Section 4.6).

The AIU receives the snoop message and translates it into the appropriate snoop transaction on the native interface. In the case of a proxy cache, it is presented with an equivalent internal representation of the “snoop transaction.”

The agent generates a snoop response.

This snoop response can engender one or more *secondary subprocesses* within the AIU which are described below:

- Intervention Direct DTR Process,
- Intervention Indirect DTR Process, and
- Intervention DTW Process

If intervention data transfers are likely to happen, the DCE reserves a *Request Buffer (RB)* in the DMI via an RBRreq message and includes its identifier in the SNPreq for the snooper AIU to utilize.

The AIU ensures that these secondary subprocesses are completed before proceeding to the next step.

The AIU derives its own snoop response as a combination of the agent’s snoop response and the secondary subprocesses it undertook (See Section 4.6). This combined response is sent back to the DCE via the SNPrsp message.

The reception of SNPrsp message concludes the snoop process.

5.1.3 Intervention Direct DTR Process

When a snoop is for a Read operation, the agent snoop response may contain a full cacheline worth of data. This data is a copy of the cacheline containing the latest contents. This is called *intervention data response* from the agent.

This data is forwarded by the snooper AIU directly to the requester AIU and thence to the requesting agent using a DTRreq message.²⁵

The requesting AIU forwards this data to the requesting agent with a request for transport level acknowledgment (RL[1:0] = 0b01).²⁶

It then sends a DTRrsp to the snooper AIU that sent the DTRreq.

The Intervention Direct DTR Process itself is completed when the DTRrsp is received by the snooper AIU.

In certain cases, the snooper AIU may also need to perform the *Intervention DTW Process*, described below, to *clean* the intervention data to memory. Thus, if necessary, the snooper AIU must complete the *Intervention DTW Process*, as well, before performing the next step of sending the snoop response. (Note: The *Intervention DTW Process* can, and may, be carried out concurrently with *Intervention Direct DTR Process*.)

After the completion of *Intervention Direct DTR Process*, and if necessary, of the *Intervention DTW Process*, the snooper AIU sends the snoop response to the DCE.

²⁵ CCMP ensures that there is at most one AIU that may send such an intervention DTRreq

²⁶ Transport level acknowledgment is sufficient because the STRrsp is issued to DCE only after the data transfer has completed to the requesting agent

5.1.4 Intervention Indirect DTR Process

When a snoop is for a Read operation, the agent snoop response may contain a partial cacheline worth of data with valid bytes indicated by the corresponding Byte Enables set to 1. This data represents the latest values of the indicated bytes. The remaining bytes are valid in the home copy of the cacheline in memory.

This partial data is forwarded by the snooper AIU indirectly to the requester AIU via the home DMI by having it merged there with the full cacheline.

The snooper AIU issues a DTWMrgMRD to the home DMI. It also requests protocol layer completion of the DTWMrgMRD ($RL[1:0] = 0b10$), which requires the DMI to wait for the data to be delivered to the requesting agent before issuing a DTWrsp back to the snooper AIU.

The type of DTWMrgMRD with respect to the state of the cacheline implied depends on the Read operation.

The DTWMrgMRD refers to the RBID indicated by the DCE in the SNPreq.

The DTWMrgMRD is also marked with attribute Primary = 0, indicating that this is intervention data.

The DMI, upon receiving DTWMrgMRD, Reads the copy of the cacheline from memory. Note that DMI might find a copy of the cacheline in the SMC.

The DMI then merges the indicated bytes from the DTWMrgMRD into the copy of the cacheline. This updates this cacheline copy to the latest version of data.

The DMI then forwards this copy to the requesting AIU via a DTRreq, also requesting a protocol layer completion.

The type of DTRreq is determined by the type of DTWMrgMRDreq with respect to the type of cacheline state delivered to the AIU and its agent.

The cacheline copy is written to memory, which could mean updating the SMC.

Architecture Note

Note that processing of DTWMrgMRDreq involves *Read-Modify-Write* operation on the cacheline.

This operation is conducted in an *atomic* manner by the DMI with respect to other accesses to the cacheline referenced by the DTW message

The requesting AIU, after receiving the DTRreq from the DMI, forwards the cacheline to the requesting agent.

The requesting AIU then sends a DTRrsp to the DMI that issued the DTRreq. After the DMI receives the DTRrsp, it sends a DTWrsp to the snooper AIU. This process ends when the DTWrsp is received by the snooper AIU.

After the process ends, the snooper AIU sends the snoop response to the DCE.

5.1.5 Intervention DTW Process

Certain operations explicitly require the latest copy of the cacheline to be written to memory. In certain other cases, the CCMP protocol requires that the data be cleaned to memory to prevent data loss. Example of the latter is a Read operation in which the snooper sends out *Dirty* cacheline data, but the operation requires only a *Clean* copy. Both cases invoke the *Intervention DTW Process* in the snooper AIU. As with the Intervention DTR Processes, this process, too, is completed by the snooper AIU before it returns the snoop

response to the DCE.

This data can be a full cacheline or a partial cacheline worth.

In the case of a full cacheline transfer, the snooper AIU uses DTWDataFullDty message with attribute Primary = 0.

In the case of a partial cacheline transfer, the snooper AIU uses DTWDataPtlDty message with attribute Primary = 0.

The snooper AIU issues the selected DTWreq to the home DMI.

The DMI, upon receiving DTWreq, updates the cacheline in memory. After that, the DMI sends a DTWrsp to the snooper AIU.

This process ends when the DTWrsp is received by the snooper AIU.

After the process ends, the snooper AIU can send the snoop response to the DCE.

5.1.6 Write Data DTW Process

This process is undertaken by the requesting AIU to transmit data carried by the Write operation to its home unit. It is used for Write operations to both coherent and non-coherent locations.

5.1.6.1 Classification of Writes

CCMP distinguishes between Writes that are coherent snoopable, coherent non-snoopable, and non-coherent. Table 5-1 shows the CCMP classification among these three categories of Write operations initiated by CHI-based agents.

Table 5-1: Classification of CHI-generated Write operations in CCMP

Coherent Snoopable Writes	Coherent Non-snoopable Writes	Non-coherent Writes
WriteUniqueFull, WriteUniquePtl	WriteCleanFull, WriteCleanPtl, WriteBackFull, WriteBackPtl, WriteEvict	WriteNCFull, WriteNCPtl

Table 5-2 shows the CCMP classification among these three categories of Write operations initiated by ACE-based agents. WriteClean, WriteBack, and WriteEvict operations in ACE are classified as Non-coherent Writes because of need to guarantee forward progress.²⁷

Table 5-2: Classification of ACE-generated Write operations in CCMP

Coherent Snoopable Writes	Coherent Non-snoopable Writes	Non-coherent Writes
WriteUniqueFull, WriteUniquePtl	-	WriteCleanFull, WriteCleanPtl, WriteBackFull, WriteBackPtl, WriteEvict WriteNCFull, WriteNCPtl

5.1.6.2 The Write DTW process

In CCMP, the CMDreqs for the Coherent Writes, both Snoopable and Non-snoopable are sent to home DCE unit. Non-coherent Writes, on the other hand, are sent to the home DMI or DII unit, as determined by the address carried by the operation.

The *Write DTW process* starts at the initiator AIU with the reception of STRreq from the home unit.

The home unit supplies the AIU with an RBID at the destination Ncore unit via the STRReq. The AIU refers to this RBID in the DTWreq used to transmit the Writedata.

For coherent Writes, the destination Ncore unit is the DMI. For non-coherent Writes, the destination Ncore unit may be a DMI or DII.

For all Writes except the WriteEvict, DTWDataDtyFull or DTWDataDtyPtl are used, depending on whether the data being transmitted is a full or partial cacheline, respectively. Byte Enables associated with the bytes indicate which bytes are valid and which are not.

For WriteEvict operations, DTWDataClean is used, which carries only full cachelines. The *Primary* attribute is set to 1 for these DTWreqs.

The DTWreqs are sent with RL[1:0] = 0b10.

²⁷ See AMBAAXI and ACE Protocol Specification, Issue E, 22 Feb. 2013, Section C6.6.1

Prior to sending the DTW, the requesting AIU must interact with the requesting agent to retrieve Write data from the agent that is then forwarded with the DTW.

Upon receiving the DTWreq, the destination unit processes the DTW and performs the Write to memory. Depending on the allocation policy in effect, the Write could occur in the SMC, instead of in the system memory.

After the memory Write has been performed, the destination unit issues a DTWrsp back to the requesting AIU.

After the DTWrsp is received, the AIU issues STRrsp to the home Ncore unit. This represents the end of the Write Data DTW process.

5.1.7 The Write Stashing process

5.1.7.1 Write Stash Request and Snoop Process

A Write Stash operation from a native agent may explicitly identify a Target agent. The explicit Target agent identification is mapped into a Target AIU (unit identifier) by the requesting AIU, which will be included in the stashing CMDreq to the DCE.

If an explicit Target agent has not been identified, the DCE, based on the cacheline ownership information in the directory, selects the AIU currently with *unique* state for the cacheline as the Target AIU. It also chooses an agent identity based in an implementation specific manner.

If no AIU with *unique* state for the cacheline can be identified, the Write Stash operation is downgraded to a Write Unique.

5.1.7.2 The Full Cacheline Write Stashing Process

The full cacheline Write Stashing process starts when the requesting AIU receives an STRreq with CMStatus[7 : 0] set to 0b00xxxxx1. CMStatus[0] = Snarf = 1 indicates that the *Target agent* has indicated its acceptance of the invitation to stash data.

The STRreq delivers to the requester AIU the *StashAIUID* in the MPF1 field and stashing MessageId in the MPF2 field.

Upon receiving the STRreq, the requester AIU sends the cacheline to be stashed in a *DtrDataUDty* message with *StashAIUID* as the Target Id and *MessageId* in the *RMessageId* field.

The RL[1:0] field is set to 0b10 to ensure that DTRrsp from the target AIU represents the completion of stash data delivery to the Target agent at the agent's coherent protocol level.

Once the requester AIU receives the DTRrsp message, it sends the STRrsp to the DCE.

Once the DTRrsp is received, the DCE removes the hazard against the cacheline, and the Full Cacheline Writing Write Stashing process ends.

5.1.7.3 The Partial Cacheline Write Stashing Process

The partial cacheline Write Stashing process starts when the requesting AIU receives an STRreq with CMStatus[7 : 0] set to 0b00xxxxx1. CMStatus[0] = Snarf = 1 indicates that the *Target agent* has indicated its acceptance of the invitation to stash data.

The STRreq delivers to the requester AIU the *StashAIUID* in the MPF1 field and stashing MessageId in the MPF2 field.

Upon receiving the STRreq, the requester AIU sends the partial cacheline to be stashed in a *DTWMrgMRDreq* message to the home DMI with *StashAIUID* in the MPF1 field and *MessageId* in the MPF2 field.

The *RL[1:0]* field is set to 0b10 to ensure that DTWrsp from the DMI represents the completion of stash data delivery to the Target agent at the agent's coherent protocol level.

The *DTWMrgMRD* refers to the *RBID* indicated by the DCE in the STRreq. The *Primary* field in the *DTWMrgMRD* message is set to 1.

The DMI, upon receiving *DTWMrgMRD*, Reads the copy of the cacheline from memory. Note that DMI might find a copy of the cacheline in the SMC.

The DMI then merges the indicated bytes from the *DTWMrgMRD* into the copy of the cacheline. This updates this cacheline copy to the latest version of data.

The DMI then forwards this copy to the Target AIU via a *DTRDataUDty* message. The RL[1:0] for the DTRreq is set to 0b10, also requesting a protocol layer completion of the message at the Target AIU.

The identity of the Target AIU is given by the *StashAIUID* value that is found in the *MPF1* field of the *DTWMrgMRDreq* message. The *RMessageld* needed for the *DTRDataUDty* is taken from the *MPF2* field of the *DTWMrgMRD* message.

Since the state of the cacheline transferred to the agent is *unique-dirty*, the cacheline is not written to memory by the DMI.

The Target AIU receives the *DTRDataUDty* message and transfers the data to the Target agent.

After the transfer to the target agent is complete, the Target AIU sends DTRrsp back to the DMI.

After receiving the DTRrsp, the DMI sends a DTWrsp back to the requester AIU.

Once the requester AIU receives the DTWrsp message, it sends the STRrsp to the DCE.

Once the DTRrsp is received, the DCE removes the hazard against the cacheline, and the Partial Cacheline Write Stashing process ends.

5.2 Processing of coherent operations

Native agent coherent operations are processed in several sequential steps. Some steps are common to all operations, while some are specific to

- Upon receiving an agent read request, the requesting AIU issues a CMDreq message to a DCE.
- Once the CMDreq message has been accepted, the DCE generates a CMDrsp message and processes the CMDreq by examining its portion of the system directory and issuing some number of SNPreq messages to the required snooping AIUs.
- Each snooping AIU performs the necessary coherence operation and acknowledges the SNPreq message with a SNPrsp message. Depending on the result of the coherence operation, a snooping AIU may provide the data directly to the requesting AIU using a DTReq message.
- If none of the snooping AIUs are able to provide the cacheline data, the DCE issues an MRDreq message to a DMI. Additionally, the DCE issues a STRreq message to the requesting AIU, summarizing the results of the coherence operations and indicating the final cache state.
- Once the MRDreq message has been accepted, the DMI generates an MRDrsp message to the DCE, reads the requested memory location, and sends a DTReq message to the requesting AIU.
- Once the requesting AIU has received both the DTReq message, to which the unit generates a DTDrv message, and the STRreq message, the unit responds to the STRreq message with a STRrsp message, signaling the completion of the original agent read transaction to the DCE.

Similarly, native agent coherent write transactions are generally processed as follows:

- Upon receiving an agent write request, the requesting AIU issues a CMDreq message to a DCE.
- Once the CMDreq message has been accepted, the DCE generates a CMDrsp message and processes the CMDreq by examining its portion of the system directory and issuing some number of SNPreq messages to the required snooping AIUs.
- Each snooping AIU performs the necessary coherence operation and acknowledges the SNPreq message with a SNPrsp message.
- Once all the required SNPrsp messages have been received, the DCE issues a STRreq message to the requesting AIU.
- After the STRreq message has been received, the requesting AIU issues a DTWreq message to a DMI, which provides the DTWrsp message when the DTWreq message has reached the point of visibility.
- The DTWrsp message then causes the requesting AIU to issue a STRrsp message, signaling the completion of the original agent write transaction to the DCE.

Protocol transactions are further classified, based on their processing requirements in the system, as either **protocol coherent transactions** or **protocol update transactions**. For a given cacheline address, protocol coherent transactions *must* be initiated by an AIU in the order that the corresponding native agent requests were received on a particular native agent interface, and the protocol request messages associated with the transaction *must* be transmitted and processed in the same order throughout a Concerto system. Likewise, for a given cacheline address, protocol update transactions must also be initiated in the order that the native agent requests were received, and the protocol request messages *must* also be transmitted and processed in the same order. At all times, Concerto units *must* be able to process protocol request messages associated with a protocol update transaction out of order with respect to protocol request messages associated with any protocol coherent transactions.

Additionally, if more than one native agent request is received on a native agent interface at the same time, an AIU must respect the ordering required by the native agent interface; otherwise, the AIU may choose to initiate the corresponding protocol transactions in any order.

A native agent transaction that performs a full-cacheline access or a partial-cacheline access is completed with a single protocol transaction; a native agent transaction that performs a multiple-cacheline access is broken into multiple protocol transactions.

For more details on Concerto protocol transactions and protocol messages, see Chapter 4.

5.3 Snoop Protocols

This section specifies the protocols for snoop requests in CCMP.

The snoop tables indicate the Snoop request issued on the CHI-B interface, the possible responses received for the type of snoop request and the action taken by the CCMP units involved.

In the tables that follow X indicates that the field is “don’t care.”

5.3.1 Snoop Protocols

5.3.1.1 SnpNITCDtr

5.3.1.1.1 Snoop operation on CHI.

Table 5-3: SnpNITCDtr protocol on CHI

CHI Snoop Request: SnpUnique	U P	CCMP Snoop Response {RV, RS, DC, DT[1:0]}	Snooping AIU		CHI Completion Response
			DTRreq	DTWreq	
SnpResp_I		00000	-	-	CompData_I
SnpResp_SC		11000	-	-	
SnpResp_UC		10000	-	-	
SnpRespData_I		00010	DtrDataInv	-	
SnpRespData_SC		11010	DtrDataInv	-	
SnpRespData_UC		10010	DtrDataInv	-	
SnpRespData_UD		10010	DtrDataInv	-	
SnpRespData_SD		10010	DtrDataInv	-	
SnpRespData_I_PD	X	00011	DtrDataInv	DtwDataFullDty	
SnpRespData_SC_PD		11011	DtrDataInv	DtwDataFullDty	
SnpRespData_UC_PD		Illegal	-	-	-
SnpRespDataPlt_I_PD		00011	-	DtwMrgMrdInv	CompData_I
SnpRespDataPlt_UD		10011	-	DtwMrgMrdInv	

Table 5-4: DCE-issued MRDreq for SnpNITCDtr

Cumulative Retained State	MRD issued by DCE
Invalid (00)	MrdRdWInv
Shared-Clean (11)	MrdRdWInv
Owner (10)	MrdRdWInv

5.3.1.2 SnpNITCCIDtr

5.3.1.2.1 Snoop operation on CHI.

Table 5-5: SnpNITCCIDtr protocol

CHI Snoop Request: SnpUnique	U P	CCMP Snoop Response {RV, RS, DC, DT[1:0] }	Snooping AIU		CHI Completion Response
			DTReq	DTWreq	
SnpResp_I	X	00000	-	-	CompData_I
SnpResp_SC		Illegal	-	-	
SnpResp_UC		Illegal	-	-	
SnpRespData_I	X	00010	DtrDataInv	-	
SnpRespData_SC		Illegal	-	-	
SnpRespData_UC		Illegal	-	-	
SnpRespData_UD		Illegal	-	-	
SnpRespData_SD		Illegal	-	-	
SnpRespData_I_PD	X	00011	DtrDataInv	DtwDataFullDty	
SnpRespData_SC_PD		Illegal	-	-	
SnpRespData_UC_PD		Illegal	-	-	
SnpRespDataPtl_I_PD	X	00011	-	DtwMrgMrldInv	CompData_I
SnpRespDataPtl_UD		Illegal	-	-	-

Table 5-6: DCE-issued MRDreq for SnpNITCCIDtr

Cumulative Retained State	MRD issued by DCE
Invalid (00)	MrdRdWInv
Shared-Clean (11)	NA
Owner (10)	NA

5.3.1.3 SnpNITCMIDtr

5.3.1.3.1 Snoop operation on CHI.

Table 5-7: SnpNITCMIDtr protocol

CHI Snoop Request: SnpUnique	CCMP Snoop Response {RV, RS, DC, DT[1:0] + Snarf}	Snooping AIU		CHI Completion Response
		DTRreq	DTWreq	
SnpResp_I	X	00000	-	CompData_I
SnpResp_SC		Illegal	-	
SnpResp_UC		Illegal	-	
SnpRespData_I	X	00010	DtrDataInv	
SnpRespData_SC		Illegal	-	
SnpRespData_UC		Illegal	-	
SnpRespData_UD		Illegal	-	
SnpRespData_SD		Illegal	-	
SnpRespData_I_PD	X	00010	DtrDataInv	
SnpRespData_SC_PD		Illegal	-	
SnpRespData_UC_PD		Illegal	-	-
SnpRespDataPlt_I_PD	X	00011	-	DtwMrgMrdInv ^a
SnpRespDataPtl_UD		Illegal	-	CompData_I
Note:				
a. DtwMrgMrdInv updates the memory. Although not required, it is permitted for ReadNITCMI				

Table 5-8: DCE-issued MRDreq for SnpNITCMIDtr

Cumulative Retained State	MRD issued by DCE
Invalid (00)	MrdRdWInv
Shared-Clean (11)	NA
Owner (10)	NA

5.3.1.4 SnpCleanDtr

5.3.1.4.1 Snoop operation on CHI.

Table 5-9: SnpCleanDtr protocol

CHI Snoop Request: SnpClean	U P	CCMP Snoop Response {RV, RS, DC, DT[1:0] }	Snooping AIU		CHI Completion Response
			DTReq	DTWreq	
SnpResp_I		00000	-	-	CompData_UC
SnpResp_SC		11000	-	-	CompData_SC
SnpResp_UC		Illegal	-	-	-
SnpRespData_I		00110	DtrDataUCIn	-	CompData_UC
SnpRespData_SC		11010	DtrDataSCIn	-	CompData_SC
SnpRespData_UC		Illegal	-	-	-
SnpRespData_UD		Illegal	-	-	-
SnpRespData_SD		10010	DtrDataSCIn	-	CompData_SC
SnpRespData_I_PD	00	00011	DtrDataSCIn	DtwDataFullDty	CompData_SC
	01	00111	DtrDataUCIn	DtwDataFullDty	CompData_UC
SnpRespData_SC_PD		11011	DtrDataSCIn	DtwDataFullDty	CompData_SC
SnpRespData_UC_PD		Illegal	-	-	-
SnpRespDataPtl_I_PD		00111	-	DtwMrgMrdUCIn	CompData_UC
SnpRespDataPtl_UD		Illegal	-	-	-

Table 5-10: DCE-issued MRDreq for SnpCleanDtr

Cumulative Retained State	MRD issued by DCE
Invalid (00)	MrdRdWUCln
Shared-Clean (11)	MrdRdWSCln
Owner (10)	NA

5.3.1.5 SnpVldDtr

5.3.1.5.1 Snoop operation on CHI.

Table 5-11: SnpVldDtr protocol

CHI Snoop Request: SnpShared	U P	CCMP Snoop Response {RV, RS, DC, DT[1:0] }	Snooping AIU		CHI Completion Response
			DTRreq	DTWreq	
SnpResp_I		00000	-	-	CompData_UC
SnpResp_SC		11000	-	-	CompData_SC
SnpResp_UC		Illegal	-	-	-
SnpRespData_I		00110	DtrDataUCln	-	CompData_UC
SnpRespData_SC		11010	DtrDataSCln	-	CompData_SC
SnpRespData_UC		Illegal	-	-	-
SnpRespData_UD		Illegal	-	-	-
SnpRespData_SD		10010	DtrDataSCln	-	CompData_SC
SnpRespData_I_PD	00	00110	DtrDataSDty	-	CompData_SD_PD
	01	00110	DtrDataUDty	-	CompData_UD_PD
SnpRespData_SC_PD		11110	DtrDataSDty	-	CompData_SD_PD
SnpRespData_UC_PD		Illegal	-	-	-
SnpRespDataPlt_I_PD		00111	-	DtwMrgMrdUCln	CompData_UC
SnpRespDataPlt_UD		Illegal	-	-	-

Table 5-12: DCE-issued MRDreq for SnpVldDtr

Cumulative Retained State	MRD issued by DCE
Invalid (00)	MrdRdWUCln
Shared-Clean (11)	MrdRdWSCln
Owner (10)	NA

5.3.1.6 SnpInvDtr

5.3.1.6.1 Snoop operation on CHI.

Table 5-13: SnpInvDtr protocol

CHI Snoop Request: SnpUnique	U P	CCMP Snoop Response {RV, RS, DC, DT[1:0]}	Snooping AIU		CHI Completion Response
			DTRreq	DTWreq	
SnpResp_I		00000	-	-	CompData_UC / CompData_UD_PD
SnpResp_SC		Illegal	-	-	-
SnpResp_UC		Illegal	-	-	-
SnpRespData_I		00110	DtrDataUCIn	-	CompData_UC
SnpRespData_SC		Illegal	-	-	-
SnpRespData_UC		Illegal	-	-	-
SnpRespData_UD		Illegal	-	-	-
SnpRespData_SD		Illegal	-	-	-
SnpRespData_I_PD	00	00110	DtrDataUDty	-	CompData_UD_PD
	01			-	-
SnpRespData_SC_PD		Illegal	-	-	-
SnpRespData_UC_PD		Illegal	-	-	-
SnpRespDataPtl_I_PD		00111	-	DtwMrgMrdUDty	CompData_UD_PD
SnpRespDataPtl_UD		Illegal	-	-	-

5.3.1.6.1.1 DCE issued MRD requests for SnpInvDtr

Table 5-14: DCE-issued MRDreq for SnpInvDtr

Cumulative Retained State	MRD issued by DCE
Invalid (00)	MrdRdWU
Shared-Clean (11)	NA
Owner (10)	NA

5.3.1.7 SnpNoSDIntDtr

5.3.1.7.1 Snoop operation on CHI.

Table 5-15: SnpNoSDIntDtr protocol

CHI Snoop Request: SnpNotSharedDirty	U P	CCMP Snoop Response {RV, RS, DC, DT[1:0] + Snarf}	Snooping AIU		CHI Completion Response
			DTReq	DTWreq	
SnpResp_I		00000	-	-	CompData_UC
SnpResp_SC		11000	-	-	CompData_SC
SnpResp_UC		Illegal	-	-	-
SnpRespData_I		00110	DtrDataUCIn	-	CompData_UC
SnpRespData_SC		11010	DtrDataSCIn	-	CompData_SC
SnpRespData_UC		Illegal	-	-	-
SnpRespData_UD		Illegal	-	-	-
SnpRespData_SD		10010	DtrDataSCIn	-	CompData_SC
SnpRespData_I_PD	00	00010	DtrDataSCIn	DtwDataFullDty	CompData_SC
	01	00110	DtrDataUDty	-	CompData_UD_PD
SnpRespData_SC_PD		11011	DtrDataSCIn	DtwDataFullDty	CompData_SC
SnpRespData_UC_PD		Illegal	-	-	-
SnpRespDataPtl_I_PD		00111	-	DtwMrgMrdUCIn	CompData_UC
SnpRespDataPtl_UD		Illegal	-	-	-

5.3.1.7.1.1 DCE issued MRD requests for SnpNoSDIntDtr

Table 5-16: DCE-issued DCE-issued MRDreq for SnpNoSDIntDtr

Cumulative Retained State	MRD issued by DCE
Invalid (00)	MrdRdWUCLn
Shared-Clean (11)	MrdRdWSCln
Owner (10)	NA

5.3.1.8 SnpClnDtw

5.3.1.8.1 Snoop operation on CHI.

Table 5-17: SnpClnDtw protocol

CHI Snoop Request: SnpCleanShared	U P	CCMP Snoop Response {RV, RS, DC, DT[1:0] + Snarf}	Snooping AIU		CHI Completion Response
			DTRreq	DTWreq	
SnpResp_I		00000	-	-	Comp_I
SnpResp_SC		11000	-	-	Comp_SC
SnpResp_UC		10000	-	-	Comp_UC
SnpRespData_I		Illegal	-	-	-
SnpRespData_SC		Illegal	-	-	-
SnpRespData_UC		Illegal	-	-	-
SnpRespData_UD		Illegal	-	-	-
SnpRespData_SD		Illegal	-	-	-
SnpRespData_I_PD	X	00001	-	DtwDataFullDty	Comp_I
SnpRespData_SC_PD		11001	-	DtwDataFullDty	Comp_SC
SnpRespData_UC_PD		10001	-	DtwDataFullDty	Comp_UC
SnpRespDataPtl_I_PD		00001	-	DtwDataPtlDty	Comp_I
SnpRespDataPtl_UD		Illegal	-	-	-

5.3.1.8.1.1 DCE issued MRD requests for SnpCleanDtw

This snoop corresponds to the CleanShared CMO.

Table 5-18: DCE-issued MRDreq for SnpCleanDtw

Cumulative Retained State	MRD issued by DCE
Invalid (00)	MrdCln
Shared-Clean (11)	MrdCln
Owner (10)	MrdCln

5.3.1.9 SnpInvDtw

5.3.1.9.1 Snoop operation on CHI.

Table 5-19: SnpInvDtw protocol

CHI Snoop Request: SnpCleanInvalid	U P	CCMP Snoop Response {RV, RS, DC, DT[1:0] }	Snooping AIU		CHI Completion Response
			DTRreq	DTWreq	
SnpResp_I		00000	-	-	Comp_I
SnpResp_SC		Illegal	-	-	-
SnpResp_UC		Illegal	-	-	-
SnpRespData_I		Illegal	-	-	-
SnpRespData_SC		Illegal	-	-	-
SnpRespData_UC		Illegal	-	-	-
SnpRespData_UD		Illegal	-	-	-
SnpRespData_SD		Illegal	-	-	-
SnpRespData_I_PD	X	00001	-	DtwDataFullDty	Comp_I
SnpRespData_SC_PD		Illegal	-	-	-
SnpRespData_UC_PD		Illegal	-	-	-
SnpRespDataPlt_I_PD		00001	-	DtwDataPtlDty	Comp_I
SnpRespDataPtl_UD		Illegal	-	-	-

5.3.1.9.1.1 DCE issued MRD requests for SnpInvDtw

This snoop corresponds to the CleanInvalid CMO.

Table 5-20: DCE-issued MRDreq for SnpInvDtw

Cumulative Retained State	MRD issued by DCE
Invalid (00)	MrdFlush
Shared-Clean (11)	NA
Owner (10)	NA

5.3.1.10 SnpInv

5.3.1.10.1 Snoop operation on CHI.

Table 5-21: SnpInv protocol

CHI Snoop Request: SnpMakeInvalid	U P	CCMP Snoop Response {RV, RS, DC, DT[1:0] + Snarf}	Snooping AIU		CHI Completion Response
			DTReq	DTWreq	
SnpResp_I		00000	-	-	Comp_I
SnpResp_SC		Illegal	-	-	-
SnpResp_UC		Illegal	-	-	-
SnpRespData_I		Illegal	-	-	-
SnpRespData_SC		Illegal	-	-	-
SnpRespData_UC		Illegal	-	-	-
SnpRespData_UD		Illegal	-	-	-
SnpRespData_SD		Illegal	-	-	-
SnpRespData_I_PD	X	Illegal	-	-	-
SnpRespData_SC_PD		Illegal	-	-	-
SnpRespData_UC_PD		Illegal	-	-	-
SnpRespDataPlt_I_PD		Illegal	-	-	-
SnpRespDataPlt_UD		Illegal	-	-	-

4.5.3.3.1.1 DCE issued MRD requests for SnpInv

This snoop corresponds to the MakeInvalid CMO.

Table 5-22: DCE-issued MRDreq for SnpInv

Cumulative Retained State	MRD issued by DCE
Invalid (00)	MrdInv
Shared-Clean (11)	NA
Owner (10)	NA

5.3.2 Stashing Snoop Protocols

The stash transaction identifies a **target** native agent. Other agents are termed **non-target** agents. Stashing protocol requires that a different snoop transaction is presented to the **target** of the stash operation than one presented to a non-target snooping agent. Since only CHI-B (and later versions of CHI protocol) supports stashing functionality, only those agents connected to CHI-B ([and later versions of CHI protocol](#)) interfaces can be *valid targets*. All [CHI-E and ACE](#) agents are always *nontarget* agents.

Since stashing transactions are not supported on [CHI-E or ACE](#), if a stash transaction identifies [CHI-E or an ACE](#) agent as the *target*, only *non-target* snoops are generated on native interfaces.

5.3.2.1 SnplInvStsh

The stash transaction WriteUniqueFullStash transaction causes the DCE to generate the SnplInvStsh snoop message. This same message is sent to all AIUs that are candidates for snooping.

The AIU identified in the snoop message as the *target*, issues *SnpMakeInvalidStash* snoop transaction on its CHI-B/[E](#) interface while the non-target CHI-B and CHI-E AIUs generate *SnpMakeInvalid* snoops on their native interfaces. ACE AIUs generate *MakeInvalid* transaction on their native interfaces.

5.3.2.1.1 Snoop operation on Target CHI-B

Table 5-23: SnplInvStsh protocol on target CHI-B

CHI Snoop Request: <i>SnpMakeInvalidStash</i>	UP	CCMP Snoop Response {RV, RS, DC, DT[1:0] + Snarf}	Snooping AIU		CHI Completion Response
			DTRreq	DTWreq	
SnpResp_I		00000 + 0/1	-	-	Comp_I
SnpResp_SC		Illegal	-	-	-
SnpResp_UC		Illegal	-	-	-
SnpRespData_I		Illegal	-	-	-
SnpRespData_SC		Illegal	-	-	-
SnpRespData_UC		Illegal	-	-	-
SnpRespData_UD		Illegal	-	-	-
SnpRespData_SD		Illegal	-	-	-
SnpRespData_L_PD	X	Illegal	-	-	-
SnpRespData_SC_PD		Illegal	-	-	-
SnpRespData_UC_PD		Illegal	-	-	-
SnpRespDataPlt_I_PD		Illegal	-	-	-
SnpRespDataPtl_UD		Illegal	-	-	-

5.3.2.1.2 Snoop operation on non-target CHI-B or on CHI-E

On non-target native interfaces, *SnpInvStsh* in CCMP is translated into *SnpMakeInvalid* snoop.

Table 5-24: SnpInvStsh protocol on non-target CHI-B or CHI-E

CHI Snoop Request: <i>SnpMakeInvalid</i>	U P	CCMP Snoop Response {RV, RS, DC, DT[1:0] }	Snooping AIU	
			DTRreq	DTWreq
SnpResp_I		00000	-	-
SnpResp_SC		Illegal	-	-
SnpResp_UC		Illegal	-	-
SnpRespData_I		Illegal	-	-
SnpRespData_SC		Illegal	-	-
SnpRespData_UC		Illegal	-	-
SnpRespData_UD		Illegal	-	-
SnpRespData_SD		Illegal	-	-
SnpRespData_I_PD	X	Illegal	-	-
SnpRespData_SC_PD		Illegal	-	-
SnpRespData_UC_PD		Illegal	-	-
SnpRespDataPlt_I_PD		Illegal	-	-
SnpRespDataPtl_UD		Illegal	-	-

5.3.2.2 SnpUnqStsh

5.3.2.2.1 Snoop operation on Target CHI-B.

Table 5-25: SnpUnqStsh protocol on target CHI-B

CHI Snoop Request: SnpUniqueStash	U P	CCMP Snoop Response {RV, RS, DC, DT[1:0] + Snarf}	Snooping AIU		CHI Completion Response
			DTRreq	DTWreq	
SnpResp_I		00000 + 0/1	-	-	Comp_I
SnpResp_SC		Illegal	-	-	-
SnpResp_UC		Illegal	-	-	-
SnpRespData_I ^a	!01	00000 ^b + 0/1	-	Drop data ^b	Comp_I
	01	00001 + 0/1	-	DtwDataFullCln (CA=AC=1)	Comp_I
SnpRespData_SC		Illegal	-	-	-
SnpRespData_UC		Illegal	-	-	-
SnpRespData_UD		Illegal	-	-	-
SnpRespData_SD		Illegal	-	-	-
SnpRespData_I_PD	X	00001 + 0/1	-	DtwDataFullDty (CA=AC=1)	Comp_I
SnpRespData_SC_PD		Illegal	-	-	-
SnpRespData_UC_PD		Illegal	-	-	-
SnpRespDataPlt_I_PD		00001 + 0/1	-	DtwDataPtlDty (CA=AC=1)	Comp_I
SnpRespDataPtl_UD		Illegal	-	-	-

Note:

- a. This can come from SC or UC under SnpUniqueStash (see CHI-B Table 4-20).
- b. See Table Footnote a. Allowing DtwDataFullCln (CA=AC=1) here creates a prospect of issuing multiple DTWreqs to DMI if there is also an SD in the system. Hence the data is dropped here. Note that SnpRespData_I from UC will also be dropped. However, since it is a *Clean* copy from UC, no dirty data is lost. CMStatus[] in SNPrsp is adjusted appropriately.

5.3.2.2.2 Snoop operation on non-target CHI-B and CHI-E

On non-target native interfaces, *SnpUnqStsh* in CCMP is translated into *SnpUnique* snoop.

Table 5-26: SnpUnqStsh protocol on non-target CHI-B and CHI-E

CHI Snoop Request: <i>SnpUnique</i>	U P	CCMP Snoop Response {RV, RS, DC, DT[1:0] }	Snooping AIU		CHI Completion Response
			DTReq	DTWreq	
SnpResp_I		00000	-	-	
SnpResp_SC		Illegal	-	-	
SnpResp_UC		Illegal	-	-	
SnpRespData_I ^a		00001	-	DtwDataFullCln (CA=AC=1)	
SnpRespData_SC		Illegal	-	-	
SnpRespData_UC		Illegal	-	-	
SnpRespData_UD		Illegal	-	-	
SnpRespData_SD		Illegal	-	-	
SnpRespData_I_PD	X	00001	-	DtwDataFullDty (CA=AC=1)	
SnpRespData_SC_PD		Illegal	-	-	
SnpRespData_UC_PD		Illegal	-	-	
SnpRespDataPlt_I_PD		00001	-	DtwDataPtlDty (CA=AC=1)	
SnpRespDataPtl_UD		Illegal	-	-	
Note:					
a. This can come only from UC under <i>SnpUnique</i> since RetToSrc is set to 0.					

5.3.2.3 SnpStshShd

5.3.2.3.1 Snoop operation on target CHI-B.

Table 5-27: SnpStshShd protocol on target CHI-B

CHI Snoop Request: SnpStashShared	U P	CCMP Snoop Response {RV, RS, DC, DT[1:0] + Snarf}	Snooping AIU		CHI Completion Response
			DTRreq	DTWreq	
SnpResp_I		00000 + 0	-	-	Comp
SnpResp_I_Read		00000 + 1	-	-	Comp
SnpResp_SC		11000 + 0	-	-	Comp
SnpResp_SD		10000 + 0	-	-	Comp
SnpResp_UC		10000 + 0	-	-	Comp
SnpResp_UC_Read ^a		10000 + 1	-	-	Comp
SnpResp_UD		10000 + 0	-	-	Comp
SnpRespData_I		Illegal	-	-	-
SnpRespData_SC		Illegal	-	-	-
SnpRespData_SD		Illegal	-	-	-
SnpRespData_UC		Illegal	-	-	-
SnpRespData_UD		Illegal	-	-	-
SnpRespData_SD		Illegal	-	-	-
SnpRespData_I_PD	X	Illegal	-	-	-
SnpRespData_SC_PD		Illegal	-	-	-
SnpRespData_UC_PD		Illegal	-	-	-
SnpRespDataPtl_I_PD		Illegal	-	-	-
SnpRespDataPtl_UD		Illegal	-	-	-

Note:

a. Generated when agent is in UCE state.

5.3.2.3.2 Snoop operation on non-target CHI-B and CHI-E

Only if the stash target indicates Snarf = 1 in its snoop response to *SnpStshShd*, does the DCE initiate a *SnpStshShd* snoop to a non-target native interface. The *SnpStshShd* in CCMP is translated into *SnpShared* snoop on non-target CHI-B and CHI-E interfaces.

Table 5-28: SnpStshShd protocol on non-target CHI-B and CHI-E

CHI Snoop Request: <i>SnpShared</i>	U P	CCMP Snoop Response {RV, RS, DC, DT[1:0]}	Snooping AIU		CHI Completion Response
			DTRreq	DTWreq	
SnpResp_I		00000	-	-	-
SnpResp_SC		11000	-	-	-
SnpResp_UC		Illegal	-	-	-
SnpRespData_I		00001	-	DtwDataFullCln (CA=AC=1)	-
SnpRespData_SC		11001	-	DtwDataFullCln (CA=AC=1)	-
SnpRespData_UC		Illegal	-	-	-
SnpRespData_UD		Illegal	-	-	-
SnpRespData_SD		10001	-	DtwDataFullCln (CA=AC=1)	-
SnpRespData_I_PD	X	00001	-	DtwDataFullDty (CA=AC=1)	-
SnpRespData_SC_PD		11001	-	DtwDataFullDty	-
SnpRespData_UC_PD		Illegal	-	-	-
SnpRespDataPlt_I_PD		00001	-	DtwDataPtlDty	-
SnpRespDataPtl_UD		Illegal	-	-	-

The DCE issues an MRD message if Snarf is set in the snoop response from the target. It provides the data it has requested.

Table 5-29: DCE-Issued MRDreq for SnpStshShd

Cumulative Retained State	MRD issued by DCE if Snarf == 1	MRD issued by DCE if target is not identified
Invalid (00)	MrdRdWSIn	-
Shared-Clean (11)	-	-
Owner (10) ^a	MrdRdWUIn	MrdPref
Note:		b. Can only result from target agent returning Owner indication in SNPrsp. See Table 5-27.

5.3.2.4 SnpStshUnq

5.3.2.4.1 Snoop operation on target CHI-B.

Table 5-30: SnpStshUnq protocol on target CHI-B

CHI Snoop Request: SnpStashUnique	U P	CCMP Snoop Response {RV, RS, DC, DT[1:0] + Snarf}	Snooping AIU		CHI Completion Response
			DTRreq	DTWreq	
SnpResp_I		00000 + 0	-	-	Comp
SnpResp_I_Read		00000 + 1	-	-	Comp
SnpResp_SC		11000 + 0	-	-	Comp
SnpResp_SC_Read		11000 + 1	-	-	Comp
SnpResp_SD		10000 + 0	-	-	Comp
SnpResp_SD_Read		10000 + 1	-	-	Comp
SnpResp_UC		10000 + 0	-	-	Comp
SnpResp_UC_Read		10000 + 1	-	-	Comp
SnpResp_UD		10000 + 0	-	-	Comp
SnpRespData_I		Illegal	-	-	-
SnpRespData_SC		Illegal	-	-	-
SnpRespData_SD		Illegal	-	-	-
SnpRespData_UC		Illegal	-	-	-
SnpRespData_UD		Illegal	-	-	-
SnpRespData_SD		Illegal	-	-	-
SnpRespData_I_PD	X	Illegal	-	-	-
SnpRespData_SC_PD		Illegal	-	-	-
SnpRespData_UC_PD		Illegal	-	-	-
SnpRespDataPtl_I_PD		Illegal	-	-	-
SnpRespDataPtl_UD		Illegal	-	-	-

5.3.2.4.2 Snoop operation on non-target CHI-B and CHI-E

Only if the stash target indicates Snarf = 1 in its snoop response to *SnpStshUnq*, does the DCE initiate a *SnpStshUnq* snoop to non-target native interfaces. The *SnpStshUnq* in CCMP is translated into *SnpUnique* snoop on non-target CHI-B and CHI-E interfaces.

Table 5-31: SnpStshUnq protocol on non-target CHI-B or CHI-E

CHI Snoop Request: <i>SnpUnique</i>	CCMP Snoop Response {RV, RS, DC, DT[1:0]}	Snooping AIU		CHI Completion Response
		DTRreq	DTWreq	
SnpResp_I	00000	-	-	-
SnpResp_SC	Illegal	-	-	-
SnpResp_UC	Illegal	-	-	-
SnpRespData_I	00000	-	DtwDataFullCln (CA=AC=1)	-
SnpRespData_SC	Illegal	-	-	-
SnpRespData_UC	Illegal	-	-	-
SnpRespData_UD	Illegal	-	-	-
SnpRespData_SD	Illegal	-	-	-
SnpRespData_I_PD	X	00001	-	DtwDataFullDty (CA=AC=1)
SnpRespData_SC_PD		Illegal	-	-
SnpRespData_UC_PD		Illegal	-	-
SnpRespDataPlt_I_PD		00001	-	DtwDataPtlDty (CA=AC=1)
SnpRespDataPtl_UD		Illegal	-	-

The DCE issues an MRD message if Snarf is set in the snoop response from the target or if the target has not been identified. It provides the data it has requested.

Table 5-32: DCE-issued MRDreq for SnpStshShd

Cumulative Retained State	MRD issued by DCE if Snarf == 1	MRD issued by DCE if target is not identified
Invalid (00)	MrdRdWUCIn	-
Shared-Clean (11)	MrdRdWUCIn	-
Owner (10)	MrdRdWUCIn	MrdPref

5.4 Snoop Response Processing in DCE

5.4.1 Snoop responses

Depending on the type of coherent CMD request and the state of the cacheline registered in the directory at the current time, the DCE sends snoop messages to one or more snooper AIUs.

These AIUs translate the snoop message into a snoop transaction to their agents on the native interfaces. An agent so snooped sends a snoop response transaction in return. The nature of the snoop message might also prompt an *intervention data transfer* from the snooping agent.

5.4.1.1 Retained State of Cacheline

Snoop response from the agent provides an indication of the state of the cacheline retained by the agent *as a result of the snoop*. The indication of the **retained state of the cacheline** provided by the native agent may be precise, as in the case of CHI, or imprecise, as in the case of ACE.

The snooper AIU might process the snoop response provided by the agent and undertake additional actions per CCMP protocol, such as passing data directly to the requesting AIU and thence to its agent.

In response to the snoop, the snooper AIU also provides the DCE with a **snoop response** via the SNPrsp message. This response considers and reflects the original snoop response from the agent combined with the subsequent actions taken by the AIU.

The CCMP coherency directory classifies the states of the cacheline in three categories:

- **Invalid**,
- **Shared** (the Shared-Clean state), and
- **Owned** (SD, UC, UCE, UD, or UDP states) (see Section 2.4.2).

The CMStatus field within a SNPrsp carries the snoop response returned by the AIU (see Section 4.5.3.1).

CMStatus[5:4] indicate the *retained state* of the cacheline in the snooped AIU as recognized by the CCMP.

The retained state is encoded as follows:

- *Invalid* = 0b00,
- *Shared* = 0b11, and
- *Owned* = 0b10.

5.4.2 Implied Retained State in System

For any given snooping process, there may be AIUs in the system that the DCE chooses not to snoop. Since the DCE always snoops the AIU with *owner* state, any non-snooped AIU indicated in the directory as having cacheline presence is at most in *Shared* state. Thus, if there is even a single non-snooped AIU indicating cacheline presence, the **implied retained state** of the cacheline among non-snooped AIUs equals *Shared-Clean*. If there are no non-snooped AIUs indicating cacheline presence, the *implied retained state* in the system is *Invalid*.

5.4.3 Cumulative Retained State in System

The DCE combines the CMStatus[5:4] bits from snoop responses of individual snooped AIUs with the implied retained state contribution from non-snooped AIUs to compute a **cumulative retained state (CSR)** of the cacheline in the system. This *cumulative retained state* excludes the state of the cacheline in the requesting AIU, of course.

To calculate the cumulative state, bits 4 and 5 from the snoop responses are *ORed* respectively and then *ORed* with bit 0 and 1 respectively of the *cumulative retained state* implied by the non-snooped AIUs. Note that because of the invariants associated with cache states in CCMP (see Section 2.4.3), the resultant state can again be *Invalid*, *Shared*, or *Owner*.

5.4.4 Processing Snoop Responses from AIUs

After receiving the snoop response from the agent, the snooping AIU calculates the **Unit Snoop Response (USR)** tuple $\{RV, RS, DC, DT[1:0], Snarf\}$ (see Section 4.6.5.1) and sends it to the DCE, embedded within the CMStatus[7:0] field of the SNPrsp message. These tuples are specified in the tables for various snoop types in the snoop protocol tables of Section 5.3.

The DCE uses Snoop Responses for multiple purposes, discussed below.

5.4.5 Update of snooper's retained state

Bits *RV* and *RS* in the USR indicate the state of the cacheline retained by the agent as a result of the snoop as follows:

- 00:** Retained cacheline state is *Invalid*
- 10:** Retained cacheline state is *Owner*
- 11:** Retained cacheline state is *Sharer*
- 01:** Reserved code. Not a permitted response.

The directory directly registers this value for the cacheline state for the snooper.

5.4.6 Summary Snoop Response Calculation

After the DCE has received SNPrsp messages from all of the AIUs to which it issued the snoop message, the DCE calculates a **Summary Snoop Response (SSR)** tuple {SO, SS, SD, ST, Snarf/Ex-Okay} taking into account the *USR* tuples received via **SNPrsp** messages and the *Implied Retained State* of the cacheline in the directory (Section 5.4.2).

The bits in the SSR are set as follows:

- Bit SO is set if there is an *Owner* state in the rest of the system. Bit SS is set if there is a *Sharer* state in the rest of the system.
- Bit ST is set if an AIU has indicated $DT[1]=1$ in its *USR*. That is, the AIU has indicated that it has forwarded or has arranged to forward a copy of the cacheline to the requester AIU in response to a Read snoop.
- Bit SD is set if an AIU has indicated $DC=1$ in its *USR*. That is, the AIU has forwarded a *Dirty* copy of the cacheline to the requester AIU in response to a Read snoop.
- Bit Snarf is set if a snooper AIU has set the *Snarf* bit in its *USR*.
- Bit Ex-Okay is set if the DCE determines that the *Exclusive Store* by the requester has been successful.

For all non-Read commands, the SSR informs the native completion response issued by the requesting AIU to the requesting agent.

Note that for Read commands, the SSR tuple is only informational to the requesting AIU since the native completion response is instead derived from the type of DTRreq received by the requesting AIU.

5.4.7 STRreq generation

After the SSR value is computed, the DCE issues an STRreq message to the requesting AIU. Generation of STRreq may also imply additional architectural events. (See Section 4.11.1.)

The following table indicates for various CMDreqs dependencies for the generation of STRreq and, in turn, what architectural events the message implies for those CMDreqs.

Table 5-33: Dependencies and implications of STRreq generation for CMDreqs

CMDreq Mnemonics	Issued By	STRreq Dependencies	STRreq Architectural Event Implications
CmdRdCln, CmdRdVld, CmdRdNShDty, CmdRdUnq, CmdRdNITC, CmdRdNITCClnInv, CmdRdNITCMkInv, CmdClnUnq, CmdMkUnq	DCE	SNPrsp reception	Snoop processing complete. Coherently performed.
CmdRdNC	Target unit: DMI or DII	None	None.
CmdClnVld, CmdClnShPsist, CmdClnInv, CmdMkInv	DCE	SNPrsp reception MRDrsp reception	Snoop processing complete. Coherently performed when requested. System level visibility achieved when requested.

Table 5-33: Dependencies and implications of STRreq generation for CMDreqs

CMDreq Mnemonics	Issued By	STRreq Dependencies	STRreq Architectural Event Implications
CmdWrUnqPtl, CmdWrUnqFull	DCE	SNPrsp reception RBID availability	RB allocated at target unit; Data can be sent to target unit. Globally performed for coherency purposes.
Coherent: CmdWrAtm, CmdRdAtm, CmdSwapAtm, CmdCompAtm	DCE	SNPrsp reception	Coherency pre-operation performed.
Non-coherent: CmdWrAtm, CmdRdAtm, CmdSwapAtm, CmdCompAtm	Target unit: DMI	SNPrsp reception	An RB has been allocated at the target unit.
CmdWrStshFull, CmdWrStshPtl	DCE	SNPrsp reception	RB allocated at target unit; Data can be sent to target unit. Globally performed for coherency purposes. The Snarf bit when set indicates if the stashing request was accepted by the target agent.
CmdLdCchShd, CmdLdCchUnq	DCE	SNPrsp reception – if Stash-success feed-back (Snarf = 1) is to be delivered to the requesting unit. Otherwise: None	Operation completed with respect to requester.
CmdDVMMMsg	DVE	None	RB allocated at DVE; Data can be sent to it.
CmdWrEvict, CmdWrClnFull, CmdWrClnPtl, CmdWrBkFull, CmdWrBkPtl	DCE	RB availability	Performed globally for coherency purposes. RB allocated at DVE; Data can be sent to target unit.
CmdWrNCFull, CmdWrNCPtl	Target Unit: DMI or DII	RB availability	Performed globally with respect to all accesses via Ncore 3. RB allocated at DVE; Data can be sent to target unit.

Application Note

It might be possible the final values of SSR bits can be inferred before all the SNPrsp messages are received. This may allow STRreq message to be sent earlier.

In some cases, other considerations may require that STRreq to await reception of all SNPrsp messages, however. One example is when the system implements no filtering of snoops (Null Snoop Filter). In this case, SNPrsps may be used to signal the completion of intervention data transfers from snooper AIUs to the requester AIU and STRreq to signal the completion of all such possible data transfers.

5.4.8 Cache Maintenance Operation Processing

Semantics of Cache Management Operations (CMOs) are applicable to all caches in the system, including system level caches, if any exist.

A CMO snoop request issued to agents enforce the CMO semantics at the agent level. This enforcement may cause the cacheline to be cleaned or “cast out” to memory.

The snooper AIUs ensure that the DTW used to push a cacheline, or portion thereof, to memory has been completed coherently to DMI before issuing the SNPrsp message.

In the case of CMO commands *CleanShared*, *CleanSharedPersist*, *CleanInvalid*, and *MakeInvalid*, the reception of *SNPrsp* messages from snoopers indicates to the DCE that the CMO- appropriate actions have been performed and completed by snooped agents in their cache hierarchies. These operations involve:

- Cleaning dirty data to point of coherency (*CleanShared* or *CleanSharedPersist*),
- Cleaning dirty data to point of coherency and then invalidating agent’s caches (*CleanInvalid*), or
- Invalidating the cacheline in situ in the agent’s caches (*MakeInvalid*).

Since a System Memory Cache (SMC) can be in-line with the memory controller, the cacheline pushed out by an agent may get allocated into the SMC.

The CMOs semantics are also applicable to system caches. Since the SMC does not directly participate in the CCMP, the DCE sends the MRDreq messages to the appropriate DMI to enforce the CMO at the SMC after receiving all the expected SNPrsps.

The SMC may also independently hold a copy of the cacheline within itself. Therefore, even if no snoops are generated, the DCE must still issue the appropriate MRDreq messages to the requisite DMI to enforce the CMO there.

Table 5-34: MRDreq issued for CMOs

CMO	MRDreq
CleanShared	MrdClean
CleanSharedPersist	MrdClean
CleanInvalid	MrdFlush
MakeInvalid	MrdInvalidate

For MrdClean and MrdFlush, the DMI must ensure that if the SMC has a *dirty* copy of the cacheline, it is cleaned to system memory making that data system visible, as mandated by CMO semantics.

For all above MRD requests, the DMI must ensure that any accesses it may have outstanding to the memory are completed before sending an MRDrsp to the DCE. Completion of the MRD signals the completion of the CMO in the system.

5.4.9 DTRreq generation for Reads

In response to a snoop corresponding to a coherent Read command, the Read data could be directly transferred from a snooped agent to the requesting agent using a DTRreq, including, possibly via the DMI.²⁸

²⁸ For a system supporting ACE agents and implementing no snooping filters (i.e. Null Filter), multiple DTRreqs might occur depending on the type of snoop message and the states of the cacheline within the snooped agents

But, under certain cacheline states in the snooping agent and agent implementations, it is also possible that such a direct transfer does not occur.

If a snooping AIU intermediates a direct intervention data transfer from a snooping agent to the requesting agent, it sets the CMStatus[2] bit in its SNPrsp message to inform the DCE about such a transfer. If CMStatus[2] is negated in all of the snoop response messages returned to the DCE in response to a Read snoop, it indicates to the DCE that a direct intervention transfer to the requesting agent has not occurred as a result of the particular snoop. In absence of any direct intervention transfer, the DCE becomes responsible for supplying the data to the requester.

To determine the state of the cacheline to be returned to the requesting agent, the DCE uses the *cumulative retained state* of the cacheline computed above. Based on this state and the type of Read command, the DCE issues an appropriate MRDRd request to the DMI. The DMI then follows through by issuing the appropriate DTRreq to the requesting AIU.

The MRD request types used by the DCE are specified for each type of snoop in the subsections within Section 5.3.1 and Section 5.3.2.

5.5 Message Transcriptions

5.5.1 DTRreqs issued in response to MRDreqs

Table 5-35 below shows the DTRreqs issued by the DMI in response to different types of MRDreqs issued by the DCE.

Table 5-35: DTRreq resulting from MRDs

MRD issued by DCE	Cacheline state in System Memory Cache	DTRreq Issued by DMI
MrdRdWInv	x	DtrDataInv
MrdRdWSIn	x	DtrDataSIn
MrdRdWUIn	x	DtrDataUIn
MrdRdWU	Clean	DtrDataUCln
	Dirty	DtrDataUDty

5.5.2 Native Completion Responses for Reads

Table 5-36 below shows the Read Completion Responses produced on the CHI and ACE interfaces in response to DTRreqs received by the requesting agent's AIU.

Table 5-36: CHI/ACE native Read Completion Responses corresponding to DTRreqs

DTRreq	Read Completion Response delivered on CHI	Read Completion Response (RESP[3:2]) delivered on ACE	
		IS	PD
DtrDataInv	CompData_I	0	0
DtrDataSCln	CompData_SC	1	0
DtrDataSDty	CompData_SD_PD	1	1
DtrDataUCln	CompData_UC	0	0
DtrDataUDty	CompData_UD_PD	0	1

5.6 STRreq and Operation Completions

5.6.1 Coherent Read Operations

Tables below show {Reserved, Snarf} bits received via STRreq message from the DCE. The AIU does not depend on these bits to provide the CHI Completion response. That is derived from the DTRreq received either from a snooping AIU or a DMI.

5.6.1.1 ReadNITC

Table 5-37: ReadNITC protocol

STRreq received {Reserved, Snarf}	DTRreq	CHI Completion Response
00000	DtrDataInv	CompData_I
00000		

5.6.1.2 ReadNITCCI

Table 5-38: ReadNITCCI protocol

STRreq {Reserved, Snarf}	DTR	CHI Completion Response
00000	DtrDataInv	CompData_I
00000		

5.6.1.3 ReadNITCMI

Table 5-39: ReadNITCMI protocol

STRreq {Reserved, Snarf}	DTR	CHI Completion Response
00000	DtrDataInv	CompData_I
00000		

5.6.1.1 ReadUnique

Table 5-40: ReadUnique protocol

STRreq {Reserved, Snarf}	DTR	CHI Completion Response
00000	DtrDataUDty	CompData_UD_PD
00000	DtrDataUCIn	CompData_UC
00000	DtrDataUCIn	

5.6.1.2 ReadClean

Table 5-41: ReadClean protocol

STRreq {Reserved, Snarf}	DTR	CHI Completion Response
00000	DtrDataUCln	CompData_UC
00000	DtrDataUCln	
00000	DtrDataSCln	
00000	DtrDataSCln	
00000	DtrDataSCln	CompData_SC
00000	DtrDataSCln	

5.6.1.3 ReadValid or ReadShared

Table 5-42: ReadValid or ReadShared protocol

STRreq {Reserved, Snarf}	DTR	CHI Completion Response
00000	DtrDataUDty	CompData_UD_PD
00000	DtrDataUCln	CompData_UC
00000	DtrDataUCln	
00000	DtrDataSDty	CompData_SD_PD
00000	DtrDataSCln	
00000	DtrDataSCln	
00000	DtrDataSCln	CompData_SC
00000	DtrDataSCln	

5.6.1.4 ReadNotSharedDirty

Table 5-43: ReadNotSharedDirty protocol

STRreq {Reserved, Snarf}	DTR	CHI Completion Response
00000	DtrDataUDty	CompData_UD_PD
00000	DtrDataUCln	CompData_UC
00000	DtrDataUCln	
00000	DtrDataSCln	
00000	DtrDataSCln	
00000	DtrDataSCln	CompData_SC
00000	DtrDataSCln	

5.6.2 Dataless Operations

5.6.2.1 CleanValid or CleanShared

Table 5-44: CleanValid protocol

STRreq received {Reserved, Snarf}	DTR	CHI Completion Response
00000	-	Comp-I
00000	-	Comp_SC
00000	-	Comp-UC

5.6.2.2 CleanSharedPersist

Table 5-45: CleanSharedPersist protocol

STRreq {Reserved, Snarf}	DTR	CHI Completion Response
00000	-	Comp-I
00000	-	Comp_SC
00000	-	Comp-UC

5.6.2.3 CleanInvalid

Table 5-46: CleanInvalid protocol

STRreq {Reserved, Snarf}	DTR	CHI Completion Response
00000	-	Comp-I

5.6.2.4 MakeInvalid

Table 5-47: MakeInvalid protocol

STRreq {Reserved, Snarf}	DTR	CHI Completion Response
00000	-	Comp-I

5.6.2.5 MakeUnique

Table 5-48: MakeUnique protocol

STRreq {Reserved, Snarf}	DTR	CHI Completion Response
00000	-	Comp-UC

5.6.2.6 CleanUnique

Table 5-49: CleanUnique protocol

STRreq {Reserved, Snarf}	DTR	CHI Completion Response
00000	-	Comp-UC

5.6.3 Writes

For Write CMDreqs, the STRreq message indicates the permission to the requesting AIU to send data to the home unit (DMI or DII).

The STRreq delivers RBID, which is the identifier of a *Request Buffer* (see Section 4.12.1) in the home unit that must be referenced by the DTWreq message issued by the AIU.

The reception of the DTWrsp command is used by the AIU to signal completion of some of the Write operations.

Application Note

AIU implementations may use translate this permission in turn into a permission to the native agent to send its data to the AIU, which the AIU can forward to the home unit.

This method, however, increases the lifetime of the data buffer holding the Write data inside the agent as well as the lifetime of the operation itself. If AIU provides the permission to the native agent before the arrival of STRreq, the AIU must provide a buffer to hold the data until the STRreq arrives, which increases storage area inside the AIU.

5.6.3.1 WriteUniqueFull, WriteUniquePtl, WriteUniqueStashFull, and WriteUniqueStashPtl

Table 5-50: WriteUniqueFull/Ptl and WriteUniqueStashFull/Ptl DBIDResp protocol for CHI

STRreq	Result CHI Transaction
RBID	DBIDResp

Table 5-51: WriteUniqueFull/Ptl and WriteUniqueStashFull/Ptl Completion protocol for CHI

Message	Result CHI Transaction
DTWrsp	Comp

5.6.3.2 WriteBackFull, WriteBackPtl, WriteCleanFull, WriteCleanPtl²⁹, WriteEvictFull

Table 5-52: WriteBackFull/Ptl, WriteCleanFull/Ptl and WriteEvictFull DBIDResp protocol for CHI

STRreq	Result CHI Transaction
RBID	CompDBIDResp

Table 5-53: WriteBackFull/Ptl, WriteCleanFull/Ptl and WriteEvictFull Completion Protocol for CHI

Message	Result CHI Transaction
DTWrsp	-

²⁹ Occurs only in CHI-E system

Chapter 6: Consistency Processing

This chapter specifies various mechanisms available in the CCMP that can be used by elements of a coherency platform to achieve native transaction ordering semantics. These mechanisms include messages, their semantics, message dependencies, and message completions.

The goal of these mechanisms is to provide means to the coherency platform to achieve and satisfy consistency semantics expressed in the operations conducted by native agents via the CCMP domain.

In a system incorporating a CCMP domain, there may be extra-CCMP domain paths to storage. Only cache management operations initiated into the CCMP domain achieve system-wide consistency semantics across caches inside the CCMP-domain and system storage outside it.

6.1 Concepts

Consistency processing concerns itself with ensuring that performance of two or more storage operations executed by an agent is observed by all other agents in the system in their intended order.

6.1.1 Global observability of operations

The concept of observability of an operation is fundamental to consistency.

In CCMP, storage operations are expected to be **executed atomically** in system storage, i.e. without interruption and in entirety.

Execution of any two storage operations to a given coherency granule are **serialized**. Thus, if there are two operations A and B that reference the same coherency granule, then the final effect after both have been executed is one of only these two: Either execution of A followed by that of B or, execution of B followed by that of A.

An operation that has been issued by an agent but not yet been executed is termed as being **outstanding**.

In general, the result of an operation may not be visible to all the agents in the system at the same point in time.

An operation becomes **globally observable** or **globally visible** when every other operation that is *outstanding* in the system is able to observe the effect of the said operation.³⁰

A Read operation becomes *globally visible* when no other *outstanding* operation in the system to the same coherency granule can affect the value returned by it and, every other *outstanding* operation to the same coherency granule is able to observe the effect of the Read, including any side-effects. A Write operation becomes *globally visible*, when any *outstanding* Read operation from any agent to the same address returns the value deposited by the Write.

When an operation reaches a point of *global visibility*, it is said to have been **globally performed** (or simply, **performed**).³¹

CCMP distinguishes between two types of global visibility:

One, labeled **CCMP-global visibility (or CCMP-global observability)**, refers to *global visibility* to agents that may access a system storage location only via the CCMP domain (see Section 1.2.1.1). Second, labeled **System-global visibility (or System-global observability)**, on the other hand, implies *global visibility* to agents that may access a system storage location via the CCMP domain, as well as to those agents that may access the said location via **extraCCMP domain pathways**.

Global coherent visibility (or global coherent observability) implies CCMP-global visibility of a coherent operation in presence in the system of agents with cache hierarchies that may hold copies of a coherency granule.

CCMP does not automatically provide System-global visibility for every coherent operation accessed via CCMP domain. To enforce System-global visibility to locations accessed via such accesses, Cache Management Operations can be employed. (See Table 3-1.)

³⁰ Such other *outstanding* operations include operations that may have not yet been issued by other agents

³¹ When an operation issued by an agent reaches a point of visibility to all other operations to the same coherency granule that might be issued by that agent – or by a subset of agents in the CCMP domain – it is said to have been **locally performed**, with respect to that agent, or the said subset of agents

6.1.1.1 Indication of observability requirement for operations

The observability attribute in CCMP is the **VZ** bit (see Section 4.5.3.3.4). When the required observability is **CCMP-global**, the VZ bit is set to 0. When the required observability is *System-global*, the VZ bit must be set to 1.

Agents indicate the desired observability for an operation via its observability attributes. When AIUs transform the native operations to CMDreq messages, the observability attributes are translated into an equivalent setting for the VZ bit.

For native AXI agents, CCMP requires VZ to use following values, based on AxCACHE values received:

Table 6-1: Cache allocation and visibility policies

AxCACHE[4:0]	Visibility	VZ	Description
0000	Late (system)	1	Device Non-Bufferable
0001	Early	0	Device Bufferable
0010	Late (system)	1	Normal-Memory, non-cacheable, non-bufferable
0011	Early	0	Normal-Memory, non-cacheable, non-bufferable
011x ... 1xxx	Early	0	Normal-Memory, cacheable, different allocation policies

Note:

- a. These transactions must be directed towards a device interface (DII)
- b. It is not recommended to send non-cacheable transactions to an address range mapped to coherent (normal) memory. Sending non-cacheable transaction to cacheable address space breaks memory consistency.

For native CHI agents, Device == 1 and EWA == 0 translates to VZ = 1³². VZ is set to 0, otherwise.

6.1.2 Coherence ordering

In general, when two accesses to the same address are issued in the system by two distinct agents, without any prior mutually ordering action(s), no inherent order is implied between the two accesses. That is, the two accesses are **mutually unordered** when they are issued by the AIUs. This is also true in the case of coherent accesses.

However, in order to enforce coherency, a **total order** must be defined over all coherent accesses to a given coherency granule that might occur in the system. That is, previously *mutually unordered* operations get cast into a strict **sequence**.

This *total order* or *sequence* is called the **coherence order** of all the coherent accesses made to the said coherency granule.

Note

Coherence order refers to coherent accesses to a coherence granule from all native agents made into the CCMP domain, this defines an overall **global order**.

The distinct coherency granules are independent of each other and accesses to them are mutually unordered.

³² In CHI, Device == 1 and EWA == 0 corresponds to a no-Reordering; no-Early Response (nRnE) type access

6.1.2.1 Execution of coherent operations

Coherent operations belonging to a given *coherence order* are **executed** or *performed* one at a time, in a strict serial sequence, in accordance with the said coherence order. That is, one operation at a time that is currently first in the coherence order is executed to the point of *global coherent visibility* with respect to all the outstanding operations in the coherence order, before the next operation in the *coherence order* starts execution.

Operations belonging to distinct *coherence orders* may be executed concurrently.

6.1.3 Indication of global observability over native interfaces

The achievement of global observability for an operation is communicated via a **Completion** response over the CHI interface. The following table shows the events in CCMP that are necessary to occur before issuing a completion response for different types of operations.

The achievement of global observability for an operation is indicated via **OKAY** or **EXOKAY** codes on the **RRESP** or **BRESP** channels of an ACE interface.

See Section 6.3.4.1 for completion requirements at the AIU for indicating completions on native interfaces.

6.1.4 Ordered Write Observation

Ordered write observation is a concept in computer architecture that refers to the order in which write operations to memory locations are observed by different processing elements in a system. In a multi-core or multi-processor system, multiple processing elements may write to the same memory location at the same time. However, due to the inherent parallelism in these systems, the order in which these write operations are observed by different processing elements may not be deterministic.

In some cases, it is necessary to ensure that write operations are observed in a specific order by all processing elements in the system. This is particularly important in systems where data dependencies exist between different processing elements or where the order of execution of instructions is critical for correct program behavior.

To ensure ordered write observation in a system, a memory model or coherence protocol is used. These models specify the rules governing how memory operations are observed by different processing elements in the system, and how different memory operations are ordered with respect to each other.

One example of a memory model that provides ordered write observation is the Total Store Order (TSO) memory model used in x86 processors. Under TSO, all memory operations (reads and writes) are ordered in a total order, meaning that the order in which they are observed by all processing elements in the system is deterministic. This makes it easier to reason about the behavior of concurrent programs and ensures that program correctness is maintained even in the presence of concurrency.

Other memory models, such as the Relaxed Memory Order (RMO) model used in ARM processors, allow for more relaxed ordering of memory operations, but still provide mechanisms for enforcing ordering constraints when necessary.

Enforcing ordered write observation when writes are spread to more than one target can be challenging, as it requires coordinating the ordering of writes across multiple memory locations and processing elements. However, there are several techniques that can be used to ensure ordered write observation in such

scenarios.

One approach is to use a memory coherence protocol that provides ordering guarantees across multiple memory locations. For example, the MESI coherence protocol, which is commonly used in shared-memory systems, maintains a per-cache-line state that tracks the ownership and sharing status of the line. This protocol ensures that writes to a shared cache line are observed in the same order by all processors that have a copy of the line, regardless of which processor initiated the write.

Another approach is to use synchronization primitives such as locks, barriers, and fences to enforce ordering constraints on write operations. Locks can be used to serialize access to shared resources, ensuring that only one processing element can write to a shared resource at a time. Barriers can be used to enforce a specific ordering of memory operations across all processing elements, while fences can be used to explicitly specify ordering constraints between specific memory operations.

Finally, software algorithms can be designed to take into account the specific requirements for ordered write observation in a given application. For example, a distributed algorithm that requires ordered write observation across multiple nodes in a network can be designed to use a specific message-passing protocol that ensures message ordering, or to use a specific data structure that enforces ordering constraints.

In general, enforcing ordered write observation in systems where writes are spread across multiple targets requires careful design and coordination between the hardware and software components of the system. By using a combination of coherence protocols, synchronization primitives, and software algorithms, it is possible to ensure that write operations are observed in a specific order by all processing elements in the system, even when writes are spread across multiple memory locations.

6.2 Consistency mechanisms in CCMP

Mechanisms available in CCMP that are relevant to consistency achievement are discussed in this section.

6.2.1 Response Level: RL[]

Response level (RL[1:0]) field in CCMP messages is used to require performance of a message at the desired layer of the CCMP protocol, in order to insert and enforce causal dependencies between appropriate actions and events in the CCMP protocol execution. These dependencies when satisfied by the platform implementation, contribute to achievement of the desired ordering and consistency properties of the native protocol.

See Section 4.9.4.3.2 for more information on the field's usage.

6.2.2 CMDrsp

Coherent accesses: CMDrsp is issued by DCE for coherent CMDreqs. It indicates that the DCE has received the CMDreq. Additionally, CMDrsp also indicates that the coherent CMDrsp has been *coherently ordered* (also see Section 6.3.1).

Non-coherent accesses: CMDrsp is issued by DMI or DII for non-coherent CMDreqs. It indicates that the respective home node has received the CMDreqs. However, unlike for the coherent accesses, reception of CMDrsp may not guarantee global ordering of the CMDreq.

6.2.3 STRreq

STRreq is issued by home units to the requesting AIU per CMDreq. DCE issues STRreq for coherent CMDreqs. DMI and DII issue STRreq messages for non-coherent CMDreqs that they receive directly from AIUs.

Depending on the CMDreq, reception of STRreq indicates that the operation has reached the point of *CCMP-global visibility*³³, *Coherent-global visibility*, or *System-global visibility*.

Table 6-2 below shows the type of visibility represented by STRreq for various CCMP operations.

Table 6-2: Visibility semantics of STRreq

Operation Mnemonic (CMDreq Mnemonic)	STRreq semantics
ReadNITC (RdNITC)	CCMP-global coherent visibility
ReadClean (RDCLn)	
ReadValid (RdVld)	
ReadUnique (RdUnq)	
ReadNotShDirty (RdShDty)	
ReadNC (RdNC)	CCMP-global visibility
CleanUnique (ClnUnq)	CCMP-global coherent visibility
CleanValid or CleanShared (ClnVld)	System-global visibility
CleanSharedPersist (ClnShPsist)	
CleanInvalid (ClnInv)	
MakeInvalid (MkInv)	
MakeUnique (MKUnq)	

³³ Achievement of *System global visibility* is not always indicated via STRreq. See WrNCPl/Full with VZ=1. See Table 6-1

Table 6-2: Visibility semantics of STRreq

Operation Mnemonic (CMDreq Mnemonic)	STRreq semantics
Evict (Evict)	CCMP-global coherent visibility
WriteUniquePtl (WrUnqPtl)	
WriteUniqueFull (WrUnqFull)	
WriteCleanFull (WrClnFull)	
WriteBackPtl (WrBkPtl)	CCMP-global coherent visibility
WriteBackFull (WrBkFull)	
WriteEvict (WrEvict)	
WriteNCPtl (WrNCPtl) ^a	CCMP-global visibility
WriteNCFull (WrNCFull) ^a	
WriteStashPtl (WrStshPtl)	CCMP-global coherent visibility
WriteStashFull (WrStshFull)	
LoadCchShared (LdCchShd)	
LoadCchUnique (LdCchUnq)	
ReadNITCCleanInvalid (RdNITCCI)	
ReadNITCMakeInvalid (RdNITCMI)	CCMP-global visibility (Coherent visibility for coherent operations)
Read Atomic (RdAtm)	
Write Atomic (WrAtm)	
Compare And Swap Atomic (CompAtm)	
Swap Atomic (SwapAtm)	Not Applicable ^b
Prefetch (Pref)	
DVMOp (DVMMMsg)	CCMP-global visibility
Notes:	
a.	WriteNCPtl/Full, VZ=1 demands <i>System-global visibility</i> to be achieved for the Writeoperation, which the STRreq does NOT indicate, in this case. When it is achieved, it instead is reported via a DTWrsp message sent to the requesting AIU. See Section 6.2.8and Section 6.2.3.3.
b.	No visibility requirements defined for this operation.

6.2.3.1 Execution of operations in sequential order

To ensure that a pair of operations, except for WriteNoSnp with EWA = 0, referring to distinct coherency granules and with sequential ordering requirement of observability are performed in the system in the correct order, the AIU can await the receipt of STRreq for the first operation before issuing the CMDreq for the latter operation.

When WriteNoSnp with EWA = 0 is the first of the two operations, the AIU must wait until the operation is **complete** before issuing the second CMDreq. (See Section 6.2.3.3.)

6.2.3.2 Use of STRreq for Data Transfers

STRreq supplies an RBID whenever the requester AIU must send data associated with the operation. Correct RBID must be referenced in the DTWreq carrying the data to be sent.

All Write operations must therefore await the reception of STRreq before associated data can be sent by the requesting AIU to the Ncore 3 home unit – a DMI or DII.

Since Write Atomic, Swap Atomic, and Compare and Swap Atomic operations also involve sending a data

operand to the home unit, they must also await the reception of STRreq to transfer that data.

A DBIDResp is sent to the CHI agent to prompt it to issue a correlated data transaction on the WDAT channel. In general, the DBID response can be sent to the agent before or after the STRreq is received.

Implementation Note

A CHI AIU is permitted to wait for STRreq to arrive before it issues a DBIDResp to its CHI agent. This can reduce storage in the AIU but could also lower Write performance in the system.

For coherent operations, STRreq must be sent by the DCE after all SNPrsps are received for the given CMDreq. This ensures that the operation is globally visible for coherency purpose.

For stashing operations, *StashOnceUnique/Shared* and *WriteUniqueStashFull/Ptl*, the STRreq delivers an indication if the targeted native agent has accepted the stashing request by asserting the Snarf bit. For the WriteUniqueStash transactions, if the Snarf bit is set, the requesting AIU stashes the associated data into the target agent via a subsequent DTRreq, either directly or via the DMI.

6.2.3.3 Use of STRreq in operation completion indication

If the Comp response is offered separately from the DBIDresp, the issuance of Comp response must NOT wait for the arrival of data from the CHI agent.³⁴

- For *WriteBackFull/Ptl*, *WriteEvictFull*, and *WriteCleanFull* operations, the reception of STRreq allows the requesting CHI-AIU to offer a combined CompDBID response over the CHI interface.
- For *WriteUniqueFull/Ptl* operations, the reception of STRreq allows the requesting CHI-AIU to offer a DBIDresp and Comp response over the CHI interface. These responses can be sent separately or in a combined form.

The CHI-AIU must send STRrsp to the DCE only after the AIU has received a CompAck from the requesting agent.

- For *WriteNoSnpFull/Ptl* with *EWA==1* operations, which translates to a CMDreq with *VZ=0*, the reception of STRreq allows the requesting CHI-AIU to offer a DBID response as well as a Comp response over the CHI interface. The Comp response can be sent separately from the DBID response or combined with it.

For these operations, if the Comp response is issued separately by the CHI-AIU, its issuance need not wait for the data transfer.

Implementation Note

In Ncore 3, the Comp and DBID responses are issued in combined form.

- For *WriteNoSnpFull/Ptl* with *EWA==0* operations, which translates to a CMDreq with *VZ=1*, however, the reception of STRreq allows the requesting CHI-AIU to offer only a DBID response over the CHI interface. For these operations, the Comp response can be offered by the CHI-AIU only after the DTWrsp message is received from the home unit.
-

³⁴ Waiting for arrival of data from the agent before issuing the Comp response, may result in deadlocks in Producer-Consumer scenarios where the Producer waits for global visibility of its Write before transmitting the corresponding data

-
- For *CleanShared*, *CleanInvalid*, and *MakeInvalid* see Section 6.3.1.3.1.
-

6.2.4 STRrsp

A STRrsp is issued by a requesting AIU in response to an STRreq to indicate to the home unit the completion or performance of the operation at the requesting native agent.

The Completion actions and events over the native interface for various types of operations are stipulated by the respective native interface architecture. The AIU must satisfy those requirements.³⁵

In certain cases, these actions from the AIU include reception of an acknowledgment of AIU's actions from the requesting agent.

For CHI AIUs, if the native agent asserts the *ExpCompAck* signal when issuing a Request transaction, then its AIU must wait for a corresponding CompAck response transaction on the SRESP channel before issuing the STRrsp.

For an ACE AIU must wait for RACK or WACK, as appropriate for the operation under consideration, to be signaled by the agent, before issuing the STRrsp.

For Write operations, the requester AIU needs to issue a DTWreq message. In this case, the STRrsp may be issued only after the DTWreq has become *globally visible*. (See Section 6.2.8.) In this case, the STRrsp indicates that the operation is complete both at the requester as well as the home unit.

For WriteStashFull operations, the AIU may need to perform a DTRreq message. Such a DTRreq is sent with RL[1:0] = 11. In this case, the STRrsp may be issued only after the corresponding DTRrsp is received.. In addition to indicating completion at the requesting agent, the STRrsp indicates to the DCE, in this case, that the operation is also coherently complete at the stashing target.

For WriteStashPtl operations, the AIU may need to perform a DTWMrgMRDreq message. Such a DTWMrgMRDreq is sent with RL[1:0] = 11. In this case, the STRrsp may be issued only after the corresponding DTWrsp is received. In addition to indicating completion at the requesting agent, the STRrsp indicates to the DCE, in this case, that the operation is also coherently complete at the stashing target.

Generally, the AIU issues the STRrsp after it has performed all the necessary Completion actions for the operation over the AIU's native interface. However, in certain special circumstances, such as when Ordered Write Observation is required, it may wait for certain other conditions to be satisfied before issuing this STRrsp.³⁶

For a coherent native operation, STRrsp indicates that the said operation has been made visible and completed at the requesting native agent.³⁷ Only upon receiving the STRrsp from the requester AIU may the DCE remove the hazard related to the coherency granule referenced by the operation. And only thereafter can the DCE process a coherent CMDreq that is next in the same coherence order.

For DMI and DII, which issue STRreq for the non-coherent Write operations, the STRrsp indicates that the operation is complete at the requesting agent.

³⁵ These can be found specified in the AXI, ACE, and CHI architecture documents

³⁶ These include reception of STRreq messages for the previous relevant Write operations, indicating their global performance

³⁷ And thus, the requesting native agent is also ready to respond to snoops referring to the same coherency granule

6.2.5 UPDrsp

UPDreq of type *Update Directory Invalid* or *Update Directory Shared* is used by ACE AIU or IOAIU with Proxy Cache for WriteBack, and WriteEvict operations to update the state of the coherency granule for the agent or Proxy Cache in the directory within DCE. The DCE performs the update and returns the UPDrsp message. Thus, reception of UPDrsp indicates to the AIU global visibility of these operations.

The AIU indicates the global visibility and completion of these transaction in the system to the ACE agent via BRESP and to Proxy Cache via internal microarchitectural means only after the reception of UPDrsp.

For a detailed example of flow of CCMP messages for these operations, see Section 6.3.4.3.1.

6.2.6 SNPrsp

If the snooper AIU is required to issue a DTWreq that may have been produced as a result of a snoop response, the AIU may send the SNPrsp only after the DTWreq has become *globally visible* (see Section 6.2.8).

If the snooper AIU is required to issue a DTWMrgMRDreq, it must issue the message with an appropriate RL[1:0] setting. The AIU may then issue the SNPrsp only after the corresponding DTWrsp has been received. (See Section 6.2.8.)

If it is not necessary for the snooper AIU to ascertain that the receiving AIU has observed the DTRreq, the RL[1:0] should be set to 01 in the DTRreq. In this case, SNPrsp may be sent by a snooper AIU only after a DTRreq has been issued to the requester AIU.

When the CCMP domain is running without snoop filtering³⁸, it is possible that multiple snooper agents supply intervention data to the requesting agent. In this case, the CCMP requires that an SNPrsp issued to the DCE signals positively that the requester AIU has received the DTRreq that may have been sent to it by the snooping AIU issuing the SNPrsp.

To ensure that the requester AIU has observed the DTRreq at the CCMP layer, then the snooper AIU must set RL[1:0] = 10 in the DTRreq. This ensures that the DTRrsp is issued by the CCMP layer at the requester AIU and not by the underlying transport layer. In this case, the snooper AIU must wait for the reception of the DTRrsp before issuing the SNPrsp.

6.2.7 DTRrsp

DTRrsp is issued by the receiving AIU after it receives a DTRreq.

If RL[1:0] in DTRreq is set to 01, the DTRrsp can be sent as soon as the DTRreq is received without registering against the original command for which this might be the response. In this case, the DTRrsp merely confirms the delivery of the DTRreq to its destination.

If RL[1:0] is set to 10, the DTRrsp not only confirms the delivery of the DTRreq to its destination, but also acknowledges its reception by the CCMP protocol layer at the destination. In this case, the DTRrsp may only be sent after the DTRreq is registered against the original command for this DTRreq might be a response. (This method is used in Null filter case when the requesting AIU might receive multiple DTRreqs and it is necessary to ensure that SNPrsp from the snooper AIU indicates reception of data by the requester AIU and registration of that data against the original CMD. This ensures that when all the SNPrsps for the original

³⁸ Also referred to as Null Filter

CMD are received by the DCE, no further DTRreq delivery to the requesting AIU remains outstanding.)

If RL[1:0] is set to 11, the acknowledgment must be provided in a transitive manner all the way from the final destination of the data. That is, the DTWrsp may only be sent after the DTRreq has been translated into a data response transaction on the native interface and the AIU has received an acknowledgment from the native agent of having received that data response.

6.2.8 DTWrsp

DTWrsp is issued to an AIU by either DMI or DII in response to a DTWreq. By default, a DTWrsp indicates that the DTWreq is *CCMP-globally visible*.

If the original operation represented by the CMDreq is coherent, the DTWrsp also indicates that DTWreq has become *globally coherent visible*.

In case the original operation required a System-global visibility – i.e. VZ bit was set to 1 in the CMDreq –, the DTWrsp indicates that the DTWreq, and thus the CMDreq and operation, has achieved *System-global visibility*.

6.2.8.1 DTWrsp for DTWMrgMRDs

When a DTWMrgMRDreq is received, it, in turn, causes a DTRreq to be issued from the DMI.

If the DTWMrgMRD is sent with RL[1:0]=01, the DTRreq must be issued with RL[1:0]=01 and the DTWrsp may be issued after the DTRreq is issued, without waiting for DTWrsp.

If the DTWMrgMRDreq is received with RL[1:0]==10, the DTRreq must be issued with RL[1:0]=10. In this case, the DTWrsp must be issued only after receiving the corresponding DTWrsp.

If the DTWMrgMRDreq is received with RL[1:0]==11, the DTRreq must be issued with RL[1:0]=11. In this case, the DTWrsp must be issued only after receiving the corresponding DTWrsp.

6.3 Component Requirements

6.3.1 DCE requirements

A DCE receives CMDreq, UPDreq, STRrsp, and SNPrsp messages and issues CMDrsp, STRreq, MRDreq, and UPDrsp messages, that are relevant to consistency management in CCMP domain. In general, DCE must satisfy relevant requirements associated with these messages as specified in Section 6.2. Additional requirements, if any, are specified here.

6.3.1.1 Coherence ordering and processing in DCE

A DCE coordinates the processing of coherent accesses.

A CCMP-based coherency platform may contain many DCEs. All coherent accesses represented by the respective CMDreqs to a given coherency granule are sent to a unique DCE.

The DCE to which accesses for the given coherency granule are sent is responsible for defining the *coherence order* for that coherency granule.

Note that a given DCE typically is responsible for serializing accesses to multiple coherency granules, on a per coherency granule basis.

A distinct *coherence order* gets defined for each distinct coherency granule in the system, with no mutual ordering defined between operations addressed to different coherency granules. Thus, there may be multiple coherence orders active at a time in any given DCE.

Architecture Notes

If two coherent accesses match on the coherency granule referenced by their addresses, the two accesses are said to **collide**. To maintain coherency of the said granule across many accesses referencing it, such accesses are **serialized** into a *total order* over all coherent operations that may reference the said coherency granule. This order is referred to as the *coherence order* of these accesses.

In general, two coherent accesses to the same address are issued in the system by two distinct agents, there is no inherent order implied. Their order of execution is determined by the relevant DCE

Implementation Notes

Coherence order gets imposed on accesses to a coherency granule by the order in which the DCE processes the CMDreqs it receives.

A DCE is assumed to implement a buffer to receive CMDreqs sent to it by various agents in the system. While in this buffer, the DCE does not process the CMDreqs. This buffer is referred herein as an **ingress buffer**.

To process a CMDreq, the DCE reads at most one CMDreq out of the ingress buffer *at a time*³⁹, if one is available in the ingress buffer. After a CMDreq is read out of the ingress buffer, it is eliminated from there. The CMDreq may be stored in another data structure in the DCE while it is being processed.

As a CMDreq is read out of the ingress buffer, it is expected that it **atomically** gets coherently ordered with respect to all other applicable coherent CMDreqs issued within the CCMP domain. At most one CMDreq may get coherently ordered at a time.

CMDreqs to the same coherency granule address that were read out before the said CMDreq, are *ordered before or ahead* of the said CMDreq in the *global coherence order*; the ones that are read out after the said CMDreq are *ordered after or behind* it in the global coherence order.

The DCE processes the CMDreqs referencing a given coherency granule strictly in its coherence order. Thus, once so ordered, a CMDreq cannot be processed before the CMDreqs to the same coherency granule that were ordered before it. Nor can any CMDreq within the global order that is ordered *after* the said CMDreq may be processed ahead of the said CMDreq.

In order to ensure that at most one CMDreq to a given coherency granule is processed at any time, once the DCE reads a CMDreq out of the ingress buffer, **collision** process must be turned on against the coherency granule of the said CMDreq starting from the very next cycle.

This setting up of collision comparison against a coherency granule of a previously read CMDreq that is still being processed is called setting up a **hazard**.

If a CMDreq read out of the ingress buffer does not encounter a hazard, it is immediately eligible for coherent processing.

If a CMDreq encounters a hazard from a previous CMDreq, it is marked and held in a data structure in a **pending** state.

Each hazard is associated with a specific coherency granule. As CMDreqs are read out sequentially out of the ingress buffer, they form strictly ordered FIFO sequences of pending CMDreqs associated with their coherency granule, if there are earlier CMDreqs referencing it which are still being processed or in pending state. CMDreqs associated with a coherency granule are executed in the order of their sequence, with only the CMDreq in front of the sequence being actively processed. Once that CMDreq has been processed to completion, it leaves the sequence and the next pending CMDreq in the coherence order becomes eligible for processing.

A DCE processes one CMDreq within a given global coherence order at a time. Only after one is processed to completion, it takes up processing the next one.

³⁹ At a time, in this context, is equivalent to per DCE clock cycle

6.3.1.2 Consistency considerations for coherent accesses

Processing of a CMDreq involves enacting the CMDreq's all coherency semantics throughout the CCMP domain.

A coherent operation or access is globally visible when the CMDreq issued on its behalf is globally visible. A CMDreq, in turn, is globally visible when all agents within the CCMP domain can observe the results of the CMDreq.

Enforcement of coherency require actions at one or more of: Snooper AIUs, DMI, and the requester AIU.

The completion of enactment of coherency actions for a given CMDreq are signaled by the following CCMP protocol events.

- The reception of **SNPrsp** from the snooper AIU indicates completion of coherency actions at the snooping agent. For a given CMDreq, reception of SNPrsp from all snooper AIUs indicates that the CMDreq has been **globally performed** or has become **globally visible** to all coherent agents in the CCMP domain.
- For CMO operations, **MRDrsp** received from the DMI that is the home unit for the address referenced by the CMDreq indicates the completion of coherency actions at the home unit.
- The reception of **STRrsp** from the requesting AIU indicates completion of coherency actions at the requesting agent.

6.3.1.3 Consistency rules for DCE

Following rules must be followed by DCE to satisfy consistency requirements:

- DCE may issue the CMDrsp for a CMDreq only after the CMDreq has been *coherently ordered*. Thus, a CMDrsp message received from the DCE indicates that corresponding CMDreq has been coherently ordered.
- ACMDreq received by a DCE from any requester to the same coherency granule as an earlier CMDreq for which the DCE has already issued a CMDrsp, must be coherently ordered after that earlier CMDreq.⁴⁰
- DCE may issue STRreq for a CMDreq to the requesting AIU only after CMDreq has become *globally visible*.
- DCE may issue STRreq for CMO CMDreq to the requesting AIU only after the CMO has been completed at the target endpoint of the CMO. (Also see Section 6.3.1.3.1 below.)
- DCE must receive STRrsp for a coherent access before removing the hazard on the coherency granule referenced by the access.

Note that:

The DCE may not perform address or requester collision check when picking the CMDreq out of the ingress buffer. For example, under QoS, the DCE may pull CMDreqs out of the ingress buffer based on QoS values. Requester and address ordering under QoS must be done in the AIU and use appropriate message events and rules, so that DCE also processes them in the same order. DCE does not enforce any ordering for messages carrying different coherency granule addresses. Any ordering between such messages must be performed at the requesting AIU.

⁴⁰ This also guarantees Request Ordering for coherent accesses

6.3.1.3.1 Cache Management Operation Handling

Cache Management Operations (CMOs) can be coherent and non-coherent.

They are of type Clean (*CleanShared* or *CleanSharedPersist*), Flush (*CleanInvalid*), or Invalidate (*MakeInvalid*) cache copy (see Table 3-1).

CMOs can be sent to DCE, DMI, and DII.

The completion of a CMO indicates that the corresponding operation has been completed at the intended or target endpoint. For Clean and Flush, that endpoint is the home location of storage. Invalidate refers to copies in caches, both coherent and system-based, of the cacheline being referenced.

In CCMP, the completion of CMOs is indicated via STRreq. This means that STRreq must be sent to the initiating AIU only after the CMO has completed per above definition.

In the case of coherent CMOs, the CMO is sent to the DCE. The DCE then sends the corresponding MRDreq with RL = 2b10 to the DMI. To ensure that STRreq correctly indicates CMO completion, the MRDrsp must be issued by the DMI only after the CMO operation as communicated via the MRDreq has been completed.

Only after getting such an MRDrsp, can the DCE issue an STRreq back to the AIU.

The AIU must then deliver the appropriate CMO completion response to the agent on its native interface, and only then issue the STRrsp back to the DCE.

6.3.2 DMI consistency requirements

A DMI receives CMDreq, MRDreq, DTWreq, DTWMrgMRDreq, DTRrsp, and STRrsp messages and issues CMDrsp, STRreq, MRDrsp, DTWrsp, and DTRreq messages, that are relevant to consistency management in CCMP domain. In general, DMI must satisfy relevant requirements associated with these messages as specified in Section 6.2. Additional requirements, if any, are specified here.

The following are assumed for a DMI:

- CMDs, MRDs, and DTW messages may all reach a DMI via physically *independent* hardware ports, and as such, are not inherently ordered with respect to each other.
- CMDs, MRDs, and DTWs sent to the DMI are first written to **receive** or **ingress buffers** at the DMI's boundary. Messages of multiple types may share a ingress buffer.
- The DMI **receives** a message when the DMI *reads* it out of its *ingress buffer*.

6.3.2.1 Consistency rules for DMI

In the following, any statement referring to DTWs, also applies to DTWMrgMRDs.

In DMI, if two messages refer to the same coherency granule, they are treated as carrying the same address for ordering purposes.

DMI treats the size of end-point device to be equal to the size of a coherency granule. Thus, messages referring to different addresses may also refer to distinct end-point devices.

The DMI must execute the micro actions so that the effects of DTWs and MRDs, the memory Reads and Writes over the AXI interface to memory controller, the SMC Reads and updates, are in concert with the system order of the Read and Write operations from native agents that may have submitted them.

To achieve the required correspondence the DMI must satisfy the following ordering or consistency requirements:

- DMI must satisfy the semantics of the OR[1:0] field in the CMDreq it receives. (See Section 4.5.3.3.4.)
- Once DMI issues a DTWrsp for a DTW, any subsequent MRD to the same address must see the result of the DTW.
- Once DMI issues a DTWrsp for a DTW, any subsequent non-coherent Read to the same address must see the result of the DTW.⁴¹
- Once DMI issues a DTWrsp for a DTW, any subsequent DTW to the same address must overwrite the result of the earlier DTW. The two DTWs must also be performed in memory in the order of their reception.
- Once DMI issues a DTRreq in response to a DTWMrgMRD, any subsequent MRD to the same address must see the result of the DTWMrgMRD.
- Once DMI issues a DTRreq in response to a DTWMrgMRD, any subsequent non-coherent Read to the same address must see the result of the DTWMrgMRD.¹²

⁴¹ This allows Producer-Consumer interactions between coherent producers and non-coherent consumers

- Once DMI issues a DTRreq in response to a DTWMrgMRD, any subsequent DTW to the same address must overwrite the result of the DTWMrgMRD. The DTWMrgMRD and the DTW must also be performed in memory in the order of their reception.
- The DMI must perform non-coherent Writes to a given address in the order in which it issues the STRreqs for those Write CMDreqs. That is, the corresponding DTWreqs must be performed in the same order.
- Once DMI issues an STRreq for a non-coherent Write, any subsequent non-coherent Read to the same address received by the DMI must return the data of the Write.
- Once DMI issues an STRreq for a non-coherent Write, any subsequent non-coherent Write to the same address received by the DMI must overwrite the data of the first Write.
- Once the DMI issues an MRDrsp for an MRDFlushreq, MRDCleanreq, or MRDInvalidatereq, the effect of the said MRDreq must be visible across the system, which includes the CCMP domain as well as the home device for the address referenced in the MRDreq.

Note that:

- If two non-coherent Write CMDreqs to the same address are issued in the System sent by two distinct agents, there is no inherent order implied. So the system order gets imposed by interplay of Ncore 3 units. This order typically may be determined by the order the DMI reads them out of ingress buffer.
- A DMI may issue CMDrsp and STRrsp for Writes at the same time when the Write is read out of the ingress buffer.
- A DMI may read CMDreqs, MRDreqs, or DTWreqs out of their ingress buffers without regard to the addresses carried by them or to the identifiers of the Ncore 3 units issuing them. For example, they could be chosen based on the QoS label they carry.
- DMI might not perform address or requester collision when choosing the ingress buffer. For example, under QoS, the DMI may pull CMDreqs out of the ingress buffer based on QoS values. So requester and address ordering under QoS must be done in the AIU and use appropriate events so that DMI processes them in the desired order.
- There are no ordering requirements for DMI for messages carrying addresses that refer to distinct coherency granules.

6.3.2.2 Enforcing CMO operation semantics at DMI

Cache maintenance operations MakeInvalid, CleanShared, CleanSharedPersist and CleanInvalid refer to a cacheline that must achieve certain state in the coherency domain of the Ncore 3 platform. All three affect all the caches in the system, including the System Memory Cache (SMC).

CleanShared, CleanSharedPersist, and CleanInvalid operations result in generating MrdClean and MrdFlush messages. These messages prompt the DMI to ensure that if there is a copy of the referenced cacheline in the SMC is Cleaned or Flushed from it all the way to system memory, respectively, so that after the corresponding CMOs complete, all devices in the system have visibility to the cacheline irrespective of their access paths to its home location. The CMO semantics also require that any Writes to the cacheline already in flight from the DMI, also complete all the way to its home location. In general, there can be multiple Writes to a given cacheline in flight downstream from the DMI.

Writes are considered complete at the DMI when they receive a BRESP on DMI's immediate native interface.

If these Write are of buffered type, then the BRESP can be generated by the immediate device on the DMI's native interface receiving the Write. Such a completion carries no guarantee that the Write has completed at its home location, which is required by the CMO. Thus, such a Write completion does not satisfy the visibility requirements of CMOs.

In order to facilitate the CMO semantics, the DMI must issue every Write on AXI to system memory with AWCACHE = 0b0010, i.e., as a non-Buffered Write. This ensures that a Write is retired from the DMI only after it has completed at the DDR, thus ensuring its global visibility.

This does increase the lifetime of Writes in the DMI, and therefore may increase the size of the WTT.

Another approach is a MrdClean or MrdFlush causes a Write of null bytes (zeroed bytes and BE=0) with AWCACHE = 0b0010 is issued to the cacheline address toward the system memory and wait for it to finish. Such a Write must wait for other previous Writes (buffered or non-buffered) to receive BRESP.

6.3.3 DII consistency requirements

A DII receives CMDreq, DTWreq, DTRrsp, and STRrsp messages and issues CMDrsp, STRreq, DTWrsp, and DTRreq messages, that are relevant to consistency management in CCMP domain. In general, DII must satisfy relevant requirements associated with these messages as specified in Section 6.2. Additional requirements, if any, are specified here.

The following assumed for a DII:

- CMD and DTW messages may all reach a DII via physically *independent* hardware ports, and as such, are not inherently ordered with respect to each other.
- CMDs and DTWs sent to the DII are first written to **receive** or **ingress buffers** at the DII's boundary. Messages of multiple types may share an ingress buffer.
- The DII **receives** a message when the DII *reads* it out of its *ingress buffer*. DII receives one message per cycle.
- In DII, if two messages refer to the same coherency granule, they are treated as carrying the same address for ordering purposes.
- A DII may optionally be supplied with the size of the largest-sized endpoint device connected to it via its AXI interface. Its size must be of 2^n bytes, where $n > 5$. If so supplied, DII treats two messages with addresses with bits of address higher than n to be the same to have the *same endpoint address*. If a DII is not so supplied with endpoint device size, it assumes the size of the full address space, thereby fully serializing CMDreqs that are Endpoint ordered.

DII supports **No ordering**, **Request ordering**, and **Endpoint ordering** operations represented by CMDreqs. The type of ordering requested by a CMDreq is indicated by the OR[1:0] field carried by it (see Section 4.5.3.3.4.2).

The DII utilizes Table 4-9, the InitiatorID field comparison, and appropriate address comparison to determine the actual extent of ordering to be imposed on pairs of operations. This set of comparisons is made for a received CMDreq and all the CMDreqs that are currently active in the DII. The outcome is: No ordering, Request ordering, or Endpoint Ordering requirement between pairs of operations.

The DII must execute its actions so that the effects of DTWs, the memory Reads and Writes over the AXI interface are in concert with the system order of the Read and Write operations from native agents that may have engendered them.

Implementation Note

A DII may optionally derive a AXIID based on the address of the CMDreq for its Read and Write requests on its AXI bus. For this purpose, it can use a functional mapping of CMDreq address bits to produce an AXIID value to use with Writes.

One method uses the same number of bits out of the CMDreq address – or some hash thereof producing the same number of bits – as the number of bits in the AXIID field in the AXI bus.

For Request ordering, the chosen bits must be of higher significance than bits representing the offset with the coherency granule. For example, for an AXIID size of 6 bits, and coherency granule size of 64 bytes, the DII can choose bits 11:6 of the address.

For Endpoint ordering, the chosen bits must be of higher significance than $n-1$, where 2^n is the size of the largest endpoint device connected to the DII. For example, for an AXIID of 4 bits, and largest endpoint device size of 2^{20} bytes, the DII can choose bits 20, 21, 30, and 35 bits of the address as AXIID. The above choices of AXIID ensure that even buffered Writes on the AXI bus are appropriately ordered all the way to an endpoint device.

Note that, as described, this method is pessimistic in that it creates aliasing and orders buffered Writes from different requesters. A more sophisticated technique that utilizes InitiatorID in the computations of the AXIID could be used to alleviate the above aliasing.

Other approaches include never issuing buffered Writes on the AXI bus, converting buffered Write into unbuffered Writes, thus ensuring that a Write completes in the DII only after it completes at the endpoint. This enables the DII to choose AXIIDs for a Write so that it doesn't inadvertently bypass previously issued Writes that the said Write is expected to be strictly ordered after. This method has the disadvantage of increasing the lifetime of Writes within the DII, causing the number of buffers for holding their contexts to increase significantly.

Using a fixed AXIIDs for accesses is also a possible solution, though one that will likely deliver lower performance.

6.3.3.1 Consistency rules for DII

To achieve the required correspondence the DII must satisfy the following:

1. DII must satisfy the semantics of the OR[1:0] field in the CMDreq it receives. (See Section 4.5.3.3.4.)
2. Once DII issues an STRreq for a Read to a given address, any subsequent Read or Write to the same address that is either Request or Endpoint ordered with respect to the said Read must be performed after the said Read performs at its destination.
3. Once DII issues an STRreq for a Write to a given address, any subsequent Read or Write to the same address that is either Request or Endpoint ordered with respect to the said Write must be performed after the said Write performs at its destination.
4. Once DII issues an STRreq for a Read to a given endpoint, any subsequent Read or Write to the same endpoint that is Endpoint ordered with respect to the said Read must be performed after the said Read performs at its destination endpoint.
5. Once DII issues an STRreq for a Write to a given endpoint, any subsequent Read or Write to the same endpoint that is Endpoint ordered with respect to the said Write must be performed after the said Write performs at its destination endpoint.
6. Once the DII issues an STRrsp for a CmdClnVldreq, CmdClnInvreq, or CmdMkInvreq, the effect of the said CMDreq must be visible across the computing system, which includes the CCMP domain as well as the home device for the address referenced in the CMDreq that may be accessed by some system devices via extra-CCMP domain paths.

Note that:

- If two non-coherent Write CMDreqs to the same address are issued in the System sent by two distinct agents, there is no inherent order implied. So, the system order is imposed by interplay of Ncore 3 units. This order typically will be the order the DII reads them out of ingress buffer.
- A DII may issue CMDrsp and STRrsp for Writes at the same time when the Write is read out of the ingress buffer.
- A DII may read DTWreqs out of their ingress buffers without regard to the identifiers of the Ncore 3 units issuing them. For example, they could be chosen based on the QoS label they carry.
- DII might not perform address or requester collision when choosing a CMDreq from the ingress buffer. For example, under QoS, the DII may pull CMDreqs out of the ingress buffer based on QoS values. So, requester and address ordering under QoS must be done in the AIU and use appropriate events so that DII receives them in the desired order.
- There are no ordering requirements for DII for accesses to different endpoints. Any ordering between such accesses must be performed at the requesting AIU.

6.3.4 AIU requirements

An AIU uses consistency mechanisms and their consistency semantics defined in the current chapter to achieve the ordering of transactions requested by its native agent(s).

An AIU receives CMDrsp, DTWrsp, DTRrsp, and STRreq messages and issues CMDreq, STRrsp, DTWreq, DTRMrgMRDreq, DTRreq and DTRrsp messages, that are relevant to consistency management in CCMP domain. In general, an AIU must satisfy relevant requirements associated with these messages as specified in Section 6.2. Additional requirements, if any, are specified here.

For coherent accesses the AIU may use CMDrsp as an indication of the access having been *coherently ordered* (but not necessarily *globally observable*). Another coherent access to the same coherency granule issued after the reception of CMDrsp for the first access can be assumed to be performed after the first access.

For achieving correct global visibility order between two accesses to different addresses the AIU must issue the second CMDreq only after it has received the STRreq for the first CMDreq.⁴²

Since STRreq indicates achievement of global observability and also identifies an RBID to be used for a possible DTWreq message, the CHI AIU must issue the CompDBID response to the requesting CHI agent for its *CopyBack Write* operation (*WriteBack*, *WriteClean*, or *WriteEvict*) after receiving the STRreq message.

The CHIAIU must issue either separate DBIDResp and Comp responses or a combined CompDBID response for a *WriteUniquePtl/Full* operation after receiving the STRreq. The delivery of the separate Completion response (a separate Comp on CSRP) must NOT wait for the data to arrive from the CHI agent.

The CHI AIU must issue either separate DBIDResp and Comp responses or a combined CompDBID response for a *WriteNoSnp* with *EWA* = 1 operation after receiving the STRreq.

For *WriteNoSnp* with *EWA* = 0 operation, the CHI AIU must wait for DTWrsp message from DMI or DII, before issuing a Comp response to the requesting agent.

For ACE, ACE-Lite, ACE5-Lite and AXI based agents, the AIU must choose appropriate values of the OR[1:0] field in the CMDreq to ensure that the correct operation ordering semantics are expressed in CCMP on behalf of its native agents. (See Section 4.5.3.3.4.)

⁴² This is also useful in avoiding a potential deadlock: A pair of producers belonging to two concurrent independently running Producer-Consumer scenarios could issue a pair of Writes A, B to two home units, except in opposite order.

6.3.4.1 Times of operation completions on native interfaces

Table 6-3 shows the table of architectural events in CCMP that must occur at an AIU before it can indicate a Completion on the native interfaces.

Table 6-3: CCMP operation completion events for CHI and ACE/AXI operations

Operation Types		Completion delivered after receiving
CHI	ACE, ACE-Lite, ACE5-Lite	
Mnemonic	Mnemonic	
ReadOnce	ReadOnce	DTRreq
ReadClean	ReadClean	
ReadShared	ReadShared	
ReadUnique	ReadUnique	
ReadNotSharedDirty	ReadNotSharedDirty	
ReadNoSnp	ReadNoSnoop	
CleanUnique	CleanUnique	STRreq
CleanShared	CleanShared	STRreq
CleanSharedPersist	ALE: CleanSharedPersist	
CleanInvalid	CleanInvalid	
MakeInvalid	MakeInvalid	
MakeUnique	MakeUnique	STRreq
Evict	Evict	UPDrsp
WriteUniquePtl	WriteUnique	STRreq
WriteUniqueFull	WriteLineUnique	STRreq
(WriteCleanPtl)	NA	STRreq
WriteCleanFull	WriteClean	STRreq
WriteBackPtl	NA	STRreq
WriteBackFull	WriteBack	STRreq
WriteEvictFull	WriteEvict	STRreq
WriteNoSnpPtl/Full with EWA = 1	WriteNoSnpPtl/Full with AxCACHE[1] != 0	STRreq
WriteNoSnpPtl/Full with EWA = 0	WriteNoSnpPtl/Full with AxCACHE[1] == 0	DTWrsp
WriteUniquePtlStash	ALE: WriteUniquePtlStash	STRreq
WriteUniqueFullStash	ALE: WriteUniqueFullStash	
StashOnceShared	ALE: StashOnceShared	
StashOnceUnique	ALE: StashOnceUnique	
ReadOnceCleanInvalid	ReadOnceCleanInvalid	DTRreq
ReadOnceMakeInvalid	ReadOnceCleanInvalid	
AtomicLoad	ALE: AtomicLoad	DTRreq
AtomicStore	ALE: AtomicStore	STRreq
AtomicCompare	ALE: AtomicCompare	DTRreq
AtomicSwap	ALE: AtomicSwap	DTRreq
PrefetchTgt	NA	STRreq
DvmOp (Non-sync)	DVM Message (Non-sync)	CMPrsp
DvmOp (Sync)	DVM Message (Sync)	CMPrsp

6.3.4.2 Informing DCE of completions of operations at the agent

6.3.4.3 Special handling of selected operations at the AIU

6.3.4.3.1 CopyBack Write operations from ACE agent and Proxy Cache

The CCMP message sequence for performing a WriteBackPtl/Full or WriteEvict operation from an ACE agent issued with AWDOMAIN[1:0] = 0b01 or 0b10 is shown in Figure 6-1. In this sequence, only after the AIU receives DTWrsp from the DMI, may it issue a UPDreq to the DCE. The DCE responds thereafter with a UPDrsp. Only after the UPDrsp from the DCE may the AIU issue a BRESP to the ACE agent. The WACK response from the ACE agent completes the operation at the ACE AIU.

When WriteCleanPtl/Full operations are issued from an ACE agent, with AWDOMAIN[1:0] == 0b01 or 0b10, the same flow as Figure 6-1 is employed, except, no UPDreq/UPDrsp messages are invoked.

A Proxy Cache also employs the same sequence for its WriteBackPtl/Full and WriteEvict operations. The AIU employs internal microarchitecture means to indicate to the Proxy Cache that its such operation is globally visible.

When WriteBackPtl/Full and WriteCleanPtl/Full operations are issued to non-coherent storage, with AWDOMAIN[1:0] == 0b00 or 0b11, the same flow as Figure 6-1 is employed, except, no UPDreq/UPDrsp messages are invoked.

Note that a WriteEvict operation with AWDOMAIN[1:0] == 0b00 or 0b11 is not forwarded to DMI or DII.

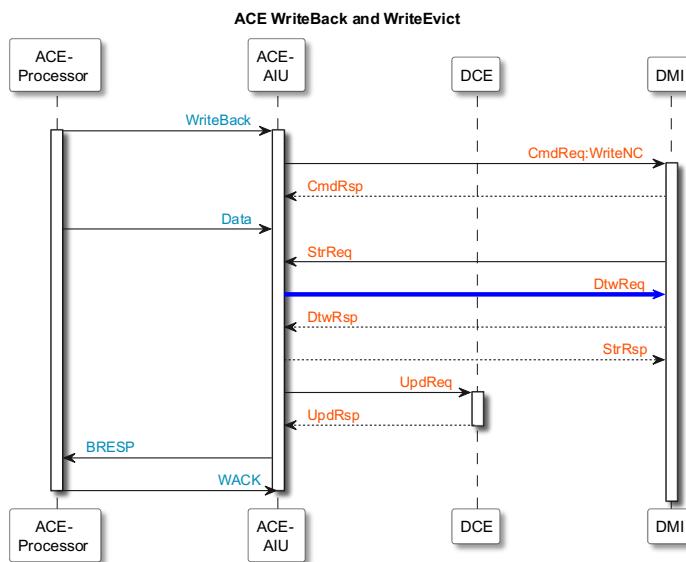


Figure 6-1: CCMP Message Flow - ACE WriteBack and WriteEvict operations AWDOMAIN[1:0] = 0b01 or 0b10

Chapter 7: DVM Op Processing

This chapter deals with processing of **Distributed Virtual Memory Operation (DVMOp)** processing in CCMP. These operations are primarily used for software management of **Translation Look-aside buffers (TLBs)** that are part of the **Memory Management Units (MMUs)** inside processors. But they may also be used to manage Instruction caches and Branch Prediction Tables inside processors.

7.1 Introduction

DVMOps are primarily used for software management of *Translation Look-aside buffers* that hold translations from **Virtual addresses** are used to disambiguate potentially **aliased** addresses generated by programs to **Real Addresses** that are issued by agents into the CCMP domain via their native interface. *Real addresses* are used to access system storage. *Real addresses* are also the addresses used in management of coherency.

TLBs are caches inside processors that hold copies of **Page Table Entries (PTEs)**, set up in system memory by an Operating System or a Hypervisor, that specify translations between address types.

Peripheral devices also use the same PTEs. A hardware block equivalent to the MMU inside the processor and called **IOMMU** or **System MMU (SMMU)** is used to authenticate accesses made by peripheral devices and translate addresses referenced within them into Real addresses. Storage accesses made by peripheral devices are sent via the SMMU to receive the translation. These translated accesses then enter the CCMP domain via an ACE-Lite or ACE- Lite-E IOAIU.

7.2 CCMP Message Flows for CHI DVMOps

Overall, the CCMP message flow closely parallels the DVM transaction flows that occur in CHI architecture. The contents of messages also mirror those of CHI transaction contents. The details of the CHI and ACE DVM request transactions and the corresponding snoops can be found in respective ARM architectures.

In response to the reception of a DVMOp transaction from a CHI agent the AIU issues a DVMOp CMDreq to the DVE. The ensuing message flow is similar to that for a WrNC CMDreq.

The DVE constitutes a pair of correlated snoop messages (SNPreqs) from each DVMOp CMDreq it receives, per CHI architecture. Two SNPreqs per DVMOp are created both for Non-Sync DVMOp and Sync DVMOp.

The two SNPreqs are sent by the DVE in identical sequence to all the **DVM-coherent agents**⁴³ in the system. The order of their transmission to snoopers by the DVE adheres to the order prescribed by the CHI architecture.

The AIU receiving these snoops present them to their respective agents in a native protocol- appropriate manner and format.

Detailed look at the message flows appears in the sections below.

Specification of contents of the DVM messages and snoops appears in Section 7.3.

⁴³ DVM Coherent agents are those agents which hold address translation tables and that must participate in the TLB management protocol via DVM snoop messages

7.2.1 CCMP Request Message Flow for CHI Requester for non-Sync or Sync DVMOp

Figure 7-1 displays the CCMP message flow for a DVM Sync or non-Sync Request issued by a CHI processor agent.

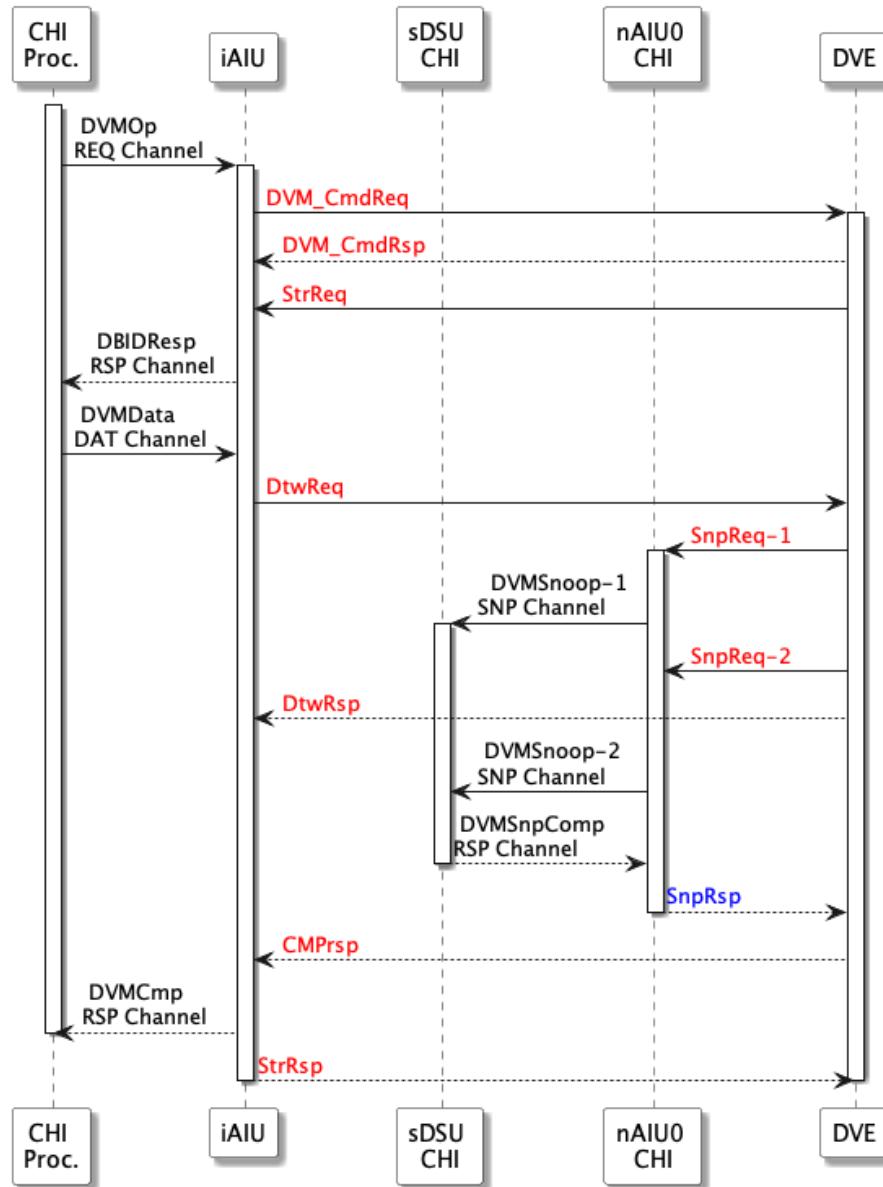


Figure 7-1: CCMP Message Flow for CHI Requester for A DVM Sync or non-Sync by A CHI Processor Agent

7.2.2 CCMP Request Message Flow for ACE Requester for non-Sync DVMOp

Figure 7-2 displays the CCMP message flow for non-Sync DVMOp Request issued by an ACE processor agent.

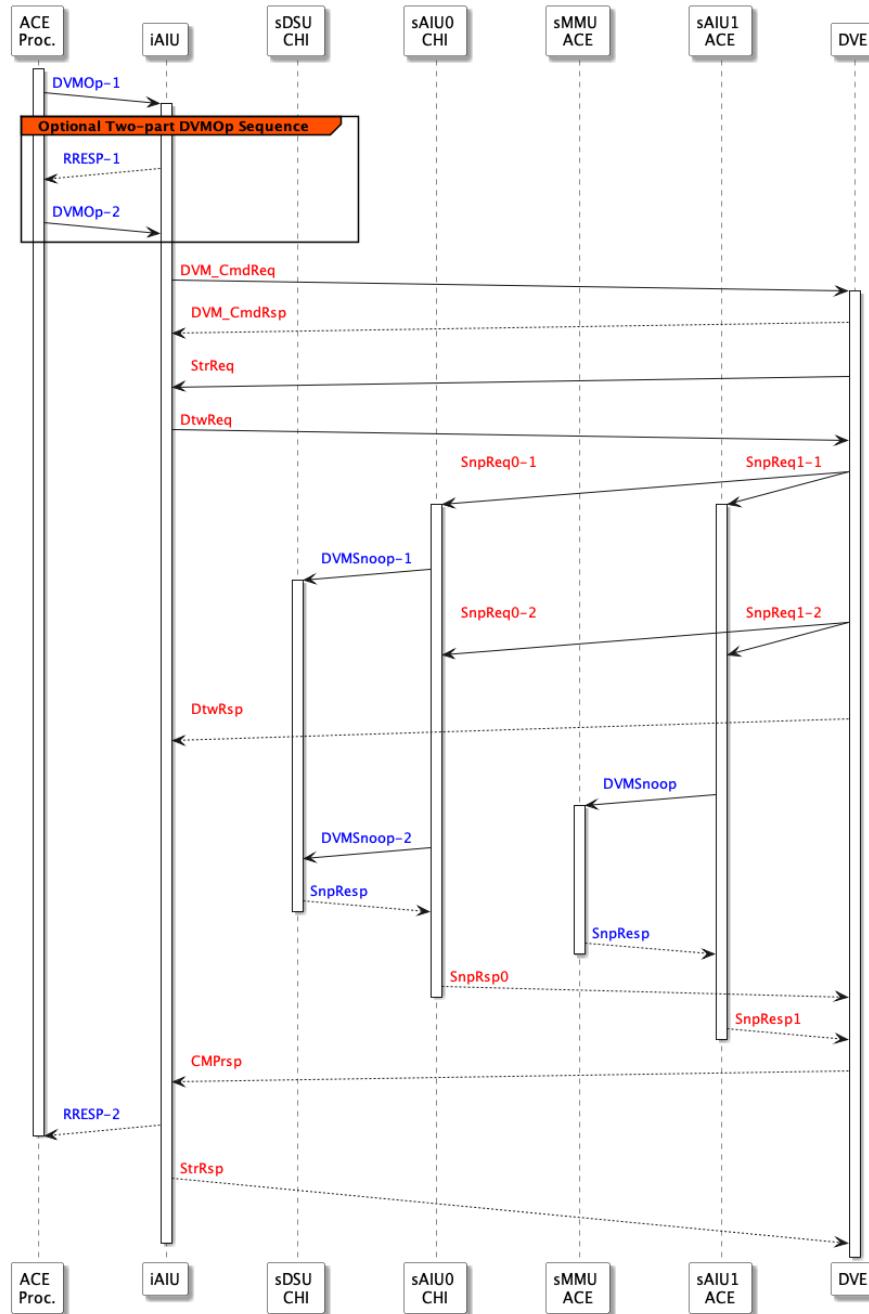


Figure 7-2: CCMP Message Flow for A DVM Non-Sync Transaction Issued from An ACE Requester

7.2.3 CCMP Request Message Flow for ACE Requester for Sync DVMOp

Figure 7-3 displays the CCMP message flow for a DVM Sync Request issued by an ACE processor agent.

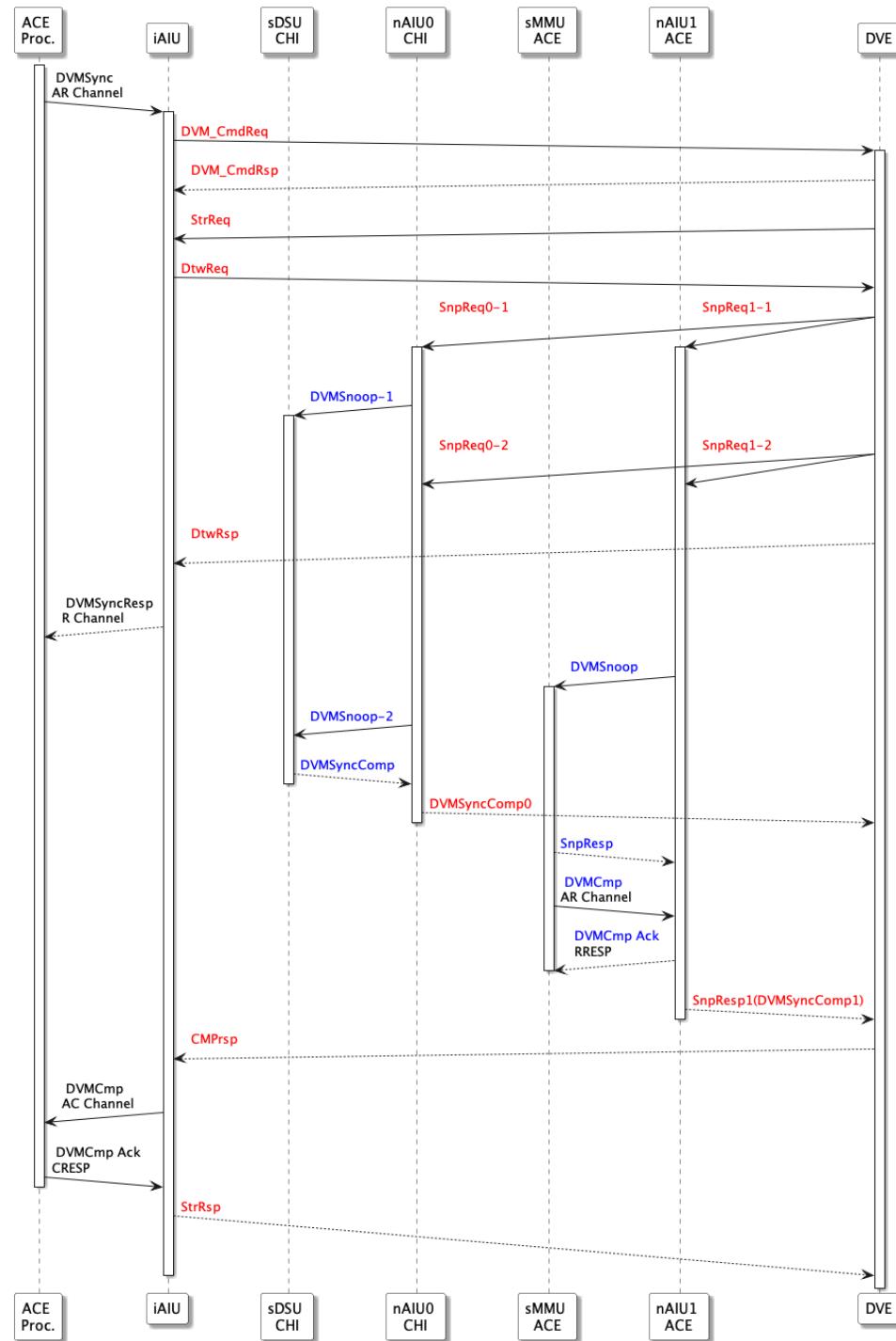


Figure 7-3: CCMP Message Flow for A DVM Sync Transaction Issued from An ACE Requestor

7.2.4 Sync or non-Sync DVMOp Snoop Process for a CHI snooper

Figure 7-4 shows the CCMP message flow for a DVM non-Sync snoop issued to a CHI snooper.

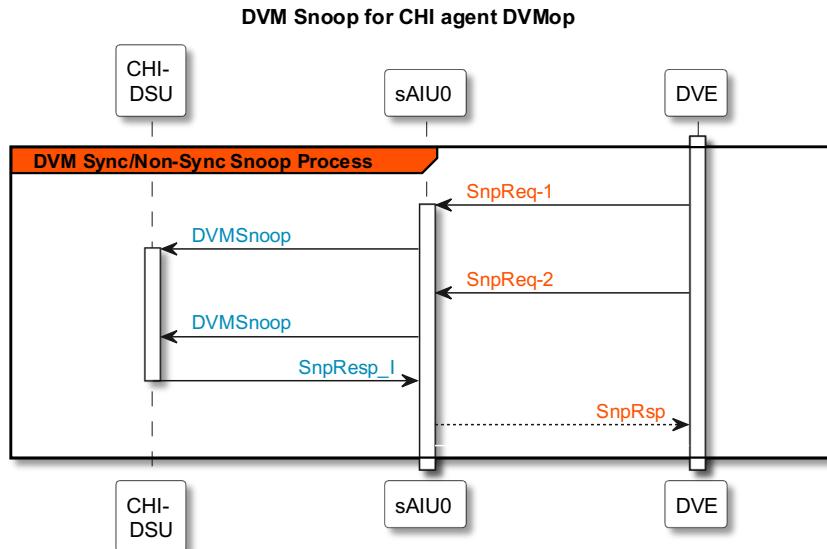


Figure 7-4: CCMP Message Flow for A DVM Non-sync Snoop Issued to A CHI Snooper

7.2.5 Non-Sync DVMOp Snoop Process for an ACE or ACE-Lite snooper

Figure 7-5 displays the CCMP message flow for a DVM non-Sync snoop issued to an ACE processor or ACE-Lite DVM snooper. It is the snoop part of the complete flow as shown in Figure 7-2.

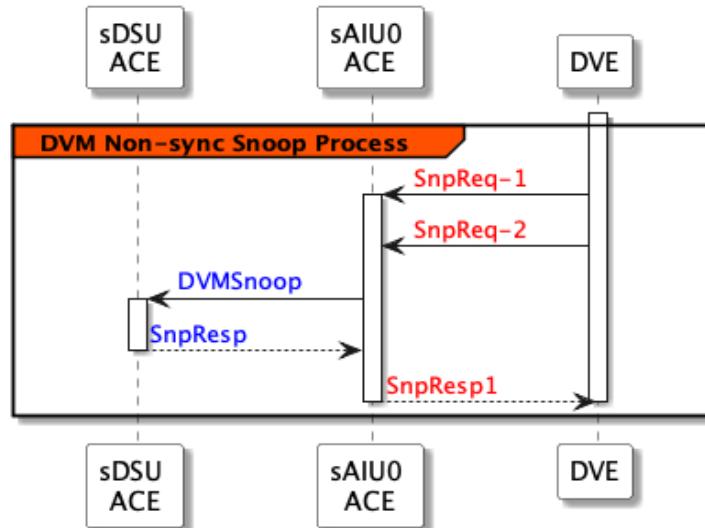


Figure 7-5: CCMP Message Flow for A DVM Non-sync Snoop Issued to An ACE Snooper

7.2.6 Sync DVMOp Snoop Process for an ACE or ACE-Lite Snooper

Figure 7-6 displays the CCMP snoop message flow for a DVM Sync snoop issued to an ACE processor or ACE-Lite DVM snooper. This is the snoop part of the complete flow as shown in Figure 7-3.

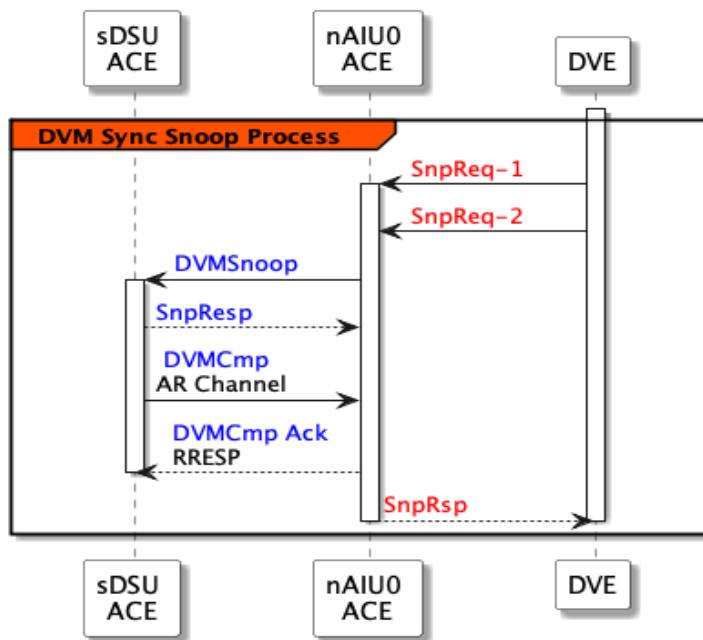


Figure 7-6: CCMP Message Flow for A DVM Sync Snoop Issued to An ACE Snooper

Note that the ACE AIU receives two-part snoop messages from DVE and constitute a CCMP compliant DVMOp CMDreq and the DTWreq message to go along with it, which is only shown in Figure 7-3.

The ACE AIU must be capable of receiving two SNPreq messages per DVMOp CMDreq. The AIU must have enough resources to support at least one pair of SNPreq messages. It may support outstanding snoops from multiple outstanding DVMOps at a time. Based on the type of agnet, the number of DVMSpns sent to the ACE snooper may only be one for an ACE agent as shown in Figure 7-6.

For an ACE agent, only one SnpResp message is expected. Also for an ACE snooper, it is expected it will send DVM completion message once ALL outstanding non-sync messages received before this DVM sync message have been COMPLETED. The ACE agent (nAIU0), which is directly connected to this ACE snooper, is expected to immediately acknowledges the completion via the R channel.

The ACE agent (nAIU0) is also responsible for sending the SnpRsp message back to the DVE to signal the completion of the DVM Sync, which triggers the completion of the entire DVM sync flow as shown in Figure 7-3.

7.3 DVM Message Contents

This section specifies the formatting of information in the CCMP DVM messages. In general, it closely follows the CHI specification.

7.3.1 DVMOp CMDreq Message for CHI and ACE

DVMOp requests on native CHI and ACE interfaces are sent in two parts:

- A CMDreq message with an address field carrying metadata to describe the target virtual address
- followed by the associated data (DTWreq) message with a data payload of 64 bits (1 doubleword) providing additional information

Table 7-1 and Table 7-2 show the contents of the Addr[] and data fields mapped from the arriving CHI DVM transactions.

Table 7-1: Format of Addr[] field of CMDreq for arriving CHI or ACE transactions

CCMP Address Field		CHI Command Address		ACE Command Address	
Addr[]	Contents	Addr[]	Contents	Addr[]	Contents
0	Reserved (0b0)	0	Reserved (0b0)	-	Reserved (0b0)
1	Reserved (0b0)	1	Reserved (0b0)	-	Reserved (0b0)
2	Reserved (0b0)	2	Reserved (0b0)	-	Reserved (0b0)
3	0b0	3	0b0	1	0b0
4	VA valid	4	VA valid	0	Single/two part (require address or not)
5	VMID Valid	5	VMID Valid	6	VMID Valid
6	ASID Valid	6	ASID Valid	5	ASID Valid
8 : 7	Security	8 : 7	Security	9 : 8	Security
10 : 9	Exception Level	10 : 9	Exception Level	11 : 10	Exception Level
13 : 11	DVMOp Type	13 : 11	DVMOp Type	14 : 12	DVMOp Type
21 : 14	VMID[7:0] (ACE:VA[27:20])	21 : 14	VMID[7:0]	31 : 24	VMID[7 : 0] or Virtual Index [27 : 20]
37 : 22	ASID[15:0] (ACE:VA[19:12])	37 : 22	ASID[15:0]	39 : 32	ASID[15 : 8]
				23 : 16	ASID[7 : 0] or Virtual Index[19 : 12]
39 : 38	Staged Invalidation (S2, S1)	39 : 38	Staged Invalidation (S2, S1)	3 : 2	Staged Invalidation (S2, S1)
40	Leaf Entry Invalidation	40	Leaf Entry Invalidation	4	Leaf Entry Invalidation
41	Range	41	Range	7	Range
42	Num[4] (for CHI agent)	42	Num[4]	-	-
Max:43	Reserved (0)	Max:43	Reserved (0)	Max:48	Reserved (0)

Table 7-2: Format of Data[] field of CMDreq for arriving CHI and ACE transactions

CCMP Dtw Data Field		CHI Dtw Data Field		ACE Dtw Field		
Data[]	Content	Data[]	Content	Data[]	Content	
					Phase 1	Phase 2
0	Num[0]	0	Num[0]	0		Num[0]
1	Num[1]	1	Num[1]	1		Num[1]
2	Num[2]	2	Num[2]	2		Num[2]
3	Num[3] (PA[4]/VA[4] for ACE agent)	3	Num[3]	4		Num[3]/ PA[4]/VA[4]
5:4	Scale[1:0]/VA[7:6]/PA[7:6]	5:4	Scale[1:0]/VA[7:6]/PA[7:6]	7:6		Scale[1:0]/VA[7:6]/PA[7:6]
7:6	TTL[1:0]/VA[9:8]/PA[9:8]	7:6	TTL[1:0]/VA[9:8]/PA[9:8]	9:8		TTL[1:0]/VA[9:8]/PA[9:8]
9:8	TG[1:0]/VA[11:10]/PA[11:10]	9:8	TG[1:0]/VA[11:10]/PA[11:10]	11:10		
37:10	VA[39:12]/PA[39:12]	37:10	VA[39:12]/PA[39:12]	39:12		VA[39:12]/PA[39:12]
38	VA[40]/PA[40]	38	VA[40]/PA[40]	40		If PICI: PA[40]
				3		Otherwise: VA[40]
42:39	VA[44:41]/PA[44:41]	42:39	VA[44:41]/PA[44:41]	44:41		If PICI: PA[44:41]
				43:40		Otherwise: VA[44:41]
46:43	VA[48:45]/PA[48:45]	46:43	VA[48:45]/PA[48:45]	1'b0, 47:45		If PICI: 1'b0, PA[47:45]
				43:40	Otherwise: VA[48:45]	
50:47	VA[52:49]/PA[52:49]	50:47	VA[52:49]/PA[52:49]	47:44		VA[52:49]
54:51	VA[56:53] for ACE agent	54:51	Reserved 0	47:44	VA[56:53]	
55	Num[4]/PA[5]/VA[5] for ACE agent	55	Reserved 0	5		Num[4]/PA[5]/VA[5]
59:56	VMID[11:8]	59:56	VMID[11:8]	AxVMIDEXT[3:0]		VMID[11:8]
63:60	VMID[15:12]	63:60	VMID[15:12]	If one part command: AxADDR[43:40]		If two part command: AxVMIDEXT[3:0]

Note:
The ACE Dtw Fields are derived from the first and second addresses submitted with the two requests (Phase 1 and 2)
PICI: for ACE agent Physical Instruction Cache Invalidate, will contain PA information in 2nd command and VA in 1st command

7.3.2 DVMOp CMDreq Message for ACE

A DVMOp request is sent in two parts: A CMDreq message followed by the associated DTWreq message with a data payload of 64 bits (1 doubleword).

The totality of the DVMOp-specific information is carried in two parts, the Addr[] field of a CMDreq message and the Data[] field of the DTWreq message.

shows the contents of the Addr[] field of the CMDreq and Data[] field of the DTWreq by bit positions.

Architecture Note

For ACE, bits [5 : 4] of the Physical/Virtual Address, these bits are not conveyed by the DVE in the corresponding SNPreq to CHI AIU. They must be set to 0b00 by DCE in the DVMSnp transaction to the CHI AIU. Also see Section 7.3.3.3.1.

CHI does not support Virtual Address bits [56:53]. Thus, for ACE, bits [56:53] of the Virtual Address are not conveyed by the DVE in the corresponding SNPreq to CHI AIU. Also see Section 7.3.3.3.1.

For ACE, if DVM message is Physical Instruction Cache Invalidate, bit[31:16] of 1st phase = VA[27:20], and bit[47:4] of 2nd phase = PA[47:4].

7.3.3 CCMP DVM SNPreq Messages

A DVMOp request is converted to a pair of SNPreq messages by the DVE.

7.3.3.1 DVMOp Identifier

The DVE assigns a unique identifier for each DVMOp CMDreq for which it has issued SNPreq and that has not yet completed. The DVE supplies this identifier in the **MPF2** field of each of the two SNPreq messages it generates for a given DVMOp.

7.3.3.2 DVM SNPreq Identifier

The two SNPreq for a DVMOp are ordered and by as such identified by the DVE as **first** and **second** DVMOp snoops. The first SNPreq for given DVMOp is identified by number 1 and the second is identified by number 2. These numbers are supplied in **MPF3** fields of the corresponding SNPreq messages.

7.3.3.3 Contents of the Addr[] fields for the two SNPreq messages

[Table 7-3](#) to [Table 7-6](#) show the contents of the Addr[] field of the two CCMP SNPreq messages sent to CHI or ACE target AIUs by bit positions and the respective mapping to the native protocol transactions.

Table 7-3: Format of the first DVM SNPReq message

CCMP DVM Snoop 1		CHI Target	ACE Target		Comment
Bit Position	SNPreq 1 Addr[] Field Contents	Addr Bit Position	Snoop 1 Addr[]	Snoop 2 Addr[]	
0	Reserved (0)		1	---	Reserved
1	Reserved (0)		15	---	Completion (depending on message type)
2	Reserved (0)		---	---	
3	0b0	0	---	---	Snoop Sequence ID
4	VA Valid	1	0	---	Single/Two part
5	VMID Valid	2	6	---	VMID valid
6	ASID Valid	3	5	---	ASID valid
8:7	Security	5:4	9:8	---	Security
10:9	Exception Level	7:6	11:10	---	Guest OS or Hypervisor
13:11	DVMOp Type	10:8	14:12	---	DVMOp Type
21:14	VMID[7:0] or Virtual Index: VA[27:20]	18:11	31:24	---	VMID[7:0] or Virtual Index[27:20]
37:22	ASID[15:0] or Virtual Index: {0x00 VA[19:12]}	34:19	23:16	---	ASID[7:0] or Virtual Index [19:12]
			39:32	---	ASID [15:8]
39:38	Staged Invalidation (S2, S1)	36:35	3:2	---	Staged Invalidation (S2, S1)
40	Leaf Entry Invalidation	37	4	---	Leaf Entry Invalidation
43:41	VA[48:46]	40:38	43:41	---	VA[48:46] if two part snoop
44	VA[50]	41	---	45	If not PICI : VA[50]
45	VA[52]	42	---	47	If not PICI : VA[52]
51:46	Reserved (0)	48:43	---	---	

Table 7-3: Format of the first DVM SNPReq message

CCMP DVM Snoop 1		CHI Target	ACE Target		Comment	
Bit Position	SNPreq 1 Addr[] Field Contents	Addr Bit Position	Snoop 1 Addr[]	Snoop 2 Addr[]		
Max:52	Reserved (0)		Max:49	Max:48		Reserved (0)

Table 7-4: Format of the first DVM SNPReq message (MPF)

CCMP DVM Snoop 1		CHI Target	ACE Target		Comment
Bit Position	SNPreq 1 MPF Field Contents	Addr Bit Position	Snoop 1 Addr[]	Snoop 2 Addr[]	
MPF1 [7:0]	VMID[15:8]	VMIDExt[7:0]	AxVMIDEXT[3:0]	---	VMID[11:8]
			43:40	AxVMIDEXT[3:0]	VMID[15:12] If one part snoop its on AxADDR 1 st If two part snoop its on AxVMIDEXT 2 nd
MPF2	DVE TTID	---	---	---	DVE TTID
MPF3[0]	0 (snoop number)	---	---	---	---
MPF3[1]	range	FwdNID[0]	7	---	Range

Table 7-5: Format of the second DVM SNPReq message

CCMP DVM Snoop 2		CHI Target	ACE Target		Comment
Bit Position	SNPreq 2 Addr[] Field Contents	Addr Bit Position	Snoop 1 Addr[]	Snoop 2 Addr[]	
0	Reserved (0)	---	---	---	
1	Reserved (0)	---	---	---	
2	Reserved (0)	---	---	---	
3	0b1	0	---	---	Snoop Sequence ID
5:4	Scale[1:0]/VA[7:6]/PA[7:6]	2:1	---	7:6	Scale[1:0]/VA[7:6]/PA[7:6]
7:6	TTL[1:0]/VA[9:8]/PA[9:8]	4:3	---	9:8	TTL[1:0]/VA[9:8]/PA[9:8]
9:8	TG[1:0]/VA[11:10]/PA[11:10]	6:5	---	11:10	TG[1:0]/VA[11:10]/PA[11:10]
37:10	VA[39:12]/PA[39:12]	34:7	---	39:12	VA[39:12]/PA[39:12]
38	VA[40]/PA[40]	35	---	40	If PICI : PA[40]
				3	If not PICI : VA[40]
42:39	VA[44:41]/PA[44:41]	39:36	---	44:41	If PICI : PA[44:41]
				43:40	If not PICI : VA[44:41]
43	VA[45]/PA[45]	40	---	45	If PICI : PA[45]
				40	If not PICI : VA[45] And if two part snoop
44	VA[49]/PA[46]	41	---	46	If PICI : PA[46]
				44	If not PICI : VA[49]
45	VA[51]/PA[47]	42	---	47	If PICI : PA[47]
				46	If not PICI : VA[51]
49:46	PA[51:48]	46:43	---	---	
51:50	Reserved (0)	48:47	---	---	
Max : 52	Reserved (0)	Max : 49	Max : 48		Reserved (0)

Table 7-6: Format of the second DVM SNPReq message (MPF)

CCMP DVM Snoop 2		CHI Target	ACE Target		Comment
Bit Position	SNPreq 2 MPF Field Contents	Addr Bit Position	Snoop 1 Addr[]	Snoop 2 Addr[]	
MPF1	CHI : 0 ACE :{4'b0, VA[56 : 53]}	---	47 : 44	---	VA[56 : 53]
MPF2	DVE TTID	---	---	---	DVE TTID
MPF3[0]	1 (snoop number)	---	---	---	---
MPF3[1]	Num[0]	FwdNID[0]	---	0	Num[0]
MPF3[2]	Num[1]	FwdNID[1]	---	1	Num[1]
MPF3[3]	Num[2]	FwdNID[2]	---	2	Num[2]
MPF3[4]	Num[3](PA[4]/VA[4] for ACE)	FwdNID[3]	---	4	Num[3](PA[4]/VA[4] for ACE)
MPF3[5]	Num[4](PA[5]/VA[5] for ACE)	FwdNID[4]	---	5	Num[4](PA[5]/VA[5] for ACE)

7.3.3.3.1 Special handling of DVMSnp on ACE or ACE-Lite interfaces

Following special considerations are required in converting CCMP SNPreq into DVMSnp on the AC channel of ACE or ACE-Lite interface:

- Bit[15] of the first DVMSnp is derived from DVM opcode
 - 1'b0: TLB Invalidiation, Branch Predictor Invalidation, Instruction Cache Invalidation, Hint
 - 1'b1: Synchronization
- Depending on if the DVM message is PICI(Physical Instruction Cache Invalidate) or not, bit[47:40] of second DVMSnp has different meaning, and the values are from different bit field of first and second SNPreq.

7.3.3.3.2 Special handling of DVMSnp on CHI interfaces

The **MPF1** field of the SNPreq carries the VMIDExt[] field that is supplied with the CHI DVMSnp requests.

The **MPF3** field of SNPreq carries the FwdNID[] field that is supplied with the CHI DVMSnp requests.

- If **Range** bit is not set, Num[] field is not applicable and must be set to 0

The CHIAIU must extract these fields and supply that with snoop request on its native interface. Also see Table 4-31.

7.3.3.3.3 Special handling of DVMSnp in DVE

DVM messages of the following fields from different type of AIU requires DVE to take special treatment when generating SNPreq.

- VA[5:4]/PA[5:4] is only applicable in ACE DVM messages, DVE needs to pass this information as is to any ACE/CHI agent(s) that is(are) involved in DVM operations. Num[] field is set to 0 inside a CAIU agent as described in section 7.3.3.3.2.
- Num[4:3]/VA[5:4]/PA[5:4] field of DVM messages are from bit[3] of CCMP DTWreq and bit[55] of CCMP DTWreq if from ACE agent or bit[42] of CCMP CMDreq if from CHI agent, the information can be directly passed on **MPF3[5:4]** of second SNPreq to ACE AIU.
- VA[56:53] is not supported in CHI DVM, therefore **MPF1** field of second SNPreq are 0 if the DVM message is from CHI AIU.

Chapter 8: Data Transfers

This chapter specifies the data transfers supported by Concerto-C Messaging Protocol.

8.1 Preliminaries and Definitions

Definitions:

- **nByteSize** = Size of transfer in number of bytes = 2^{Size}
- **nDWRIntf** = Size of Requester native data interface in number of DWs = 2^{IntfSize}
- **nBytesRIntf** = Size of Requester native data interface in number of Bytes. = $8 * nDWRIntf$
- **nBeatsRIntf** = Number of beats on data on the native interface = Ceil (Size / nBytesRIntf)

An Ncore 3 unit transmits or receives data as part of a Data Message (DM). DTRreq and DTWreq (including the DTWMrgMRD) are the two types of DM in CCMP. A CTF network port of an Ncore 3 unit is equipped to handle DMs by providing a specific field labeled **Data** (sometimes referred to as the port's **Data bus**).

The width of this *Data bus* in Ncore 3 is restricted to be 2^n Double Words (DWs), or 2^{n+3} bytes, where $n = 0, 1, 2$, or 3 . Certain CCMP messages carry this information in the field called *IntfSize* (see Section 4.5.3.7).

The actual amount of data accessed by or accompanying a message is defined by the field,

Size (see Section 4.5.3.6 for example), which is equal to 2^m bytes, where $m = 0, 1, 2, 3, \dots$, or 6 . (See Chapter 2.)

The ports of an Ncore 3 unit are expected to be **clocked**. The quantum of bytes physically transmitted or received by an Ncore 3 unit per clock cycle on any of its network DM ports equals the width of the *data bus* of that port. This quantum is termed a **beat** of data.

If *Size* \leq size of the port's data bus, only a single data *beat* of transfer is required. If *Size* $<$ size of the port's data bus, only some of the bytes of the data *beat* will be valid. If *Size* $>$ size of the port's data bus, multiple beats of data transfer are needed.

For DTWreq messages, valid bytes are required to be identified by asserting the corresponding *Byte Enable (BE)* bits accompanying the data (see Section 4.8.4.5.2), while the assertion of BEs for DTRreq messages is Ncore 3 platform implementation specific (see Section 4.7.5.3.2).

Implementation Note

Ncore 3 Requirement: In Ncore 3, the target units DMI and DII are required to transfer **Max{Size, IntfSize}** worth of data in **DTRreq** messages to the receiving AIU, where *Size* refers to the *Size* either indicated in, or implied by, the CMDreq⁴⁴ in connection with which the DTRreq is being sent, and *IntfSize* refers to the native data bus size of the AIU receiving the DTRreq (see Section 4.5.3.7).

⁴⁴ Read CMDreqs directly indicate the Size of access. Stashing CMDreq imply a CG worth data transfer to the

8.2 Data Alignment on Buses

As stated in Section 8.1, data buses supported in CCMP are of the size 2^{n+3} bytes wide, where $n = 0, 1, 2$, and 3 . The wires of these data buses form ordered **byte lanes** over which data can be transferred. The byte lanes of a bus are labeled right to left in **LittleEndian** order – The rightmost byte is labeled 0 , the next adjacent is labeled 1 , the next adjacent to 1 is labeled 2 , and so on, all the way to $2^{n+3}-1$.

The labels of the byte lanes also designate the significance of the data bytes those lanes are expected to carry. Thus, byte lane 0 carries the least significant byte of the 2^{n+3} bytes the bus carries, lane 1 carries the next significant byte, and so on.

Bytes in storage are identified by their addresses. When being transferred over CCMP data buses, the data contained in these bytes of storage must be placed on bytes lanes of buses based on the addresses of the storage bytes. For a data bus with 2^b bytes, the least significant b bits of the address of the byte identify the lane on which the data byte must travel. This is referred to as **natural alignment** of data bytes over byte lanes of the bus. CCMP supports only *natural alignment* of bytes for data transfers.

CCMP only supports 2^{Size} bytes of *aligned accesses*, where $\text{Size} = 0, 1, 2, 3$, etc., up to a CG. (See Chapter 2.)

For a transfer of 2^{Size} bytes of data over a data bus of 2^b bytes where $\text{Size} = b$ requires a single beat of data transfer.

For a transfer of 2^{Size} bytes of data over a data bus of 2^b bytes, where $\text{Size} > b$, $2^{(\text{Size} - b)}$ number of beats of data transfers are necessary.

- Of the least significant Size number of bits of address, the most significant $(\text{Size} - b)$ bits form the **beat identifier or beat index (BI)**. A transfer of 2^{Size} bytes of data over a bus of size 2^b , with $\text{Size} > b$, beats with BI of 0 through $2^{(\text{Size} - b)}-1$ are transferred. All of the bytes transferred have the same address bits above their least significant Size number of bits as the address of the access itself.

CCMP requires that the first beat of a multi-beat data transfer carry the byte of data whose address is indicated in the access. This beat is referred to as the **critical beat**. The BI for the critical beat is referred to as **CBI**.

The remaining beats are carried in a **Wrap** order. Successive beats of data transfers carry 2^b bytes of successively higher addresses and thus significance, up to the beat with $BI = 2^{(\text{Size} - b)}-1$. If more bytes need to be transferred, the next beat is the one with $BI = 0$, with an address equal to the address of the access except with least significant Size number bits being 0 and continues through the beat with $BI = CBI - 1$.

For a transfer of 2^{Size} bytes of data over a data bus of 2^b bytes, where $\text{Size} < b$, also requires a single beat of data transfer.

In such a transfer, not all bytes are valid. Byte enables (BEs) associated with respective byte lanes of a bus may be used to identify valid bytes. Valid bytes may be explicitly identified by asserting the corresponding BEs. On the other hand, if the receiver knows which bytes are valid in the transfer, assertion of BEs may not be necessary.

Implementation Note

In certain implementation, the actual transfer of data may be $2^{\text{Size}'}$, where $\text{Size}' > \text{Size}$. In such a delivery, the requested data must be delivered to the designated receiver before additional padding bytes are delivered.

In such circumstances, the receiver is required to extract the requested data from the transfer and discard the padding bytes.

8.3 Data Identification

When data is transferred in CCMP it is accompanied by auxiliary information on a per-doubleword basis. This auxiliary information is described in Section 4.7.5.3.2 through Section 4.7.5.3.6 for DTRreq messages and Section 4.8.4.5.2 through Section 4.8.4.5.6 for DTWreq messages. The DWID field identifies the doubleword within the coherency granule referenced by the CMDreq message which may have resulted in the data message.

8.3.1 DWID identification

The DWID fields within a CCMP message are assigned values by the unit that initiates the message. These values are derived from the explicit *Data Identifier (DataID)* fields supplied over the native interface as in the case of CHI, or as implied by data formatting rules specified by the native interface such as AXI.

8.3.2 DataID identification

Similarly, when data from CCMP messages is transferred to native interfaces, the DataID fields must be derived from the actual DWID field values carried in the CCMP message.

8.3.3 CCID identification

In CHI, a data message also carries the *Critical Chunk Identifier (CCID)*. Unlike the DataID fields, the CCID field is derived from the address referenced by the original CHI Request transaction.

8.4 Data transfer formats

Data is sent inside Concerto-C domain in WRAP order. The amount of data is dictated by the Size field in the CMDreq – which is some power of 2 number of bytes.

The number of “beats” of data is in reference to the requester’s native bus width. It is also a number that is some power of 2, given by Ceil (Size / IntfSize), where IntfSize is also the field in the CMDreq.

The CMDreq could be of a Read or Write type.

8.4.1 DTR requests

The critical DW is delivered by the AIU over its native interface followed by remaining data in WRAP order – up to the number of beats necessary to satisfy the CMDreq. For Reads data must be delivered to the requester AIU such that there is zero cycle waiting because of out of order delivery of bytes.

Let the beat at the requester's native data bus be called “**Requester Bus Word**” or “**RBW**.” Note that RBW is also some power of 2 DWs in size.

Thus the data delivered on the native data bus is: RBW0, RBW1, RBW2, ..., as necessary, such that RBW0 contains the critical DW and RBW0, RBW1, RBW2, etc. contain data in the correct WARP order.

DWs are placed within the RBWs in little endian order per their significance by address: Thus $\text{RBW}_{<i>} = \text{RBW}_{<i>.\text{DW-highest}} | \dots | \text{RBW}_{<i>.\text{DW-lowest}}$.

For transmission within Concerto-C, these DWs are labeled as **RDWs** in ascending order starting with the RBW0 as follows. Diagram 1 shows RBWs and RDWs for a data bus with $n_{\text{DWRIntf}} = 2$. RBW0 is transmitted first followed by RBW1, RBW2, and RBW3.

Diagram 1:

$$\text{RBW}_0 = \text{RDW}_1 | \text{RDW}_0$$

$$\text{RBW}_1 = \text{RDW}_3 | \text{RDW}_2$$

$$\text{RBW}_2 = \text{RDW}_5 | \text{RDW}_4$$

$$\text{RBW}_3 = \text{RDW}_7 | \text{RDW}_6$$

Let the DWs within a cacheline labeled as **CDWs**.

Consider a cacheline Read CMDreq with critical double word of **CDW2**.

The transmission of that accessed data on the native requester bus with $n_{\text{DWRIntf}} = 2$. The data must flow in the following order

Diagram 2:

First cycle: CDW3 CDW2

Second cycle: CDW5 CDW4

Third cycle: CDW7 CDW6

Fourth cycle: CDW1 CDW0

Diagram 3 combines Diagrams 1 and 2 and shows the correspondence of RDWs and CDWs on the native bus for the same Read access.

Diagram 3:

RBW0 =	RDW1 RDW0	which should contain:	CDW3 CDW2
--------	-------------	-----------------------	-------------

RBW1 =	RDW3 RDW2	which should contain:	CDW5 CDW4
--------	-------------	-----------------------	-------------

RBW2 =	RDW5 RDW4	which should contain:	CDW7 CDW6
--------	-------------	-----------------------	-------------

RBW3 =	RDW7 RDW6	which should contain:	CDW1 CDW0
--------	-------------	-----------------------	-------------

If the native requester bus were a 256-bit bus, for the same Read CMDreq, the RDW and CDW data labeling is be as follows:

Diagram 4:

$$\begin{array}{ll} \text{RBW0} & = \text{RDW3} | \text{RDW2} | \text{RDW1} | \text{RDW0} \\ \text{RBW1} & = \text{RDW7} | \text{RDW6} | \text{RDW5} | \text{RDW4} \end{array} \quad \begin{array}{ll} = \text{CDW3} | \text{CDW2} | \text{CDW1} | \text{CDW0} \\ = \text{CDW7} | \text{CDW6} | \text{CDW5} | \text{CDW4} \end{array}$$

Rule 1:

To achieve the above-set goals, the originator of Read data into Concerto-C domain MUST inject data in the proper order of CDWs taking into account the width of the native interface at the Requester (nDWRIntf) in WRAP order of data thereafter, with total number of beats given by nBeatsRIntf. **The originator of Read data can be DMI, DII, or a snooperAIU.**

Rule 2:

Throughout the journey from the originator of Read data to the Requester AIU, the data is consistently referred to by RDW labels (as shown in examples of Diagrams 3 and 4).

Any link inside Concerto-C domain, including at the originator of Read data in Concerto-C is always of size some power-of-2 DWs (i.e. 1, 2, 4, or 8 DWs). The DWs on such a link are referred to as **Link-DWs**, or **LDWs**. A beat on the link is referred to as Link Bus Word, or **LBW**.

The following shows labeling of LDWs within the first beat of various sizes of LBWs:

Diagram 5:

1	DW LBW:	LDW0
2	DW LBW:	LDW1 LDW0
4	DW LBW:	LDW3 LDW2 LDW1 LDW0
8	DW LBW:	LDW7 LDW6 LDW5 LDW4 LDW3 LDW2 LDW1 LDW0

Rule 3:

The following assignments are used throughout the data journey:

The initiator of data transmission (DMI, DII, or snooper AIU) assigns RDWs to the correspondingly labeled LDWs where the link refers to its own SMI interface – as shown in Diagram 6.

Diagram 6:

$$\begin{array}{l} \text{LDW0} = \text{RDW0} \\ \text{LDW1} = \text{RDW1} \\ \text{LDW2} = \text{RDW2} \\ \text{LDW3} = \text{RDW3} \\ \text{LDW4} = \text{RDW4} \\ \text{LDW5} = \text{RDW5} \\ \text{LDW6} = \text{RDW6} \\ \text{LDW7} = \text{RDW7} \end{array}$$

From here on, only the LDWs are referenced.

Rule 4:

On any link, data is transmitted in little endian format within a beat and in little endian order of beats – less significant data first, with significance indicated by the subscript of the LDW.

This rule is used in data width converters. These converters are not aware of CDW identifier or the width of the requester's data bus.

The result is, for any data link, the data is sent strictly in the order of RDWs as labeled above, aligned correspondingly with LBWs in order of their significance – least significant LBW first.

Thus, for a link of size 1 DW, the assignment of RDWs to LDWs would be as follows (note that each LBW has only one LDW, which is LDW0):

Diagram 7:

LBW0:	LDW0	should be:	RDW0
LBW1:	LDW1	should be:	RDW1
LBW2:	LDW2	should be:	RDW2
LBW3:	LDW3	should be:	RDW3
LBW4:	LDW4	should be:	RDW4
LBW5:	LDW5	should be:	RDW5
LBW6:	LDW6	should be:	RDW6
LBW7:	LDW7	should be:	RDW7

If the link size is 2 DWs, for example, the assignment of RDWs to LDWs would be as follows:

Diagram 8:

LBW0: LDW1 LDW0	=	RDW1 RDW0
LBW1: LDW3 LDW2	=	RDW3 RDW2
LBW2: LDW5 LDW4	=	RDW5 RDW4
LBW3: LDW7 LDW6	=	RDW7 RDW6

A width adapter converting from 1x to 2x width simply performs the following transformation on the output:

Transform 1 -> 2:

$$\begin{aligned} O-LBW0 &= I-LBW1 | I-LBW0 \\ O-LBW1 &= I-LBW3 | I-LBW2 \\ O-LBW2 &= I-LBW5 | I-LBW4 \\ O-LBW3 &= I-LBW7 | I-LBW6 \end{aligned}$$

etc. And an adapter going from 2x to 1x must split the beats:

Transform 2 -> 1:

$$\begin{aligned} O-LBW0 &= I-LBW0_Low & O-LBW1 &= I-LBW0_High \\ O-LBW2 &= I-LBW1_Low & O-LBW3 &= I-LBW1_High \\ O-LBW4 &= I-LBW2_Low & O-LBW5 &= I-LBW2_High \\ O-LBW6 &= I-LBW3_Low & O-LBW7 &= I-LBW3_High \end{aligned}$$

Rules 1 through 4 achieve delivery of Read data with zero cycle wait due to out of order data.

8.4.2 DTW requests

Write data is also transmitted in the same manner. The data being sent again references the Interface Size at the issuer of the Write and not of the destination unit's data bus width.

(Note that the requester or a snooper is assumed to be unaware of the home unit's data bus width.)

Note

The DMI or DII may have to buffer the data before it is sent out on the unit's native AXI bus

Example:

Let's say the Write requester has 128 bit native data bus, but DMI has a 256 bit native data bus.

Let's further assume that the critical DW is 2 - same as the first example.

Assuming that the SMI interface of the requester AIU is the same width as its native interface The requester will format the received data in the following manner on its SMI interface:

$$\text{LBW0} = \text{RBW0} = \text{CDW3} | \text{CDW2}$$

$$\text{LBW1} = \text{RBW1} = \text{CDW5} | \text{CDW4}$$

$$\text{LBW2} = \text{RBW2} = \text{CDW7} | \text{CDW6}$$

$$\text{LBW3} = \text{RBW3} = \text{CDW1} | \text{CDW0}$$

Along the way, let's assume there is a width converter from 128 bits to 256 bits. After the width converter the data on the link will be:

$$\text{LBW0} = \text{LDW3} | \text{LDW2} | \text{LDW1} | \text{LDW0} = \text{CDW5} | \text{CDW4} | \text{CDW3} | \text{CDW2}$$

$$\text{LBW1} = \text{LDW7} | \text{LDW6} | \text{LDW5} | \text{LDW4} = \text{CDW1} | \text{CDW0} | \text{CDW7} | \text{CDW6}$$

Note that in Ncore 3, the address is preserved end-to-end.

The DMI must produce the data in Diagram 5 on its data bus (for it to be legally formatted).

Diagram 5:

1	DW LBW:	LDW0
2	DW LBW:	LDW1 LDW0
4	DW LBW:	LDW3 LDW2 LDW1 LDW0
8	DW LBW:	LDW7 LDW6 LDW5 LDW4 LDW3 LDW2 LDW1 LDW0

This will require DMI to hold (CDW3 | CDW2) until it receives (CDW1 | CDW0) before transmitting data on its AXI interface.

8.4.3 A few reminders

AXI address needs to be width aligned for WRAP – so zero out least significant bits of address as needed when transmitting it on AXI. No further modification is necessary.

A cacheline worth of buffer is needed in DMI and DII on both Read and Write paths for above transformation.

A cacheline worth of buffer is needed in a snooper AIU supplying DTRreq to a requesting AIU. The buffer is needed along the snooper AIU's Write path.

Chapter 9: Miscellaneous Notes

This chapter contains miscellaneous notes related to Concerto-C protocol. Here they are not categorized with any scheme, but eventually will be moved to appropriate sections within this document.

9.1 CHI Support

9.1.1 RetToSrc

In Ncore 3, **RetToSrc** is always driven 0 by an AIU in the Snoop Flit to a CHI agent.

9.1.2 Forwarding snoop

Ncore 3 does not implement Forwarding snoops.

9.1.3 FWDNodeID

Since Ncore 3 does not use Forwarding snoops, **FWDNodeID** for the Snoop Flit is treated as don't care. The AIU will initialize this field using a reserved value, 0x000.

9.1.4 DoNotGoToSD

Ncore 3 AIUs will not assert **DoNotGoToSD** in the Snoop Flit.

9.1.5 LikelyShared

Ncore 3.0 does not make use of the hint **LikelyShared**.

9.1.6 AllowRetry

Ncore 3 does not make use of the hint **AllowRetry**. Ncore 3 does not make implement the Retry protocol.

9.1.7 DoNotDataPull

Ncore 3.0 does not assert **DoNotDataPull**. However, it might be implemented for Engineering Debug Mode.

9.1.8 FwdState

Since Ncore 3.0 does not make use of forwarding snoops, **FwdState** field in the Data flit is treated as don't care.

9.1.9 DataSource

Ncore 3.0 does not make use the DataSource field and always returns a value of 0000. Later versions of Ncore 3 will need to implement this function to improve prefetching.

9.1.10 DVE Completions

DVMOp completions will be signaled by the DVE back to the requesting AIU via **CMPrsp** message.

9.1.11 Handling of StashOnces

To keep with the spirit of the CHI protocol (it is important to try to do this because it usually reflects some subtle understanding of the use cases), **StashOnce** transactions are not supposed to perturb the state of the system if the target of the transaction is unwilling to accept the stash. At present, the stashing snoop is sent to non-target AIUs without waiting a Snarf=1 response from the targetAIU (this is documented in the current transaction flow diagrams). This means that the state of the cacheline in these other agents will be disturbed even if the target indicates Snarf = 0, meaning stashing request is NOT accepted.

The following architecture decision has been made, following the CHI architecture recommendation:

DCE shall implement:

- The DCE will send the stashing snoop to the target and wait for the snoop response from the target.
 - If Snarf = 0 in the target snoop response, no further snoops are sent out and the operation is closed out.
 - After all outstanding snoop responses have been received, DCE will issue an MRD to the DMI to request the appropriate DTRreq to the snarfing AIU.
-

Target AIU:

No change is required in the target AIU's logic

Other (non-target) snooper AIU:

For SnpStshShd, the AIU generates a SnpShared snoop to the agent.

If dirty data is delivered by the snooping agent, a DTWData(Full/Ptl)Dty is sent to the DMI. After receiving the DTWrsp, the snooper AIU sends SNPrsp to the DCE

For SnpStshUnq, the AIU generates a SnpUnique snoop to the agent.

If dirty data is delivered by the snooping agent, a DTWData(Full/Ptl)Dty is sent to the DMI. After receiving the DTWrsp, the snooper AIU sends SNPrsp to the DCE

This will keep the snooper AIU design the same as it is today.

9.1.12 DVM payload distribution: VMIDext=VMID[15:8] is a separate field outside of the Addr field

Refer to AAMBA_5_chi_architecture_specification_IHI0050B.pdf

- The last row in the table below indicates VMIDext is outside of the address field in Snoop.

NOTE: The table 8.3 in the CHI spec shows VMIDext comes from bit [63:56] of the 8-byte DVM data packet. This VMIDext needs to be put in the address field of Snoop Part 1 message.

In Concerto-C, MPF1 of the SnpReq for DVM Op will be used to carry VMIDext field.

9.1.13 EWA for CHI Atomics

CHI VZ = !EWA in CCMP. CHI allows EWA to be 0 or 1 for Atomics. However, Ncore 3 will ignore the VZ value and treat it as being 0. For performance reasons, the Atomics are implemented and stored within the SMC and are not automatically pushed to DDR.

Suppression of intervention from agents with SC states

Note: In Ncore 3, snooped agents that have a copy in SC state are inhibited from sending data by:

- Not snooping such an agent, if directory is configured to so indicate, and/or
- Negating RetToSrc signal (for CHI-B agents).
- **Note:** UP=1 (here Unique Provider :-)) can be used to choose a unique SC ACE intervenor to transmit the intervention data, with UP=0 AIUs purging possible other intervenors.

9.2 MPF usage

Table 9-1: MPF Usage

Message Type	MPF1	MPF2	MPF3
CmdReq	<ul style="list-style-type: none"> Used to carry {Valid, StashNId} in the case of Stash Cmds, or ArgV: Argument Vector. ArgV[] is also used to carry: <ul style="list-style-type: none"> - Atomic Op code in the case of atomic Cmd. - {BT[1:0] ASize[2:0] ALength[2:0]} combined field of a non-coherent (CH=0), and non-cacheable (CA=0) CmdRdNC or CmdWrNC command that is a result of an AXI-borne request command. 	<p>Used to carry {Valid, StashLPId} in the case of Stash Cmds, or {Flow Id} in the case of non-Stash Cmds.</p> <p>Flow Id: Unique flow within the Initiator AIU. It can refer to a Unique Thread identifier, or an AXIID, or a Stream Id within the initiator AUID.</p>	N/A
SnpReq	Used to carry {StashNId} for a Stash snoop transactions (note, no Valid bit for SnpReq), or used to carry DTR Target FUId = Original Initiator FUId for non-stash snoop message	Used to carry {Valid, StashLPId} for a Stash snoop messages, with Valid being the most significant bit of the field and LPID being zero extended to fill the rest of the field, or used to carry DTR Message Id = Message Id of the original Cmd message for the non-Stash snoop messages	Used for certain Read Snoops. The field carries an AIUID of the AIU that is permitted to issue a DTR-req, if possible, to the requesting AIU.
MrdReq	Used to carry DTR Target Id = Initiator Id of the original Cmd message	Used to carry DTR Message Id = Message Id of the original Cmd message	N/A
StrReq	Used to carry {Valid, StashNId} in a case of Stashing Cmd if Snarf = 1.	Used to carry Message Id to be used by DTRs if Snarf = 1.	N/A
DtrReq	Used to identify starting DW for multi-DTW response for LongNCReads (future).	N/A	N/A
DtwReq	Used to carry {StashNId} or {Return Initiator Id} Also used to identify starting DW for multi-DTW response for Long- NCWrites (future).	<p>In the case of primary data of a Write command:</p> <p>Unique Reference Identifier of the original Write message to which this data belongs.</p> <p>In the case of secondary data in response to a snoop:</p> <p>Message Id supplied in the Snoop request (via MPF2).</p>	N/A
SnpRsp	Used to carry Message Id to be used by a Stashing DTR	N/A	N/A

9.3 I/O AIU Coherency State Transition Tables

9.3.1 AXI I/O AIU Tables

These tables are for MOESI for IOAIU-AXI only ($M \rightarrow UD$, $0 \rightarrow SD$, $E \rightarrow UC$, $S \rightarrow SC$ & $I \rightarrow IX$) The **Trans** column lists the AXI transaction type.

The **MT** column lists the memory type of the AXI transaction, and the **SZ** column indicates whether the access size of transaction is a partial cacheline (Ptl) or a full cacheline (Full).

The **CS** column indicates the current cache state for the cacheline copy in the IOAIU.

The **AP** column specifies the allocation policy for a read miss or a write miss, i.e., either an allocating transaction (AL) or a non-allocating transaction (NA). This allocation policy is determined either by looking at the AxCache bits or CSR configuration.

The remaining columns indicate the next cache state (**NS**) after the completion of the AXI operation, the type of message issued (**Issued MsgType**), and the outstanding transaction class (**OTC**). For this column, the class may be a protocol coherent (COH) or protocol update (UPD) transaction. The next four columns list the allowed ending cache states for each case for IO-AIU.

CCMP CMDreq generated in response to Native request transaction:

Table 9-2: Native to internal command translation

Trans	MT	SZ	CS	AP	NS	Issued MsgType	OTC	Notes
Read	x	x	IX	NA	IX	CmdRdNITC	COH	Miss no-allocate
				AL	SC/SD/ UC/UD	CmdRdVld	COH	Miss allocate
			SC	x	---	---	---	Load Hit
			SD	x	---	---	---	Load Hit
			UD	x	---	---	---	Load Hit
			UC	x	---	---	---	Load Hit
Write	WB	Ptl	IX	NA	IX	CmdWrUnqPtl	COH	Store Miss no-allocate
				AL	UD	CmdRdUnq	COH	Store Miss allocate
			SC	x	UD	CmdRdUnq	COH	Store Miss
			SD	x	UD	CmdRdUnq	COH	Store Miss - 1/31/2019: Drop data if SD not lost in the interim, but merely upgrade state to UD. (Note, data could be from memory and thus stale.)
				UC	x	UD		
			UD	x	UD	---	---	Store Hit
		Full	IX	NA	IX	CmdWrUnqFull	COH	Store Miss no-allocate
				AL	UD	CmdMkUnq	COH	Store Miss allocate
			SC	x	UD	CmdMkUnq	COH	Store Miss
			SD	x	UD	CmdMkUnq	COH	Store Miss
				UD	x	---		
			UC	x	UD	---	---	Store Hit

Note1

For Write Ptl, the current states SC or SD issue CmdRdUnq instead of CmdClnUnq to avoid the requiring the creation of the state UCE and the attendant design changes. Under an intervening snoop that might invalidate the SC and SD state, the CmdClnUnq will create the UCE state with its completion.

Note2

If cache is not present map the commands where allocation policy (AP) in NA and the next stats (NS) is IX.

Note3

If the AXI AIU is in non-coherent mode then Read is mapped to CmdRdNc and write is mapped to CmdWrNc.

9.3.2 Proxy Cache state transitions

As function of issued CCMP Commands

Table 9-3: Native to internal command translation

Issued MsgType	SZ	CS	DTR	NS	TR	Notes
CmdRdNITC (Read)	x	IX	DtrDataInv	IX	00	Data not cached
CmdRdVld (Read)	x	IX	DtrDataSCln	SC	01	Data cached
			DtrDataSDty	SD	10	
			DtrDataUCln	UC	10	
			DtrDataUDty	UD	10	
CmdRdUnq (Read)	x	IX	DtrDataUCln	UC	10	Merge partial data and cache
			DtrDataUDty	UD	10	
CmdRdUnq (Write)	Ptl	IX/SC/SD	DtrDataUCln or DtrDataUDty	UD	10	Merge partial data and cache
CmdMkUnq	Full	IX/SC/SD	X	UD	10	Replace cacheline with Write data.

Proxy Cache Responses to CCMP snoops:

Table 9-4: Proxy Cache Responses

Snoop MsgType	OTC	CS	NS	RV	RS	DC	DT[1]	DT[0]	DTRreq	DTWreq
SnpClnDtr,	None/COH	IX	IX	0	0	0	0	0	-	-
		SC	SC	1	1	0	0	0	-	-
		SD	SD	1	0	0	1	0	DtrDataSCln	-
		UC	SC	1	1	0	1	0	DtrDataSCln	-
		UD	SD	1	0	0	1	0	DtrDataSCln	-
SnpNoSDInt	None/COH	IX	IX	0	0	0	0	0	-	-
		SC	SC	1	1	0	0	0	-	-
		SD	SD	1	0	0	1	0	DtrDataSCln	-
		UC	SC	1	1	0	1	0	DtrDataSCln	-
		UD	SD	1	0	0	1	0	DtrDataSCln	-

Table 9-4: Proxy Cache Responses

Snoop MsgType	OTC	CS	NS	RV	RS	DC	DT[1]	DT[0]	DTRreq	DTWreq
SnpVldDtr	None/COH	IX	IX	0	0	0	0	0	-	-
		SC	SC	1	1	0	0	0	-	-
		SD	SD	1	0	0	1	0	DtrDataSIn	-
		UC	SC	1	1	0	1	0	DtrDataSIn	-
		UD	SD	1	0	0	1	0	DtrDataSIn	-
SnpInvDtr,	None/COH	IX	IX	0	0	0	0	0	-	-
		SC	IX	0	0	0	0	0	-	-
		SD	IX	0	0	1	1	0	DtrDataUDty	-
		UC	IX	0	0	1	1	0	DtrDataUCln	-
		UD	IX	0	0	1	1	0	DtrDataUDty	-
SnpNITC	None/COH	IX	IX	0	0	0	0	0	-	-
		SC	SC	1	1	0	0	0	-	-
		SD	SD	1	0	0	1	0	DtrDataInv (CHI TOF) DtrDataSIn (ACE TOF)	(Feb 12, 2019)
		UC	UC	1	0	0	1	0	DtrDataInv	-
		UD	UD	1	0	0	1	0	DtrDataInv	-
SnpNITCCI	None/ COH	IX	IX	0	0	0	0	0	-	-
		SC	IX	0	0	0	0	0	-	-
		SD	IX	0	0	0	1	1	DtrDataInv	DtwDataDty
		UC	IX	0	0	0	1	0	DtrDataInv	-
		UD	IX	0	0	0	1	1	DtrDataInv	DtwDataDty
SnpNITCMI	None/COH	IX	IX	0	0	0	0	0	-	-
		SC	IX	0	0	0	0	0	-	-
		SD	IX	0	0	0	1	0	DtrDataInv	-
		UC	IX	0	0	0	1	0	DtrDataInv	-
		UD	IX	0	0	0	1	0	DtrDataInv	-
SnpClnDtw	None/COH	IX	IX	0	0	0	0	0	-	-
		SC	SC	1	1	0	0	0	-	-
		SD	SC	1	1	0	0	1	-	DtwDataDty
		UC	UC	1	0	0	0	0	-	-
		UD	UC	1	0	0	0	1	-	DtwDataDty
SnpInvDtw / SnpUnqStsh / SnpStshUnq (Also used for Directory Recall in CCMP)	None/ COH	IX	IX	0	0	0	0	0	-	-
		SC	IX	0	0	0	0	0	-	-
		SD	IX	0	0	0	0	1	-	DtwDataDty
		UC	IX	0	0	0	0	0	-	-
		UD	IX	0	0	0	0	1	-	DtwDataDty
SnpInv / SnpInvStsh	None/COH	x	IX	0	0	0	0	0	-	-
SnpStshShd	None/COH	IX	IX	0	0	0	0	0	-	-
		SC	SC	1	1	0	0	0	-	-
		SD	SD	1	0	0	0	1	-	DtwDataDty
		UC	SC	1	1	0	0	0	-	-
		UD	SD	1	0	0	0	1	-	DtwDataDty

ACE-Lite I/O AIU Tables

These tables are for MOESI for IOAIU-ACE-Lite only ($M \rightarrow UD$, $0 \rightarrow SD$, $E \rightarrow UC$, $S \rightarrow SC$, & $I \rightarrow IX$) The **Trans** column list the AXI transaction type

The **MT** column lists the memory type of the AXI transaction, and the **SZ** column indicates whether the access size of transaction is a partial cacheline (Ptl) or a full cacheline (Full).

The **CS** column indicates the current cache state for the cacheline copy in the IOAIU.

The **AP** column specifies the allocation policy for a read miss or a write miss, i.e. either an allocating transaction (AL) or a non-allocating transaction (NA). This allocation policy is determined either by looking at the AxCache bits or CSR configuration

The remaining columns indicate the next cache state (**NS**) after the completion of the AXI operation, the type of message issued (**Issued MsgType**), and the outstanding transaction class (**OTC**). For this column, the class may be a protocol coherent (COH) or protocol update (UPD) transaction. The next four columns list the allowed ending cache states for each case for IO-AIU.

CCMP CMDreq generated from ACE-Lite Proxy Cache in response to Native request transaction:

Note for the following table: Operations indicated inside {} are executed “atomically.” Snoops are either accepted before such operations or after all the operations within {} are completed. Snoops handled before the {} operation may change the state and therefore the table row that might be applicable for the Transaction that caused the {} operations.

UpdSCIn will not be implemented in Ncore 3.0 but will be implemented on Ncore3’s future revisions.

Table 9-5: Native to internal command translation

Transaction	MT	SZ	CS	AP	NS	Issued MsgType	OTC	Notes
ReadOnce	X	X	IX	NA	IX	CmdRdNITC	COH	Miss, no-allocate
				AL	SC/ SD/ UC/ UD	CmdRdVld	COH	Miss, allocate
				SC	X	---	---	Load Hit
				SD	X	---	---	Load Hit
				UC	X	---	---	Load Hit
				UD	X	---	---	Load Hit
CleanShared	X	X	IX	X	IX	CmdClnVld	COH	
			SC	X	SC	CmdClnVld	COH	
			SD	X	SC	{ Nc-Write (VZ=1); UpdSCIn } CmdClnVld	---	NC-Write uses DtwDataDty. CmdClnVld is sent to ensure SMC is cleaned to memory.
			UC	X	UC	---	---	
			UD	X	UC	{ Nc-Write (VZ=1) } CmdClnVld	---	Nc-Write uses DtwDataDty. CmdClnVld is sent to ensure SMC is cleaned to memory.

Table 9-5: Native to internal command translation

Transaction	MT	SZ	CS	AP	NS	Issued MsgType	OTC	Notes
CleanSharedPersist (ACE5-Lite)	X	X	IX	X	IX	CmdClnVldPsist	COH	
			SC	X	SC	CmdClnVldPsist	COH	
			SD	X	SC	{ Nc-Write (VZ=1); UpdScln } CmdClnVldPsist	---	Nc-Write uses DtwDataDty. CmdClnVldPsist is sent to ensure SMC is cleaned to memory.
			UC	X	UC	---	---	
			UD	X	UC	{ Nc-Write (VZ=1) } CmdClnVldPsist	---	Nc-Write uses DtwDataDty. CmdClnVldPsist is sent to ensure SMC is cleaned to memory.
CleanInvalid	X	X	IX	X	IX	CmdClnInv	COH	
			SC	X	IX	CmdClnInv	COH	
			SD	X	IX	{ Nc-Write (VZ=1); UpdInv } CmdClnInv	---	Nc-Write uses DtwDataDty. CmdClnInv is sent to ensure SMC is cleaned and invalidated.
			UC	X	IX	---	---	
			UD	X	IX	{ Nc-Write (VZ=1); UpdInv } CmdClnInv	---	Nc-Write uses DtwDataDty. CmdClnInv is sent to ensure SMC is cleaned and invalidated.
CleanSharedPersist (ACE5-Lite)	X	X	IX	X	IX	CmdClnVldPsist	COH	
			SC	X	SC	CmdClnVldPsist	COH	
			SD	X	SC	{ Nc-Write (VZ=1); UpdScln } CmdClnVldPsist	---	Nc-Write uses DtwDataDty. CmdClnVldPsist is sent to ensure SMC is cleaned to memory.
			UC	X	UC	---	---	
			UD	X	UC	{ Nc-Write (VZ=1) } CmdClnVldPsist	---	Nc-Write uses DtwDataDty. CmdClnVldPsist is sent to ensure SMC is cleaned to memory.
CleanInvalid	X	X	IX	x	IX	CmdClnInv	COH	
			SC	x	IX	CmdClnInv	COH	
			SD	x	IX	{ Nc-Write (VZ=1); UpdInv } CmdClnInv	---	Nc-Write uses DtwDataDty. CmdClnInv is sent to ensure SMC is cleaned and invalidated.
			UC	x	IX	---	---	
			UD	x	IX	{ Nc-Write (VZ=1); UpdInv } CmdClnInv	---	Nc-Write uses DtwDataDty. CmdClnInv is sent to ensure SMC is cleaned and invalidated.
			UD	x	IX	{ Nc-Write (VZ=1); UpdInv } CmdClnInv	---	Nc-Write uses DtwDataDty. CmdClnInv is sent to ensure SMC is cleaned and invalidated.
MakeInvalid	X	X	IX	x	IX	CmdMkInv	COH	
			SC	x	IX	{ UpdInv } CmdMkInv	COH	CmdMkInv is sent to ensure SMC is invalidated
			SD	x	IX	{ Nc-Write (VZ=1); UpdInv } CmdMkInv	COH	CmdMkInv is treated locally as CmdClnInv under xD to ensure coherency is preserved for intervening snoops. CmdMkInv is sent to ensure SMC is invalidated.
			UC	x	IX	{ UpdInv } CmdMkInv	---	CmdMkInv is sent to ensure SMC is invalidated.

Table 9-5: Native to internal command translation

Transaction	MT	SZ	CS	AP	NS	Issued MsgType	OTC	Notes
MakeInvalid	X	X	UD	x	IX	{Nc-Write (VZ=1); UpdInv } CmdMkInv	---	CmdMkInv is treated locally as CmdClnInv under xD to ensure coherency is preserved for intervening snoops. CmdMkInv is sent to ensure SMC is invalidated.
ReadOnceClean-Invalid	x	x	IX	NA	IX	CmdRdNITClnInv	COH	Load Miss
			SC	x	IX	CmdClnInv	---	Load Hit; CmdClnInv also ensures SMC is cleaned and invalidated.
			SD	x	IX	{ Nc-Write (VZ=1); UpdInv } CmdClnInv	---	Load Hit; Nc-Write uses DtwDataDty. CmdClnInv also ensures SMC is cleaned and invalidated.
			UC	x	IX	{ UpdInv } CmdClnInv	---	Load Hit; CmdClnInv is sent to ensure SMC is cleaned and invalidated.
ReadOnceMake-Invalid	x	x	IX	NA	IX	CmdRdNITCMkInv	COH	Load Miss
			SC	x	IX	CmdMkInv	---	Load Hit
			SD	x	IX	{ Nc-Write (VZ=1); UpdInv } CmdMkInv	---	Load Hit; Treated as ROCI locally under xD to ensure coherency is preserved for intervening snoops. Final CmdMkInv ensures SMC is invalidated.
			UC	x	IX	{ UpdInv } CmdMkInv	---	CmdMkInv is sent to ensure SMC is invalidated.
			UD	x	---	{ Nc-Write (VZ=1); UpdInv } CmdMkInv	---	Load Hit; Treated as ROCI locally under xD to ensure coherency is preserved for intervening snoops. CmdMkInv is sent to ensure SMC is invalidated.
WriteUnique	WB	Ptl	IX	NA	IX	CmdWrUnqPtl	COH	Store Miss no-allocate
			AL	UD	---	CmdRdUnq	COH	Store Miss allocate
			SC	x	UD	CmdRdUnq	COH	Store Miss
			SD	x	UD	CmdRdUnq	COH	Drop Read data if cacheline not lost since Cmd sent
			UC	x	UD	---	---	Store Hit
			UD	x	UD	---	---	Store Hit
		Full	IX	NA	IX	CmdWrUnqFull	COH	Store Miss no-allocate
			AL	UD	---	CmdMkUnq	COH	Store Miss allocate
			SC	x	UD	CmdMkUnq	COH	Store Miss
			SD	x	UD	CmdMkUnq	COH	Store Miss
			UD	x	---	---	---	Store Hit
			UC	x	UD	---	---	Store Hit
			---	---	---	---	---	---

Note:

For Write Ptl, the current states SC or SD issue CmdRdUnq instead of CmdClnUnq to avoid the requiring the creation of the state UCE and the attendant design changes. Under an intervening snoop that might invalidate the SC and SD state, the CmdClnUnq will create the UCE state with its completion.

CCMP CMDreq generated from ACE5-Lite Proxy Cache in response to Native **Stashing** request transaction “**targeting**” the Proxy Cache:

Table 9-6: CCMP CMDreq generated from ACE5-Lite Proxy Cache (Native **Stashing** Transaction) – Targeting Proxy Cache

Native Transaction	MT	SZ	CS	AP	NS	Issued MsgType	Cache Action	Notes
StashWrite	WB	Ptl	IX	AL	UD	CmdRdUnq	Merge	Store Miss allocate
			SC	x	UD	CmdRdUnq	Merge	Store Miss
			SD	x	UD	CmdRdUnq	Merge	Store Miss
			UC	x	UD	---	Merge	Store Hit
			UD	x	UD	---	Merge	Store Hit
	WB	Full	IX	AL	UD	CmdMkUnq	Store	Store Miss allocate
			SC	x	UD	CmdMkUnq	Replace	Store Miss
			SD	x	UD	CmdMkUnq	Replace	Store Miss
			UD	x	---	---	Replace	Store Hit
			UC	x	UD	---	Replace	Store Hit
StashOnceShared	WB	Full	IX	AL	SC/SD UC/UD	CmdRdShd	---	Read Shared allocate
			SC	x	SC	---	---	---
			SD	x	SD	---	---	---
			UC	x	UC	---	---	---
			UD	x	UD	---	---	---
StashOnceUnique	WB	Full	IX	AL	UC UD	CmdRdUnq	---	Read Unique allocate
			SC	x	UC UD	CmdRdUnq	---	Read Unique allocate
			SD	x	UD	CmdRdUnq	---	Drop Read data if cacheline not lost since Cmd sent
			UC	x	UC	---	---	---
			UD	x	UD	---	---	---

CCMP CMDreq generated from ACE5-Lite Proxy Cache in response to Native **Stashing** request transaction
NOT “targeting” the Proxy Cache:

Table 9-7: CCMP CMDreq from ACE5-Lite Proxy Cache (Native Stashing Transactions) – Not targeting Proxy Cache

Native Transaction	MT	SZ	CS	AP	NS	Issued MsgType	Cache Action	Notes		
StashWrite	WB	Ptl	IX	x	IX	CmdWrStshPtl	Process snoops as required while the transaction is outstanding	The cache must support getting snooped back for the stashing snoop as normal		
			SC	x						
			SD	x						
			UC	x						
			UD	x						
		Full	IX	x	IX	CmdWrStshFull				
			SC	x						
			SD	x						
			UD	x						
			UC	x						
StashOnceShared	WB	Full	IX	x	IX	CmdLdCshShd	Process snoops as required while the transaction is outstanding	The cache must support getting snooped back for the stashing snoop as normal		
			SC	x	SC					
			SD	x	IX/SC/SD					
			UC	x	IX/SC					
			UD	x	IX/SC/SD					
StashOnceUnique	WB	Full	IX	x	IX	CmdLdCshUnq	Process snoops as required while the transaction is outstanding	The cache must support getting snooped back for the stashing snoop as normal		
			SC	x	IX					
			SD	x	IX					
			UC	x	IX					
			UC	x	IX					

Table 9-8: CCMP CMDreq generated from ACE5-Lite Proxy Cache in response to Native Atomic transaction request

Native Transaction	MT	SZ	CS	AP	NS	Issued MsgType	Cache Action	Notes		
CmdWrAtm CmdCompAtm CmdSwapAtm	X		IX	X	IX	Cmd<x>Atm	Process snoops as required while the transaction is outstanding	The cache must support getting snooped back for the stashing snoop as normal		
			SC	X						
			SD	X		Cmd<x>Atm w/ SS=0b1				
			UC	X						
			UD	X						

Note:

SS = 0b1 causes a self-snoop to occur.

Proxy Cache state transitions as a function of issued CCMP Commands

Table 9-9: Proxy Cache State Transitions

Issued MsgType	SZ	CS	DTR	NS	TR	Notes
CmdRdNITC (Read)	x	IX	DtrDataInv	IX	00	Data not cached
CmdRdVld (Read)	x	IX	DtrDataSCln	SC	01	Data cached
			DtrDataSDty	SD	10	
			DtrDataUCln	UC	10	
			DtrDataUDty	UD	10	
CmdRdUnq (Read)	x	IX	DtrDataUCln	UC	10	Data cached
			DtrDataUDty	UD	10	
CmdRdUnq (Write)	Ptl	IX/SC/SD	DtrDataUCln or DtrDataUDty	UD	10	Merge partial data and cache
CmdMkUnq	Full	IX/SC/SD	X	UD	10	Replace cacheline with Write data.

Table 9-10: Proxy Cache Responses to CCMP snoops

Snoop MsgType	OTC	CS	NS	RV	RS	DC	DT[1]	DT[0]	DTRreq	DTWreq
SnpClnDtr,	None/ COH	IX	IX	0	0	0	0	0	---	---
		SC	SC	1	1	0	0	0	---	---
		SD	SD	1	0	0	1	0	DtrDataSCln	---
		UC	SC	1	1	0	1	0	DtrDataSCln	---
		UD	SD	1	0	0	1	0	DtrDataSCln	---
SnpNoSDInt	None/ COH	IX	IX	0	0	0	0	0	---	---
		SC	SC	1	1	0	0	0	---	---
		SD	SD	1	0	0	1	0	DtrDataSCln	---
		UC	SC	1	1	0	1	0	DtrDataSCln	---
		UD	SD	1	0	0	1	0	DtrDataSCln	---
SnpVldDtr	None/ COH	IX	IX	0	0	0	0	0	---	---
		SC	SC	1	1	0	0	0	---	---
		SD	SD	1	0	0	1	0	DtrDataSCln	---
		UC	SC	1	1	0	1	0	DtrDataSCln	---
		UD	SD	1	0	0	1	0	DtrDataSCln	---
SnpInvDtr,	None/ COH	IX	IX	0	0	0	0	0	---	---
		SC	IX	0	0	0	0	0	---	---
		SD	IX	0	0	1	1	0	DtrDataUDty	---
		UC	IX	0	0	1	1	0	DtrDataUCln	---
		UD	IX	0	0	1	1	0	DtrDataUDty	---
SnpNITC	None/ COH	IX	IX	0	0	0	0	0	---	---
		SC	SC	1	1	0	0	0	---	---
		SD	SD	1	0	0	1	0	DtrDataInv	---
		UC	UC	1	0	0	1	0	DtrDataInv	---
		UD	UD	1	0	0	1	0	DtrDataInv	---

Table 9-10: Proxy Cache Responses to CCMP snoops

Snoop MsgType	OTC	CS	NS	RV	RS	DC	DT[1]	DT[0]	DTRreq	DTWreq
SnpNITCCI	None/ COH	IX	IX	0	0	0	0	0	---	---
		SC	IX	0	0	0	0	0	---	---
		SD	IX	0	0	0	1	1	DtrDataInv	DtwDataDty
		UC	IX	0	0	0	1	0	DtrDataInv	-
		UD	IX	0	0	0	1	1	DtrDataInv	DtwDataDty
SnpNITCMI	None/ COH	IX	IX	0	0	0	0	0	---	---
		SC	IX	0	0	0	0	0	---	---
		SD	IX	0	0	0	1	0	DtrDataInv	---
		UC	IX	0	0	0	1	0	DtrDataInv	---
		UD	IX	0	0	0	1	0	DtrDataInv	---
SnpClnDtw	None/ COH	IX	IX	0	0	0	0	0	---	---
		SC	SC	1	1	0	0	0	---	---
		SD	SC	1	1	0	0	1	---	DtwDataDty
		UC	UC	1	0	0	0	0	---	---
		UD	UC	1	0	0	0	1	---	DtwDataDty
SnpInvDtw (Also used for Directory Recall in CCMP)	None/ COH	IX	IX	0	0	0	0	0	---	---
		SC	IX	0	0	0	0	0	---	---
		SD	IX	0	0	0	0	1	---	DtwDataDty
		UC	IX	0	0	0	0	0	---	---
SnpInvDtw (continued)	None/ COH	UD	IX	0	0	0	0	1	---	DtwDataDty
SnpInv	None/ COH	x	IX	0	0	0	0	0	---	---

Chapter 10: Coherency Directory

This chapter describes the capabilities and elements of the coherency directory implemented in Ncore 3. Note that the construction and operation of this directory is independent of the CCMP protocol.

The material in this chapter is carried over almost verbatim from the documents pertaining to Ncore 2 since there is no fundamental difference to the coherency directory architecture in Ncore 3 from that in Ncore 2.

The **coherency directory** tracks the presence of cacheline copies in, and manages the contents of, the caches in each of the caching agents in the system. The functionality of the system directory is tightly integrated with that of the DCE, and the remainder of this chapter provides detailed information about the system directory.

10.1 Organization

The system directory is indexed in two dimensions: first, by cacheline address into a **directory partition**, which tracks the set of cacheline addresses associated with a given DCE; and second, by caching agent into a **directory slice**, which tracks a set of one or more caching agents. Depending on the system configuration, a system directory may consist of one or more directory partitions and one or more directory slices; however, each cacheline address is associated with a single directory partition, and each caching agent is associated with a single directory slice.

The intersection of a directory partition and a directory slice is a **directory segment**, which tracks the set of addresses associated with a DCE for a set of caching agents. For a given cacheline address, a directory segment performs a filtering function that can limit the number of snoop messages required to complete a protocol transaction by indicating which agents associated with the segment may be caching a copy of a cacheline. All directory segments in the same directory slice implement the same filtering function; as a result, for a given caching agent, the filtering function is independent of address and is applied across all cacheline addresses. With the exception of this restriction, each directory segment is configured and can operate independently, so directory segments in different slices may implement different filtering functions.

Note: In short, each DCE implements a directory partition, and caching agents are grouped into one or more directory segments, up to the number of caching agents, within that partition.

A directory segment may be configured not to store any information about the caches in the caching agents associated with the segment. In this case, the segment performs no filtering and is classified as a **null directory segment**, which implies that, when necessary, snoop messages are issued to all caching agents associated with the segment. If all directory segments are null, the system directory is a **null directory** and is effectively absent from the system.

Such a system relies on broadcasting snoop messages to all caching agents to maintain coherence.

10.1.1 Interfaces

The contents of a directory partition are read via a **lookup interface** to determine which caching agents must perform coherence operations, and the results of those operations are written via a **commit interface** to update the contents of the directory partition. Depending on the configuration of the directory segments within the directory partition, a **recall interface** may also be present.

To perform a system directory lookup, the cacheline address and the CacheID for the initiating agent, if defined for the initiating agent, are presented to the directory segments in the directory partition via the lookup interface. Each of the directory segments is accessed, and regardless of the configuration of each segment, the results from each are combined and presented in the **system directory lookup result**, which consists of an **owner lookup vector** and a **sharer lookup vector**.

Note

The system directory lookup result abstracts the structure of the individual segments within a partition.

Each caching agent in the system is represented by a bit in the owner lookup vector and a bit in the sharer lookup vector. Together, these result vectors identify the possible cacheline owner and possible cacheline sharers and determine the properties of **cacheline ownership** and **cacheline validity** for each caching

agent. Cacheline ownership specifies whether a caching agent has the owned copy of a cacheline, while cacheline validity specifies whether a caching agent has a valid copy of the cacheline. Both properties may be expressed as *true*, *uncertain*, or *false* based on the type of directory segment in which the caching agent is represented.

The lookup result may signal one of the following for each caching agent:

- If a caching agent is represented in *only* the owner lookup vector, the caching agent is the owner, cacheline ownership is *true*, and cacheline validity is *true*
- If a caching agent is represented in *only* the sharer lookup vector, the caching agent is a sharer, cacheline ownership is *false*, and cacheline validity is either *true* or *uncertain* based on the type of directory segment
- If the caching agent is represented in *both* the owner lookup vector and the sharer lookup vector, the caching agent may be **present** (i.e., the agent is either the owner or a sharer), cacheline ownership is *uncertain*, and cacheline validity is either *true* or *uncertain* based on the type of directory segment.
- If a caching agent is *not* represented in either the owner lookup vector or the sharer lookup vector, the caching agent is *not present* (i.e., the agent is *not* the owner and *not* a sharer), cacheline ownership is *false*, and cacheline validity is *false*

At most one caching agent may be represented as the owner. If an agent is represented as the owner, no caching agents may be represented as present, but if no agent is represented as the owner, any number of caching agents may be represented as present. In all cases, any number of caching agents may be represented as a sharer.

Note

The owner lookup vector and sharer lookup vector represent a logical interface between the system directory segments and the rest of the DCE logic, and an implementation may choose a more compact representation if the protocol semantics are preserved

Depending on the protocol transaction, a caching agent may be issued an owner snoop message when its corresponding owner lookup vector bit is set or may be issued a sharer snoop message when its corresponding sharer lookup vector bit is set and its corresponding owner lookup vector bit is not set. An owner snoop message is issued primarily to retrieve a cacheline copy from a caching agent, in the case of a coherent read transaction, or to perform an update of memory, in the case of a coherent clean transaction. A sharer snoop message is issued primarily to invalidate non-owned copies when a caching agent is modifying data or invalidating cacheline copies. An owner snoop message or a sharer snoop message may also be used to probe the state of a cacheline when cacheline validity is *uncertain* for a caching agent.

To perform a system directory commit, the cacheline address and the **system directory commit result**, which consists of an **owner commit vector** and a **sharer commit vector**, are presented to the directory segments in the directory partition via the commit interface. Each directory segment receives the portion of the system directory commit result that corresponds to the caching agents associated with the segment. Together, the result vectors identify the possible cacheline owner and cacheline sharers and determine the properties of cacheline ownership and cacheline validity for each caching agent.

The commit result may signal one of the following for each caching agent:

- If a caching agent is represented in *only* the owner commit vector, the caching agent is the owner, cacheline ownership is *true*, and cacheline validity is *true*

- If a caching agent is represented in *only* the sharer commit vector, the caching agent is a sharer, cacheline ownership is *false*, and cacheline validity is *true*
- If the caching agent is represented in *both* the owner commit vector and the sharer commit vector, the caching agent is present, cacheline ownership is *uncertain*, and cacheline validity is *true*
- If a caching agent is *not* represented in either the owner commit vector or the sharer commit vector, the caching agent is *not* present, cacheline ownership is *false*, and cacheline validity is *false*

At most one caching agent may be represented as the owner. In all cases, any number of caching agents may be represented as present, and any number of caching agents may be represented as a sharer. The contents of the directory segments may be written with the system directory commit result depending on the configuration of those segments.

Note

As indicated by the owner commit vector and owner sharer vector, cacheline validity always resolves to either *true* or *false*, since caching agents for which cacheline validity is *uncertain* in the lookup vectors are issued a snoop message.

The owner commit vector and sharer commit vector represent a logical interface between the system directory segments and the rest of the DCE logic, and an implementation may choose a more compact representation if the protocol semantics are preserved.

For certain configurations, directory partitions may implement a **recall interface**, on which a **recall transaction** is signaled when a directory entry replacement is required. A recall transaction is performed by issuing SnpRecall messages to the caching agents associated with a directory segment and ensures that the cacheline copies for a given address are removed from the caches of the caching agents associated with the segment. The recall transaction specifies a cacheline address and a **recall vector**, which indicates whether or not a caching agent is present in the directory segment. The recall vector is equivalent to the logical OR of the owner lookup vector and sharer lookup vector for the directory entry selected for replacement, and if a bit is set in the recall vector, a SnpRecall message is sent to the corresponding caching agent; otherwise, no message is sent to the caching agent. A recall transaction does not distinguish between owner snoop messages and sharer snoop messages.

A recall transaction is issued with the following attributes:

- Visibility: coherence domain visible
- Allocation: allocate
- Data Transfer State: not constrained

All vectors described in this section are indexed by CachelD, which identifies the bit that corresponds to a particular caching agent.

For a null directory, the above interfaces are absent, and all caching agents are presumed to be present in the directory; consequently, owner snoop messages are sent to all caching agents when required. For all other configurations, at least one directory segment *must* be configured to store state information about the caches in the caching agents associated with the segment, and snoop filtering is performed based on that information. The following subsections describe the behaviors of various filter types that may be assigned to a directory segment.

10.2 Null Filters

Null filters contain no information about the caches in the caching agents associated with the segment, and a segment configured with a null filter is classified as a null directory segment.

10.2.1 Lookups

If a directory segment is configured as a null filter, no structures are accessed on a system directory lookup, and all caching agents associated with the filter must be snooped. Because no information is stored, cacheline ownership and cacheline validity are *uncertain*. The bullets below describe how the lookup results are generated for a null filter.

Null Directory Segment

- No access:
 - Owner lookup vector – set for all caching agents (cacheline ownership is *uncertain*)
 - Sharer lookup vector – set for all caching agents (cacheline validity is *uncertain*)

Note: For a given caching agent in a null directory segment, the values of the owner lookup vector bit and the sharer lookup vector bit *must* be {set, set}, and the resulting cacheline ownership and cacheline validity states are {*uncertain*, *uncertain*}. Thus, for a null directory segment, each caching agent is represented as present in the lookup vectors.

For null filters, Table 10-1 summarizes the states of the owner lookup vector (**OL[c]**) and sharer lookup vector (**SL[c]**) bits for a given caching agent with CacheID **c** and their meaning with respect to cacheline ownership and cacheline validity, respectively, for that agent. Cells marked N/A and values in parentheses represent combinations that are not allowed on a system directory lookup; however, values in parentheses can be reached based on cacheline ownership filtering described in Section 5.6.

Table 10-1: Cacheline Ownership and Validity for a for a Null Filter on Lookup

OL[c]	SL[c]	Null Ownership	Null Validity
0	0	N/A	N/A
0	1	(False)	(Uncertain)
1	0	N/A	N/A
1	1	Uncertain	Uncertain

10.2.2 Commits

The system directory commit result for a null directory segment is ignored.

Note

The following is included to illustrate the possible states of the owner commit vector and sharer commit vector for the caching agents associated with a null filter.

For null filters, Table 10-2 summarizes the states of the owner commit vector (**OL[c]**) and sharer commit vector (**SL[c]**) bits for a given caching agent with CacheID **c** and their meaning with respect to cacheline ownership and cacheline validity, respectively, for that agent. Cells marked *N/A* represent combinations that are not allowed on a system directory commit, and values in brackets can only be reached by the initiating caching agent. At most one caching agent in the system directory commit result may have a value indicated with an asterisk.

Table 10-2: Cacheline Ownership and Validity for a Null Filter on Commit

OL[c]	SL[c]	Null Ownership	Null Validity
0	0	False	False
0	1	False	True
1	0	[True*]	[True*]
1	1	Uncertain	True

10.2.3 Tag Filters

Tag filters are set associative structures whose entries are tagged by cacheline address. In addition to storing the address, all entries in a tag filter may record one or both of the following pieces of information:

- A **presence vector** – each valid cacheline copy is tracked with a valid bit that represents validity in a caching agent or in a set of caching agents
- An **owner pointer** – the owned cacheline copy is tracked with a pointer field that indicates ownership in a caching agent

At a minimum, each entry in a tag filter stores a presence vector. A presence vector valid bit is **precise** if the valid bit represents a single caching agent; otherwise, a presence vector valid bit is **coarse** if the valid bit represents a set of two or more caching agents, collectively known as a **caching agent group**. In a precise representation, the presence vector valid bit *must* be set if the cacheline copy in the corresponding caching agent is in any valid state, i.e. SC, OC, OD, UC, or UD; however, the valid bit *may* be clear if the cacheline copy in the corresponding caching agent is in the invalid state, i.e. IX. In a coarse representation, the valid bit *must* be set if any cacheline copy in any of the corresponding caching agents is in a valid state, and the valid bit *may* be clear only if all cacheline copies in all of the corresponding caching agents are in the invalid state.

If a tag filter has been configured to store owner pointers, the owner pointer field *must* be encoded with a value representing a caching agent in any owned state, i.e. OC, OD, UC, or UD; however, the pointer field *may* be encoded with a value representing a caching agent in the shared or the invalid state, i.e. SC or IX, if no other caching agents in the system are in an owned state.

The encoding of the pointer field indicates either that a particular caching agent associated with the tag

filter is the cacheline owner or that none of the caching agents associated with the tag filter is the cacheline owner. In the former case, the pointer field is valid, while in the latter, the pointer field is invalid. In other words, the state space of the pointer field consists of one valid encoding for each caching agent associated with the tag filter and one invalid encoding. Furthermore, because the owner pointer field identifies at most a single caching agent, an owner pointer is defined to be precise.

In addition, when a tag filter is configured to store owner pointers, the presence vector becomes a **sharer vector**. Similar to a presence vector valid bit, a sharer vector valid bit may be precise or coarse based on the number of caching agents represented by the bit. In a precise representation, the sharer vector valid bit *must* be set if the cacheline copy in the corresponding caching agent is in the shared state, i.e. SC; however, the valid bit *may* be clear if the cacheline copy in the corresponding caching agent is in the invalid state, i.e. IX. In a coarse representation, the valid bit *must* be set if any cacheline copy in any of the corresponding caching agents is in the shared state, and the valid bit *may* be clear only if all cacheline copies in all of the corresponding caching agents are in the invalid state.

A caching agent may be represented by the owner pointer field, a sharer vector valid bit, or both; however, if represented by both, the sharer lookup vector bit corresponding to that caching agent *must not* be set.

A directory segment, based on the information stored in its tag filter, may be classified as one of the following:

- A presence vector directory segment stores a presence vector only
- An **owner-sharer directory segment** stores an owner pointer and a sharer vector

Note

Recall that a directory segment is simply a directory slice partitioned by DCE. As a result, if one segment is configured a certain way, all segments in the slice are configured the same way.

10.2.4 Lookups

If a directory segment is configured as a tag filter, the cacheline address selects a set of tag filter entries, and the tags in each of those entries are compared against the address. If the address matches the tag in an entry *and* either the owner pointer field is valid or any presence vector valid bit is set in that entry, the lookup results in a **hit** for that segment; otherwise, the lookup results in a **miss** for that segment. If a directory segment is associated with the initiating agent, a miss may require an entry to be allocated in that segment and may require a recall transaction to be issued to do so (see Section 5.4.2).

How the owner lookup vector and sharer lookup vector bits are populated for the caching agents associated with a directory segment depends on whether the lookup hits or misses and on the tag filter configuration and state. The type of tag filter determines whether cacheline ownership and cacheline validity are *true*, *uncertain*, or *false* for the caching agents associated with the directory segment. The bullets below describe how the lookup results are generated for each type of tag filter.

Presence Vector Directory Segment

Hit (at least one presence vector valid bit is set):

- Owner lookup vector – set for each caching agent whose corresponding precise or coarse presence vector valid bit is set (for precise and coarse, cacheline ownership is uncertain); clear otherwise (cacheline ownership is false)
- Sharer lookup vector – set for each caching agent whose corresponding precise or coarse presence vector valid bit is set (for precise, cacheline validity is true, and for coarse, cacheline validity is uncertain); clear otherwise (cacheline validity is false)

Miss (all presence vector valid bits are clear):

- Owner lookup vector – clear for all caching agents (cacheline ownership is false)
- Sharer lookup vector – clear for all caching agents (cacheline validity is false)

Note

For a given caching agent in a presence vector directory segment, the values of the owner lookup vector bit and the sharer lookup vector bit may be {set, set} or {clear, clear}, and the resulting cacheline ownership and cacheline validity states are {uncertain, true}, {uncertain, uncertain}, or {false, false}, respectively.

Owner-Sharer Directory Segment

Hit (the owner pointer field is valid or at least one presence vector valid bit is set):

- Owner lookup vector – set for the caching agent indicated by the owner pointer if the owner pointer field is valid (cacheline ownership is true and cacheline validity is true); clear otherwise (cacheline ownership is false)
- Sharer lookup vector – set for each caching agent whose corresponding precise or coarse sharer vector valid bit is set and whose corresponding owner lookup vector bit is clear (for precise, cacheline validity is true, and for coarse, cacheline validity is uncertain); clear otherwise (cacheline validity is false)

Miss (the owner pointer field is invalid and all presence vector valid bits are clear):

- Owner lookup vector – clear for all caching agents (cacheline ownership is false)
- Sharer lookup vector – clear for all caching agents (cacheline validity is false)

For tag filters, Table 10-3 and Table 10-4 summarize the states of the owner lookup vector (**OL[c]**) and sharer lookup vector (**SL[c]**) bits for a given caching agent with CacheID **c** and their meaning with respect to cacheline ownership and cacheline validity, respectively, for that agent. Cells marked N/A and values in parentheses represent combinations that are not allowed on a system directory lookup; however, values in parentheses can be reached based on cacheline ownership filtering described in Section 5.6. At most one caching agent in the system directory lookup result may have a value indicated with an asterisk.

Table 10-3: Cacheline Ownership and Validity for a Presence Vector Filter on Lookup

OL[c]	SL[c]	Vector Valid Bit	Null Ownership	Null Validity
0	0	X	False	False
0	1	Precise	(False)	(True)

Table 10-3: Cacheline Ownership and Validity for a Presence Vector Filter on Lookup

OL[c]	SL[c]	Vector Valid Bit	Null Ownership	Null Validity
		Coarse	(False)	(Uncertain)
1	0	X	N/A	N/A
1	1	Precise	Uncertain	True
		Coarse	Uncertain	Uncertain

Table 10-4: Cacheline Ownership and Validity for a Presence Vector Filter on Commit

OL[c]	SL[c]	Vector Valid Bit	Null Ownership	Null Validity
0	0	X	False	False
0	1	Precise	False	True
		Coarse	False	Uncertain
1	0	X	N/A	N/A

For a caching agent represented by a coarse vector valid bit, cacheline validity is *uncertain*. As a result, from a snoop perspective, such a caching agent is treated as if the agent is associated with a null filter when the corresponding vector valid bit is set.

10.2.5 Allocations

For the set of cacheline addresses associated with a directory segment, the tag filter entries in the directory segment must always represent a superset of the cacheline copies that may be present in the caches of the caching agents associated with the directory segment. In other words, if a copy of a cacheline with a cacheline address associated with a given directory segment is present in the cache of a caching agent associated with the same segment, a corresponding directory entry *must* be present in the tag filter of the directory segment.

If the CacheID is *defined* for the initiating agent, a miss in the directory segment associated with the initiating agent requires the allocation of an entry in the tag filter for certain coherent read or coherent clean transactions. The directory segment attempts to reuse an invalid entry in the tag filter, but if one cannot be found, the directory segment selects a valid entry for replacement. Once the entry has been selected, the directory segment presents a recall transaction on the recall interface of the directory partition, and the entry in the directory segment becomes available to the protocol transaction that requires the allocation. At the completion of the protocol transaction, the entry is committed with the results from the transaction.

Once the tag filter entry has been selected, the recall transaction may proceed in parallel with the protocol transaction that requires the allocation.

If the CacheID is *undefined* for the initiating agent, the initiating agent is *not* associated with any directory segment, and the allocation of a system directory entry is not performed.

Note

There are no requirements related to the processing or the completion of the recall transaction and the protocol transaction with respect to each other. The two transactions share resources but are otherwise architecturally independent.

Note

The recall transaction only invalidates the caching agents associated with the directory segment performing the replacement. Caching agents associated with other directory segments are not invalidated by the recall transaction.

10.2.6 Commits

If the tag filter state for a directory segment changed over the course of the protocol transaction, the new tag filter state is written into the tag filter storage. The new state of the tag filter entry depends on the state of the owner commit vector, the state of the sharer commit vector, and the configuration of the directory segment. The following bullets describe the new tag filter entry state for each type of directory segment.

Presence Vector Directory Segment

If, for a given caching agent, the bit in either the owner commit vector or the sharer commit-vector is *set*, i.e., cacheline validity is *true* for the agent, the precise or coarse presence vector valid bit corresponding to that caching agent is set

Owner-Sharer Directory Segment

If, for a given caching agent, the bit in the sharer commit vector is *set* and the bit in the owner commit vector is *clear*, i.e., cacheline ownership is *false* and cacheline validity is *true* for the agent, the precise or coarse presence vector valid bit corresponding to that caching agent is set

If, for a given caching agent, the bit in the sharer commit vector is *clear* and the bit in the owner commit vector is *set*, i.e., cacheline ownership is *true* and cacheline validity is *true* for the agent, the owner pointer field is set to a valid encoding that indicates the caching agent is the owner

For tag filters, Table 10-5 and Table 10-6 summarize the states of the owner commit vector (**OL[c]**) and sharer commit vector (**SL[c]**) bits for a given caching agent with CacheID **c** and their meaning with respect to cacheline ownership and cacheline validity, respectively, for that agent. Cells marked *N/A* represent combinations that are not allowed on a system directory commit, and values in brackets can only be reached by the initiating caching agent. At most one caching agent in the system directory commit result may have a value indicated with an asterisk.

Table 10-5: Cacheline Ownership and Validity for a Presence Vector Filter on Commit

OL[c]	SL[c]	Vector Valid Bit	Null Ownership	Null Validity
0	0	X	False	False
0	1	X	False	True
1	0	X	[True*]	[True*]
1	1	X	Uncertain	True

Table 10-6: Cacheline Ownership and Validity for a Owner-Sharer Filter on Commit

OL[c]	SL[c]	Vector Valid Bit	Null Ownership	Null Validity
0	0	X	False	False
0	1	X	False	True
1	0	X	True*	True*
1	1	X	N/A	N/A

10.2.7 Deallocation

If, at the completion of a protocol transaction, all the owner commit vector bits and the sharer commit vector bits are *clear* for the caching agents associated with a directory segment, the tag filter entry becomes invalid and may be allocated for a subsequent protocol transaction.

10.3 Ownership Filtering

The state of cacheline ownership for caching agents across all directory segments filters the number and type of snoop messages further. If cacheline ownership is *true* for any caching agent, cacheline ownership and the owner lookup vector bits for other caching agents *must* be modified as follows:

- If cacheline validity is *true* or *uncertain* for a caching agent, cacheline ownership becomes *false* for that caching agent, and the corresponding owner lookup vector bit becomes *clear*
- If cacheline validity is *false* for a caching agent, cacheline ownership *must* be *false* for that caching agent, and the corresponding owner lookup vector bit *must* be *clear*

Example

If a system directory lookup in an owner-sharer directory segment indicates a hit and the owner lookup vector is non-zero, cacheline ownership is *true* for a caching agent associated with the directory segment. In this case, the owner lookup vector bits for all other caching agents are cleared.

10.4 Coherency Processing in DCE

The DCE transport layer passes CMDreq messages in the skid buffer to the coherency protocol layer processing. To accept and process a CMDreq message, the protocol layer allocates resources in an Active Transaction Table (**ATT**) to store the fields from the message.

The ATT is divided into control resources (**ATT-Ctrl**) and snoop message resources (**ATT-Snp**). All protocol transactions allocate an ATT-Ctrl resource, and if one is not available, the protocol layer does not accept any new protocol coherent transactions until a resource becomes available.

Note

For the time being, this specification assumes that the number of ATT-Snp resources equals the number of ATT-Ctrl resources.

Once an ATT-Ctrl resource is available, the DCE protocol layer performs a system directory lookup, which determines the resources required to process a new coherent transaction. In some cases, the system directory lookup may indicate that a recall transaction is required. The DCE protocol layer also performs an address conflict check for coherent and recall transactions tracked by the ATT, which ensures that coherent and recall transactions to the same cacheline address are serialized and are performed atomically.

A coherent or recall transaction becomes **active** in the ATT once the transaction has met the address conflict requirements described below; until then, the transaction is **pending**. If a given cacheline address is tracked by the ATT, one (and only one) transaction for that cacheline address *must* be active at all times. Once active, a transaction remains active until its ATT-Ctrl resource has been deallocated, at which point the transaction has **completed** with respect to the DCE.

The following subsections discuss the various protocol coherent transaction and recall transaction processing requirements in more detail.

10.4.1 System Directory Lookup

To perform a system directory lookup, the DCE protocol layer presents the cacheline address from the CMDreq message to the system directory. If *defined* for the initiating agent, the CacheID, mapped from the value of the **InitiatorId** field in the CMDreq message, is also presented. The system directory lookup determines the snoop messages and associated resources required to complete the coherence operations for a transaction. For a coherent transaction, the lookup identifies the required set of snoop messages, which may include any number of owner snoop messages and/or any number of sharer snoop messages. The system directory lookup may also be used to determine whether to issue a memory hint message.

If any snoop messages are required to complete a transaction, the DCE issues the snoop messages to the appropriate AIUs. Once the snoop messages have completed, the DCE issues a **State Reply** message (**STRreq**) to the requesting AIU, and the DCE may issue a memory directive message to the Home DMI. If the DCE does not have the resources available to issue the necessary messages, a coherent transaction may be stalled until the resources become available.

Once all the required protocol messages for a coherent transaction have completed, the DCE creates the system directory commit result and *must* perform a system directory commit to update the system directory storage if the system directory commit result differs from the system directory lookup result.

10.4.2 Recall Transactions

For a CmdRdCln, CmdRdVld, CmdRdUnq, or CmdClnUnq message, the system directory may indicate that a recall transaction is required to allocate an entry in the directory; additionally, a recall transaction may be initiated by software to perform directory maintenance. A recall transaction consists of SnpRecall messages issued by the DCE and has similar requirements to a coherent transaction. In particular, the recall transaction becomes active or is pending based on the address conflict check. The DCE may stall, but is *not* required to stall, new coherent transactions until the recall transaction becomes active or has completed.

Once all the required protocol messages for a recall transaction have completed, the DCE creates the system directory commit result and *must* perform a system directory commit to update the appropriate system directory segment if not doing so would leave a valid directory entry in that segment.

10.4.3 Address Collision Check

The **address collision check** compares the cacheline address of a new transaction with the cacheline addresses of the coherent and recall transactions in the ATT. If the cacheline address of the new transaction matches the cacheline address of a transaction in the ATT, the two transactions are said to **collide** with each other. Such transactions get cast into a strict sequence within the ATT, referred to as the *coherence order*. Transactions are processed in that order.

A new arriving transaction is appended at the end of the correct coherence order at the time it is allocated into the ATT. (See Section 6.1.2.)

For a given cacheline address, the ATT may contain two or more colliding transactions. The following requirements apply to *colliding* coherent and recall transactions:

- A new transaction is pending if the new transaction collides with an active transaction; otherwise, the new transaction becomes active.
- A colliding pending transaction *does not* become active before a colliding active transaction has completed.
- The next colliding pending transaction in the corresponding coherence order *must* become active once a colliding active transaction has completed.

Note

The above serialization behavior ensures that at most one set of snoop messages is active for a given cacheline.

The implementation of a DCE *must* ensure that all colliding pending transactions eventually become active. This is ensured by executing transactions strictly in the sequence of the coherence order.

10.4.4 System Directory State

Transactions and recall transactions within a coherence order *must* appear to perform directory operations sequentially and atomically with respect to each other, i.e. the system directory commit result of a completed transaction *must* be observed in the system directory lookup result of any subsequent colliding transactions.

Note

A coherent transaction operates on one or more directory segments, while a recall transaction operates on only one.

Once an active transaction has completed, the ATT transfers the system directory commit result to any pending transactions in the ATT directly or indirectly through the system directory storage. If no further pending transactions for the cacheline address are present in the ATT, the DCE performs a system directory commit, if required.

Mapping of native Transactions to CCMP Request messages

This chapter describes the capabilities and structure of an Ncore 3 message transport system.

A.1 CMDreq mappings

Table 10-7: CCMP Opcode mapping and encoding

CCMP Operation			Native Operation / Request Transaction Mappings			
			CHI-E / CHI-B		ACE ACE-Lite ACE5-Lite	
Mnemonic	CMDReq	Code [7:0]	Mnemonic	Code [5:0]	Mnemonic	BAR DOMAIN SNOOP
ReadNITC	RdNITC	0x07	ReadOnce	0x03	ReadOnce	R: 0 D: {01/10} S: 0000
ReadClean	RdCln	0x01	ReadClean	0x02	ReadClean	R: 0 D: {01/10} S: 0010
ReadValid	RdVld	0x03	ReadShared	0x01	ReadShared	R: 0 D: {01/10} S: 0001
ReadUnique	RdUnq	0x04	ReadUnique	0x07	ReadUnique	R: 0 D: {01/10} S: 0111
ReadNotShDirty	RdNShDty	0x02	ReadNot-SharedDirty	0x26	ReadNot-SharedDirty	R: 0 D: {01/10} S: 0011
ReadNC	RdNC	0x0B	ReadNoSnp	0x04	ReadNoSnoop	R: 0 D: {00/11} S: 0000
CleanUnique	ClnUnq	0x05	CleanUnique	0x0B	CleanUnique	R: 0 D: {01/10} S: 1011
CleanValid	ClnVld	0x08	CleanShared	0x08	CleanShared	R: 0 D: {00/01/10} S: 1000
CleanSharedPersist	ClnShPsist	0x28	CleanSharedPersist	0x27	ALE: CleanShared-Persist	R: 0 D: {00/01/10} S: 1010
CleanInvalid	ClnInv	0x09	CleanInvalid	0x09	CleanInvalid	R: 0 D: {00/01/10} S: 1001
MakeInvalid	MkInv	0x0A	MakeInvalid	0x0A	MakeInvalid	R: 0 D: {00/01/10} S: 1101
MakeUnique	MkUnq	0x06	MakeUnique	0x0C	MakeUnique	R: 0 D: {01/10} S: 1100
Evict	Evict	0x17	Evict	0x0D	Evict ^b	W: 0 D: {01/10} S: 100
WriteUniquePtl	WrUnqPtl	0x10	WriteUniquePtl	0x18	WriteUnique	W: 0 D: {01/10} S: 000
WriteUniqueFull	WrUnqFull	0x11	WriteUniqueFull	0x19	WriteLineUnique	W: 0 D: {01/10} S: 001

Table 10-7: CCMP Opcode mapping and encoding

CCMP Operation			Native Operation / Request Transaction Mappings			
			CHI-E / CHI-B		ACE ACE-Lite ACE5-Lite	
Mnemonic	CMDReq	Code [7:0]	Mnemonic	Code [5:0]	Mnemonic	BAR DOMAIN SNOOP
WriteCleanPtl	WrClnPtl	0x19	(WriteCleanPtl)	(0x16)	NA	NA
WriteCleanFull	WrClnFull	0x15	WriteCleanFull	0x17	WriteClean	W: 0 D: {00/01/10} S: 010
WriteBackPtl	WrBkPtl	0x18	WriteBackPtl	0x1A	NA	W: 0 D: {00/01/10} S: 011
WriteBackFull	WrBkFull	0x14	WriteBackFull	0x1B	WriteBack ^b	W: 0 D: {00/01/10} S: 011
WriteEvict	WrEvict	0x16	WriteEvictFull	0x15	WriteEvict ^b	W: 0 D: {00/01/10} S: 101
WriteNCPtl	WrNCPtl	0x20	WriteNoSnpPtl	0x1C	WriteNoSnp	W: 0 D: {00/11} S: 000
WriteNCFull	WrNCFull	0x21	WriteNoSnpFull	0x1D	WriteNoSnp	W: 0 D: {00/11} S: 000
WriteStashFull	WrStshFull	0x22	WriteUnique StashFull	0x20	ALE: WriteUnique-FullStash	W: 0 D: {01/10} S: {1 001}
LoadCchShared	LdCchShd	0x24	StashOnce- Shared	0x22	ALE: StashOnceShared	W: 0 D: {01/10} S: {1 100}
LoadCchUnique	LdCchUnq	0x25	StashOn- ceUnique	0x23	ALE: StashOnceUnique	W: 0 D: {01/10} S: {1 101}
ReadNITCClean- Invalid	RdNITCCI	0x26	ReadOnce- CleanInvalid	0x24	ReadOnce-CleanInvalid	R: 0 D: {00/01/10} S: {0 100}
ReadNITCMake- Invalid	RdNITCMI	0x27	ReadOnce- MakeInvalid	0x25	ReadOnce- CleanInvalid	R: 0 D: {00/01/10} S: {0 101}
Read Atomic	RdAtm	0x13	AtomicLoad	0x28- 0x2F	ALE: AtomicLoad	W: 0 D: {01/10} S: {1 001} ^a
Write Atomic	WrAtm	0x12	AtomicStore	0x30- 0x37	ALE: AtomicStore	W: 0 D: {01/10} S: {1 001} ^a
Compare And Swap Atomic	CompAtm	0x2A	AtomicComare	0x39	ALE: AtomicCompare	W: 0 D: {01/10} S: {1 001} ^a
Swap Atomic	SwapAtm	0x29	AtomicSwap	0x38	ALE: AtomicSwap	W: 0 D: {01/10} S: {1 001} ^a
Prefetch	Pref	0x2B	PrefetchTgt	0x3A	NA	NA

Table 10-7: CCMP Opcode mapping and encoding

CCMP Operation			Native Operation / Request Transaction Mappings			
			CHI-E / CHI-B		ACE ACE-Lite ACE5-Lite	
Mnemonic	CMDReq	Code [7:0]	Mnemonic	Code [5:0]	Mnemonic	BAR DOMAIN SNOOP
DVMOp	DVMMsg	0x0F	DvmOp	0x14	DVM Message	R: 0 D: {01/10} S: 1111

Notes:

- a. AtomicLoad: AWATOP[5:0] = 0b10exxx; AtomicStore: AWATOP[5:0] = 0b01exxx; AtomicCompare: AWATOP[5:0] = 0b11001; AtomicSwap: AWATOP[5:0] = 0b11000. “e” indicates Endianess of data. In AXI / ACE- Lite-E, AWATOP[5:0] = 0b000000 for all non-atomics.
- b. In the case of ACE: WriteBack & WriteEvict map to concerto WriteNC command, they may be followed by an update command if the transaction was a coherent transaction. The Evict command does not have associated data and will only issue update command.

Table 10-8: Native operation to CCMP Operations mappings

CCMP Operation Mnemonic	CCMP CMDreq	Native Operation / Request Transaction Mappings
ReadNITC	CmdRdNITC	ACE, ACE-Lite, AC-Lite-E: ReadOnce
ReadClean	CmdRdCln	<p>Read Clean Obtain the latest <i>clean</i> coherent copy of a cacheline to be installed into an agent's cache.</p> <p>Note: This operation corresponds to the ACE and CHI ReadClean and ReadNotSharedDirty operations.</p> <p>Final Cacheline States: Requester: SC or UC. Snooper: I, SC, SD.</p>
ReadValid	CmdRdVld	<p>Read Valid Obtain a copy of a cacheline <i>in any valid state</i>.</p> <p>Note: This operation corresponds to the ACE and CHI ReadShared operations.</p> <p>Final Cacheline States: Requester: SC or UC. Snooper: I, SC, SD.</p>
ReadUnique	CmdRdUnq	<p>Read Unique Obtain a copy of a cacheline in <i>unique</i> state, requiring all other agents to invalidate all copies of the cacheline.</p> <p>Note: This operation corresponds to the ACE and CHI ReadUnique operation.</p> <p>Final Cacheline States: Requester: UC or UD. Snooper: I.</p>
ReadNotSh-Dirty	CmdRdNShDty	<p>Read with no Shared-Dirty Obtain a copy of a cacheline in <i>non-Shared-Dirty</i> state.</p> <p>Note: This operation corresponds to the ACE and CHI ReadUnique operation.</p> <p>Final Cacheline States: Requester: SC, UC or UD. Snooper: I, SC, or SD.</p>
ReadNC	CmdRdNC	<p>Non-coherent Read Obtain an Atomic Block up to a cacheline non-coherently, i.e., without snooping other agents in the system. Access may be to system memory or peripheral storage. Since the data is accessed non-coherently, it is not tracked by the directory.</p> <p>Note: This operation corresponds to the ACE and CHI Read- No Snoop operation.</p> <p>Final Cacheline States: Requester: I. Snooper: NA.</p>

Table 10-8: Native operation to CCMP Operations mappings

CCMP Operation Mnemonic	CCMP CMDreq	Native Operation / Request Transaction Mappings
CleanUnique	CmdClnUnq	<p>Clean Unique Create a unique copy of a cacheline in the initiating agent when the cacheline is in the invalid (IX) or Shared (SC or SD) state in the agent. All other copies of the cacheline are invalidated. No data is transferred to the initiating agent during this operation. So, the starting IX state results in the UCE state, and SC and SD result in UC and UD respectively.</p> <p>Final Cacheline States: Requester: UCE, UC or UD. Snooper: I.</p>
CleanValid	CmdClnVld	<p>Ensure that dirty copy of the cacheline in any of the caches in the coherency domain is written back to memory to the point of coherent visibility. If a snooped agent has a dirty copy of a cacheline in its cache, that agent <i>must</i> update the system memory. The agents may retain a clean copy of the cacheline.</p> <p>Note: In Ncore 3, home location of the cacheline in system memory is updated.</p> <p>Final Cacheline States: Requester: I, SC, UC. Snooper: I, SC, UC.</p>
CleanValid	CmdClnVld	<p>Ensure that dirty copy of the cacheline in any of the caches in the coherency domain is written back to memory to the point of coherent visibility. If a snooped agent has a dirty copy of a cacheline in its cache, that agent <i>must</i> update the system memory. The agents may retain a clean copy of the cacheline.</p> <p>Note: In Ncore 3, home location of the cacheline in system memory is updated.</p> <p>Final Cacheline States: Requester: I, SC, UC. Snooper: I, SC, UC.</p>
CleanShared-Persist	CmdClnShPsist	<p>Clean Valid to Persistent Storage Dirty copy of the cacheline in any of the caches in the coherency domain is written back to memory to the point of coherent visibility. If a snooped agent has a dirty copy of a cacheline in its cache, that agent <i>must</i> update the system memory. The agents may retain a clean copy of the cacheline. Furthermore, the dirty data must be written to the copy of the cacheline in persistent (or non-volatile) storage, if available.</p> <p>Note: In Ncore 3, home location of the cacheline in system memory is updated.</p> <p>Final Cacheline States: Requester: I, SC, UC. Snooper: I, SC, UC.</p>
CleanInvalid	CmdClnInv	<p>Clean Invalid Dirty copy of the cacheline in any of the caches in the coherency domain is written back to memory to the point of coherent visibility. If a snooped agent has a copy of a cacheline in a valid and dirty state, that agent <i>must</i> update system memory. In addition, snooped agents <i>must</i> invalidate all copies of the cacheline.</p> <p>Final Cacheline States: Requester: I. Snooper: I.</p>

Table 10-8: Native operation to CCMP Operations mappings

CCMP Operation Mnemonic	CCMP CMDreq	Native Operation / Request Transaction Mappings
MakeInvalid	CMdMkInv	<p>Make Invalid Invalidate the copy of the cacheline in situ for all agents in the coherency domain, without updating the system memory.</p> <p>Final Cacheline States: Requester: I. Snooper: I.</p>
MakeUnique	CmdMkUnq	<p>Make Unique Make copy of the cacheline in the requester unique, in place, for a full replacement of its contents. Also, invalidate the copies of the cacheline, in situ, for all other agents in the coherency domain, without updating the system memory. No data is transferred to the requester.</p> <p>Final Cacheline States: Requester: UD. Snooper: I.</p>
WriteUniquePtl	CmdWrUnqPtl	<p>Write Unique Partial Cacheline Write Atomic Block (AB) amount (up to a cacheline) of new data to the cacheline in system memory when the cacheline is invalid in the requesting agent. The actual new bytes being written can be sparse within the AB and are indicated via Byte Enables accompanying the data. If a snooped agent has the cacheline in partial or full dirty state, the dirty bytes are written to memory before the new data is written to memory. The new data associated with the operation overrides, on a per byte basis, the dirty data supplied by the agent and existing bytes of the cacheline. Also, invalidate the copies of the cacheline, in situ, for all other agents in the coherency domain.</p> <p>Final Cacheline States: Requester: I. Snooper: I.</p>
WriteUniqueFull	CmdWrUnqFull	<p>Write Unique Full Cacheline Write a whole cacheline worth of new data to the cacheline in system memory when the cacheline is invalid in the requesting agent. All of the Byte Enables accompanying the data are set. Also, invalidate the copies of the cacheline, including any dirty copy, in situ, for all other agents in the coherency domain.</p> <p>Final Cacheline States: Requester: I. Snooper: I.</p>
WriteCleanPtl	CmdWrClnPtl	<p>Write Clean Partial Cacheline Write Atomic Block (AB) amount (up to a cacheline) of cacheline data to system memory when the partial cacheline is dirty in the requesting agent. The actual new bytes being written can be sparse within the AB and are indicated via Byte Enables accompanying the data. The requester may retain the cacheline in clean but empty state (UCE) or invalidate it.</p> <p>Note: This operation corresponds to the ACE and CHI-AE WriteCleanPtl operation. This operation is not supported in CHI-B-(and later versions of CHI protocol).</p> <p>Final Cacheline States: Requester: I, UCE. Snooper: I.</p>

Table 10-8: Native operation to CCMP Operations mappings

CCMP Operation Mnemonic	CCMP CMDreq	Native Operation / Request Transaction Mappings
WriteClean-Full	CmdWrClnFull	<p>Write Clean Full Cacheline Write a full cacheline worth data to system memory when the cacheline is dirty in the requesting agent. Since a full cache line is being written, all Byte Enables accompanying the data are asserted. The requester may retain the cacheline in clean state (UC).</p> <p>Final Cacheline States: Requester: SC, UC. Snooper: I.</p>
WriteBackPtl	CmdWrBkPtl	<p>Write Back Partial Cacheline Write Atomic Block (AB) amount (up to a cacheline) of cacheline data to system memory when the partial cacheline is dirty in the requesting agent. The actual bytes being written can be sparse within the AB and are indicated via Byte Enables accompanying the data. The requester invalidates its copy of the cacheline.</p> <p>Note: This operation corresponds to the ACE and CHI-E/B WriteBackPtl operation.</p> <p>Final Cacheline States: Requester: I. Snooper: I.</p>
WriteBackFull	CmdWrBkFull	<p>Write Back Full Cacheline Write a full cacheline worth data to system memory when the cacheline is dirty in the requesting agent. Since a full cache line is being written, all Byte Enables accompanying the data are asserted. The requester invalidates its copy of the cacheline.</p> <p>Note: This operation corresponds to the ACE and CHI-E/B WriteBackFull operation.</p> <p>Final Cacheline States: Requester: I. Snooper: I.</p>
WriteEvict	CmdWrEvict	<p>Write Evict full cacheline Write a full cacheline worth data to system memory when the cacheline is in UC state in the requesting agent. Since a full cache line is being written, all Byte Enables accompanying the data are asserted. The requester invalidates its copy of the cacheline.</p> <p>Note: This operation corresponds to the ACE and CHI-E/B WriteEvictFull operation.</p> <p>Final Cacheline States: Requester: I. Snooper: I.</p>
WriteNCPtl	CmdWrNCPtl	<p>Write Partial Cacheline non-coherently Write Atomic Block (AB) amount (up to a cacheline) of cacheline data non-coherently to system memory or peripheral storage. The actual bytes being written can be sparse within the AB and are indicated via Byte Enables accompanying the data.</p> <p>Note: This operation corresponds to the ACE and CHI-E/B WriteNoSnpPtl operation.</p> <p>Final Cacheline States: Requester: Not applicable. Snooper: Not applicable.</p>

Table 10-8: Native operation to CCMP Operations mappings

CCMP Operation Mnemonic	CCMP CMDreq	Native Operation / Request Transaction Mappings
WriteNCFull	CmdWrNCFull	<p>Write Full Cacheline non-coherently Write a full cacheline worth data non-coherently to system memory or peripheral storage. Since a full cache line is being written, all Byte Enables accompanying the data are asserted.</p> <p>Note: This operation corresponds to the ACE and CHI-E/B WriteNoSnpFull operation.</p> <p>Final Cacheline States: Requester: Not applicable. Snooper: Not applicable.</p>
WriteStashPtl	CmdWrStshPtl	<p>Stash via Write Unique Partial Cacheline Write Atomic Block (AB) amount (up to a cacheline) of new data to the cacheline in system memory when the cacheline is invalid in the requesting agent. The actual new bytes being written can be sparse within the AB and are indicated via Byte Enables accompanying the data. If a snooped agent has the cacheline in partial or full dirty state, the dirty bytes are written to coherent visibility. The new data associated with the operation overrides, on a per byte basis, the dirty data supplied by the agent and existing bytes of the cacheline. Also, invalidate the copies of the cacheline, in situ, for all other agents in the coherency domain. Also, a target agent is identified so that, snoop to target agent contains hint to ask permission to install the cacheline in cache. If accepted, the full cacheline is installed in UD state in the targeted agent instead of writing to system memory. If not accepted, the updated cacheline is written to memory will allocate hint to system cache. If a specific target is not indicated in the operation, the cacheline is written to system memory with allocate hint to system cache.</p> <p>Final Cacheline States: Requester: I. Snooper: I. Target agent: UD if stash is accepted; I otherwise.</p>
WriteStashFull	CmdWrStsh-Full	<p>Stash via Write Unique Full Cacheline Write a whole cacheline worth of new data to the cacheline in system memory when the cacheline is invalid in the requesting agent. All of the Byte Enables accompanying the data are set. Operation invalidates the copies of the cacheline, including any dirty copy, in situ, for all other agents in the coherency domain. Also, a target agent is identified so that, snoop to target agent contains hint to ask permission to install the cacheline in its cache. If accepted, the full cacheline is installed in UD state in the target agent instead of writing to system memory. If not accepted, the updated cacheline is written to memory will allocate hint to system cache. If a specific target is not indicated in the operation, the cacheline is written to system memory with allocate hint to system cache.</p> <p>Final Cacheline States: Requester: I. Snooper: I. Target agent: UD if stash is accepted; I otherwise.</p>

Table 10-8: Native operation to CCMP Operations mappings

CCMP Operation Mnemonic	CCMP CMDreq	Native Operation / Request Transaction Mappings
LoadCch-Shared	CmdLdCchShd	<p>Load a cacheline in a target agent's cache Snoop a target agent to ask permission to install the cacheline in its cache. If accepted, the full cacheline is installed in a shared or unique state in the target agent. If not accepted, the cacheline may be read from memory and allocated into the system cache. If a specific target is not indicated in the operation, the cacheline may be read from the memory and allocated into the system cache.</p> <p>Final Cacheline States: Requester: I. Snooper: I, SC, or SD. Target agent: SC, UC or UD if stash is accepted; I otherwise.</p>
LoadCch-Unique	CmdLdCchUnq	<p>Load a cacheline in a target agent's cache Snoop a target agent to ask permission to install the cacheline in its cache. If accepted, the full cacheline is installed in a shared or unique state in the target agent. If not accepted, the cacheline may be read from memory and allocated into the system cache. If a specific target is not indicated in the operation, the cacheline may be read from the memory and allocated into the system cache.</p> <p>Final Cacheline States: Requester: I. Snooper: I. Target agent: UC or UD if stash is accepted; I otherwise.</p>
ReadNITC-CleanInvalid	CmdRdNITCCI	<p>Read with No Intention to Cache with Clean and Invalidate. Obtain the latest snapshot of the cacheline. The snapshot is not installed in the CCMP coherency domain. Thus, the snapshot itself is no longer kept coherent after the operation. After the snapshot is taken, dirty copy of the cacheline in any of the caches in the coherency domain is written back to memory to the point of coherent visibility. In addition, snooped agents <i>must</i> invalidate all copies of the cacheline.</p> <p>Final Cacheline States: Requester: I. Snooper: I Note: This transaction corresponds to the CHI-B ReadOnceCleanInvalid transaction</p>
ReadNITC-MakeInvalid	CmdRd-NITCMI	<p>Read with No Intention to Cache with Make Invalidate. Obtain the latest snapshot of the cacheline. The snapshot is not installed in the CCMP coherency domain. Thus, the snapshot itself is no longer kept coherent after the operation. After the snapshot is taken, all copies of the cacheline are invalidated in situ, without updating system memory.</p> <p>Final Cacheline States: Requester: I. Snooper: I Note: This transaction corresponds to the CHI-B ReadOnceMakeInvalid transaction</p>

Table 10-8: Native operation to CCMP Operations mappings

CCMP Operation Mnemonic	CCMP CMDreq	Native Operation / Request Transaction Mappings
ReadAtomic	CmdRdAtm	<p>Read Atomic</p> <p>Perform an atomic update to an aligned block of data of size 1, 2, 4, or 8 bytes and return the original value of the block.</p> <p>Perform an atomic update (Read-Modify-Write) of the block specified in the operation with the operand data accompanying the operation. For efficiency, the update is performed close to the home location of the data.</p> <p>The original value of the aligned block is returned for the Read. The data returned is not cached by the requester.</p> <p>For a coherent Read Atomic operation, coherency phase is performed first in which, the addressed location is cleaned to system memory first and the cached copies invalidated, before conducting the Atomic Read operation.</p> <p>Final Cacheline States:</p> <p>Requester: I.</p> <p>Snooper: I</p> <p>Note: CCMP supports Read Atomic only for addresses located in system memory</p>
WriteAtomic	CmdWrAtm	<p>Write Atomic</p> <p>Perform an atomic update to an aligned block of data of size 1, 2, 4, or 8 bytes.</p> <p>Perform an atomic update (Read-Modify-Write) of the block specified in the operation with the operand data accompanying the operation. For efficiency, the update is performed close to the home location of the data.</p> <p>No data is returned for the Write Atomic.</p> <p>For a coherent Write Atomic operation, coherency phase is performed first in which, the addressed location is cleaned to system memory first and the cached copies invalidated, before conducting the Write Atomic operation.</p> <p>Final Cacheline States:</p> <p>Requester: I.</p> <p>Snooper: I</p> <p>Note: CCMP supports Write Atomic only for addresses located in system memory</p>
Compare And Swap Atomic	CmdCompAtm	<p>Compare and Swap Atomic</p> <p>The operation specifies two data values: an operand for the compare and the second for the swap.</p> <p>Perform a compare of an aligned block of data of size 1, 2, 4, or 8 bytes in memory with the operand data of the same size specified by the operation. If a match occurs, then the second operand is swapped with the second operand.</p> <p>For efficiency, the compare and swap atomic is performed close to the home location of the data.</p> <p>The original value of the aligned block is returned for the operation. The data returned is not cached by the requester.</p> <p>For a coherent Compare and Swap Atomic operation, coherency phase is performed first in which, the addressed location is cleaned to system memory first and the cached copies invalidated, before conducting the Compare and SwapAtomic operation.</p> <p>Note: CCMP supports Compare and Swap atomic only for addresses located in system memory.</p> <p>Final Cacheline States:</p> <p>Requester: I.</p> <p>Snooper: I</p>

Table 10-8: Native operation to CCMP Operations mappings

CCMP Operation Mnemonic	CCMP CMDreq	Native Operation / Request Transaction Mappings
SwapAtomic	CmdSwapAtm	<p>Swap Atomic Perform a swap atomic of an aligned block of data of size 1, 2, 4, or 8 bytes in memory with the operand data of the same size specified by the operation. For efficiency, the swap atomic is performed close to the home location of the data. The original value of the aligned block is returned for the operation. The data returned is not cached by the requester. For a coherent Swap Atomic operation, coherency phase is performed first in which, the addressed location is cleaned to system memory first and the cached copies invalidated, before conducting the Swap Atomic operation.</p> <p>Note: CCMP supports Swap Atomic only for addresses located in system memory.</p> <p>Final Cacheline States: Requester: I. Snooper: I</p>
Prefetch	CmdPref	<p>Prefetch Prefetch a cacheline from system memory device to system cache.</p> <p>Final Cacheline States: Requester: NA. Snooper: NA.</p>
DVMOp	CmdDVMMMsg	Perform a Distributed Virtual Machine operation