

OpenCAPI 4.0

Transaction Layer

Specification

Version 1.0
16 June 2020

Approved

Approved for Distribution to OpenCAPI Members
Approved for Distribution to Non-Members for Learning Purposes Only

Approved

OpenCAPI 4.0 Transaction Layer Specification

OpenCAPI TL Specification Work Group
OpenCAPI Consortium

Version 1.0 (16 June 2020)

Copyright © OpenCAPI Consortium 2016-2020.

Printed in the United States of America February 5, 2021 .

Use of this document is controlled by the OpenCAPI Consortium License Agreement, which is available at <https://opencapi.org/license/>.

All capitalized terms in the following text have the meanings assigned to them in the OpenCAPI Intellectual Property Rights Policy (the “OpenCAPI IPR Policy”). The full Policy may be found at the OpenCAPI Consortium website.

THE SPECIFICATION IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, ACCURACY, COMPLETENESS AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL LICENSOR, ITS MEMBERS OR ITS CONTRIBUTORS BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THE SPECIFICATION.

OpenCAPI and the OpenCAPI logo design are trademarks of the OpenCAPI Consortium.

Other company, product, and service names may be trademarks or service marks of others.

Abstract

This document details the OpenCAPI TL specification. It is the work product of the OpenCAPI Consortium TL Specification Work Group.

This document is handled in compliance with the requirements outlined in the OpenCAPI Consortium Work Group (WG) process document. Comments, questions, etc. can be submitted to membership@opencapi.org.

Approved

Participants

Brian Allison, IBM, *Chair*

Michael Siegel, IBM, *Technical Editor*

Sanjay Goyal, Microchip

Rick Hagen, NVIDIA

Paul Hartke, Xilinx

Curt Wollbrink, IBM

Contents

List of figures	7
List of tables	8
Revision log	10
About this document	11
Architecture compliance terminology	11
Conventions used in this specification	11
Bit and byte numbering	11
Representation of numbers	12
RTL notation	12
Notes	13
Engineering notes	13
Developer notes	13
Command flows and transaction diagrams	14
Command flow diagrams	14
Transaction diagrams	14
Terms	17
1. Overview	26
1.1 OpenCAPI protocol stack	27
1.2 Host operation modes	28
1.2.1 No attached device (C0, M0)	28
1.2.2 MEM-only mode (C0, M1)	28
1.2.3 Checkout mode (C1, M0)	29
1.2.4 Checkout with MEM (C1, M1)	29
1.2.5 Cache-only mode (C2, M0)	29
1.2.6 Cache + MEM mode (C2, M1)	29
1.3 AFU _{C2}	30
1.3.1 Host proxy cache	30
1.3.2 AFU _{C2} model	30
1.3.2.1 host_tag database	30
1.3.2.2 L1 EA cache directory	31
1.3.2.3 data cache	31
1.3.3 AFU cache states	32
1.3.4 AFU Cache state transition reporting, initiation, and characteristics	33
1.3.5 Design considerations when the AFU _{C2} and host cache line sizes are different	38
1.3.5.1 Read commands	38
1.3.5.2 Force evict	38
1.3.5.3 Upgrade state command	38
1.4 Command ordering	39
1.5 Host tags	39
1.5.1 host_tag run-length-capability	41
1.5.2 host_tag update ordering	41

Approved

1.5.2.1 TL and host rules	41
1.5.2.2 TLX and AFU rules	41
1.6 Write fragmentation ordering and atomicity	42
1.6.1 Write fragmentation ordering and atomicity at the host	42
1.6.1.1 Partial write operations	42
1.6.1.2 64-, 128-, 256-byte write operations	42
1.6.2 Write fragmentation ordering and atomicity at the AFU	43
1.6.2.1 Partial write operations	43
1.6.2.2 64-, 128-, 256-byte write operations	43
1.7 OpenCAPI device PA space specification	43
1.7.1 PA-to-RA mapping rules	44
1.8 Address translation	45
1.8.1 Effective to real address translation	45
1.8.2 Translated addresses, AFU ATC, and dot-t commands	45
1.8.2.1 AFU initiated AFU ATC entry invalidation	46
1.8.2.2 Host initiated AFU ATC entry invalidation	47
2. TL and TLX command and response specifications	48
2.1 Handling multiple responses to a single command	53
2.1.1 TLX Read request getting multiple TL responses	54
2.1.2 TLX Write request getting multiple TL responses	54
2.1.3 TL read request getting multiple TLX responses.	55
2.1.4 TL write request getting multiple TLX responses	55
2.2 TL CAPP command packets	57
2.3 TLX AP command packets	82
2.4 TL CAPP response packets	122
2.5 TLX AP response packets	150
3. Virtual channel and data credit pool specification	163
3.1 Virtual channel	164
3.1.1 TLX command and response VC (TLX.vc)	164
3.1.2 TL command and response VC (TL.vc)	164
3.1.3 VC credit count specification	165
3.2 Data credit pool	165
3.2.1 TLX data DCP (TLX.dcp)	165
3.2.2 TL data DCP (TL.dcp)	166
3.2.3 DCP credit count specification	166
3.3 TL Virtual channel and service queues	168
3.3.1 Host TLX command handling	168
3.3.2 Host TLX response handling	170
3.4 TL Presync queues.....	171
3.4.1 TL queuing and service of kill_xlate_done	173
3.5 Device TL virtual channel queues	174
3.6 Virtual channel dependency rules	176
3.6.1 Dependency loop 1 resolution	178
4. The acTag table	180
4.1 acTag table contents	180

Approved

4.2 acTag table access	180
4.2.1 Error cases when accessing the acTag table	180
4.3 acTag entry management	180
5. DL frame format	182
5.1 DL frame control flit (64 bytes)	183
5.1.1 DL content	183
5.1.2 TL command/response content	184
5.1.3 Data transport, order, and alignment	184
5.1.3.1 Data alignment for commands and responses specifying a host_tag field.....	186
6. TL and TLX template specifications	189
6.1 TLX receive and TL transmit template capability specification	191
6.2 TL receive and TLX transmit template capability specification	194
6.3 Control-flit rate capability	195
6.4 Metadata capability	196
7. Error detection	197
7.1 Error events	198
8. OpenCAPI profiles	210
Appendix A. AP (TLX) command transaction diagrams	219
A.1 AFU read with no intent to cache; 128 bytes	219
A.2 TLX read with no intent to cache hits device co-located AFU _{C2} and AFU _{M1}	221
A.3 AFU DMA write; non-posted; 128 bytes	222
A.4 AFU DMA Write hits device co-located AFU _{C2} and AFU _{M1}	225
A.5 AFU DMA Write hits device co-located AFU _{M1}	226
A.6 AFU DMA partial write; non-posted, 8 bytes	227
A.7 AFU Partial read with no intent to cache hits device co-located AFU _{M1}	228
A.8 Translate touch (xlate_touch , ta_req)	229
A.9 Upgrade state	230
A.10 Host tag locking transactions	235
A.11 Castout push	238
Appendix B. CAPP (TL) command transaction diagrams	239
B.1 CAPP memory read; 128 bytes	239
B.2 CAPP memory write; 128 bytes	240

List of figures

Figure 1.	Big- and little-endian comparisons	12
Figure 2.	Command flow example	14
Figure 3.	Example TLX and TL transaction diagram	16
Figure 1-1.	OpenCAPI stack	27
Figure 2-1.	Address translation sequence: xlite_touch	108
Figure 3-1.	TL command flow from the VC queue to the service queue	169
Figure 3-2.	TL command flow from a service queue with a designated presync queue	172
Figure 3-3.	kill_xlate_done TL flow from TLX.vc.3 to host dispatch	174
Figure 3-4.	TLX command and response flow from the VC to the AFU protocol stack	176
Figure 3-5.	VC dependency graph	178
Figure 3-6.	Loop 1 detail	179
Figure A-1.	TLX and TL interaction: rd_wnitc	220
Figure A-2.	TLX rd_wnitc hits AFU _{C2} and AFU _{M1}	221
Figure A-3.	TLX and TL interaction: dma_w	223
Figure A-4.	TLX dma_w hits AFU _{C2} and AFU _{M1}	225
Figure A-5.	TLX dma_w hits host cache and AFU _{M1}	226
Figure A-6.	TL and TLX interaction: dma_pr_w	227
Figure A-7.	TLX pr_rd_wnitc hits AFUM1	228
Figure A-8.	xlite_touch TLX and TL interaction	229
Figure A-9.	upgrade_state TLX and TL interaction	230
Figure A-10.	TLX upgrade_state hits host cache and AFU _{M1}	232
Figure A-11.	TLX upgrade_state hits AFU _{C2} and AFU _{M1}	233
Figure A-12.	TLX upgrade_state hits AFU _{M1} , requires host ATC evict	234
Figure A-13.	host_tag reuse	235
Figure A-14.	host_tag reuse	236
Figure A-15.	host_tag reuse	237
Figure A-16.	castout.push example showing host_tag ordering at the host	238
Figure B-1.	TL and TLX transaction: rd_mem	239
Figure B-2.	TL and TLX transaction: write_mem	240

List of tables

Table 1.	Architecture terms	11
Table 1-1.	Cache state descriptions	32
Table 1-2.	Concurrent host proxy cache (L2) and L1 EA cache states (L1)	33
Table 1-3.	L1 EA Cache state change request and notification	35
Table 1-4.	host_tag run-length-capability definition	41
Table 2-1.	TL and TLX command operands	48
Table 2-2.	The Resp_code specification for xlata_done	58
Table 2-3.	The Resp_code specification for intrp_rdy	59
Table 2-4.	The cmd_flag specification for amo_rd	63
Table 2-5.	The cmd_flag specification for amo_rw	65
Table 2-6.	The cmd_flag specification for amo_w	67
Table 2-7.	The cmd_flag specification for kill_xlate	73
Table 2-8.	The cmd_flag specification for disable_cache	75
Table 2-9.	The cmd_flag specification for enable_cache	76
Table 2-10.	The cmd_flag specification for disable_atc	77
Table 2-11.	The cmd_flag specification for enable_atc	78
Table 2-12.	The cmd_flag specification for amo_rd	89
Table 2-13.	The cmd_flag specification for amo_rw	91
Table 2-14.	The cmd_flag specification for amo_w	93
Table 2-15.	The command flag specification for castout	96
Table 2-16.	The cmd_flag specification for upgrade_state	101
Table 2-17.	The cmd_flag specification for xlata_touch (all forms)	106
Table 2-18.	The cmd_flag specification for sync	121
Table 2-19.	The Resp_code specification for touch_resp	123
Table 2-20.	touch_resp Resp_code use by TLX command	124
Table 2-21.	synonym_detected formation and actions	127
Table 2-22.	The Resp_code specification for read_failed	135
Table 2-23.	read_failed Resp_code use by TLX command	136
Table 2-24.	The Resp_code specification of write_failed	142
Table 2-25.	write_failed Resp_code use by TLX command	143
Table 2-26.	The Resp_code specification for intrp_resp	144
Table 2-27.	intrp_resp Resp_code use by TLX command	145
Table 2-28.	The Resp_code specification for wake_host_resp	148
Table 2-29.	The Resp_code specification for mem_rd_fail	152
Table 2-30.	mem_rd_fail Resp_code use by TL command	153
Table 2-31.	The Resp_code specification for mem_wr_fail	155
Table 2-32.	mem_wr_fail Resp_code use by TL command	156
Table 2-33.	The Resp_code specification for mem_cntl_done	158

Approved

Table 2-34.	The Resp_code specification for kill_xlate_done	159
Table 2-35.	The Resp_code specification for cache_disabled	160
Table 2-36.	The Resp_code specification for cache_enabled	161
Table 2-37.	The Resp_code specification for atc_disabled	162
Table 2-38.	The Resp_code specification for atc_enabled	162
Table 3-1.	VC maximum credit count specification	165
Table 3-2.	DCP maximum credit count specification	166
Table 3-3.	Summary VC and DCP assignments	166
Table 3-4.	Example sequence of 2 block writes and 2 flag writes	172
Table 5-1.	DL frame format showing CRC and “bad data flit” coverage	182
Table 5-2.	DL frame loading to illustrate data ordering	186
Table 6-1.	Template capability definitions	190
Table 6-2.	Terms used in template capability specifications	190
Table 6-3.	TLX receive/TL transmit template	191
Table 6-4.	TL receive/TLX transmit template	194
Table 7-1.	Error event specification	199
Table 7-2.	Cache state transition errors	208
Table 8-1.	Feature compliance requirement notation	210
Table 8-2.	Profile specifications for TL commands	211
Table 8-3.	Profile specifications for TLX commands	211
Table 8-4.	Profile specifications for TL responses	213
Table 8-5.	Profile specifications for TLX responses	214
Table 8-6.	Profile specifications for TLX receive/TL transmit templates	215
Table 8-7.	Profile specifications for TL receive/TLX transmit templates	215
Table 8-8.	Profile specifications host operation modes	216
Table 8-9.	Profile specifications supported page size	217
Table 8-10.	Profile specifications supported dLength by TLX	217
Table 8-11.	Profile specifications supported dLength by TL	217
Table 8-12.	Profile specifications support of endianness data format by the TL	217
Table 8-13.	Profile specifications support of endianness data format by the TLX	218

Revision log

Each release of this document supersedes all previously released versions. The revision log lists all significant changes made to the document since its initial release. In the rest of the document, change bars in the margin indicate that the adjacent text was modified from the previous release of this document.

Revision date	Description
16 June 2020	Release of Approved OpenCAPI TL 4.0 specification.

About this document

This document provides the architectural specification of the OpenCAPI™ transaction layer (TL and TLX).

Architecture compliance terminology

In architecture descriptions, certain terms carry meaning in addition to their normal use in English. The following terms are used within this architecture specification to describe the requirements an implementation must meet to be considered compliant.

Table 1. Architecture terms

Term	Description
invalid	Used for multi-bit fields where the contents are not reliable. The field or bus shall not be examined for any functional or error checking actions.
may	An architectural option indicating that an implementation is allowed to have this behavior or characteristic.
reserved	With respect to a field of a register or bus: <ul style="list-style-type: none"> A reserved field shall be set to 0 by an implementation. A reserved field shall not be examined by an implementation. With respect to a code point: <ul style="list-style-type: none"> A reserved code point shall not be issued by a compliant implementation A reserved code point shall cause a bounded undefined response (that is, it won't hang the system). A reserved code point may be used in future revisions of the architecture. The architecture may specify that the use of a reserved code point is an error condition.
shall	An architectural requirement indicating a required behavior or characteristic.
uncertain	Used for single-bit fields where the contents are not reliable. The field or bus shall not be examined for any functional or error checking actions.
undefined	When the value of a field or a bus is undefined, the value may vary between implementations and may vary for a particular implementation for different actions. An implementation shall not examine a field when its value is undefined for functional purposes. However, the field may be checked for errors in those cases where an implementation includes error checking (that is, parity, ECC and so on)

Conventions used in this specification

Bit and byte numbering

Throughout this document, little-endian notation is used, which means that bits and bytes are numbered in descending order from left to right.

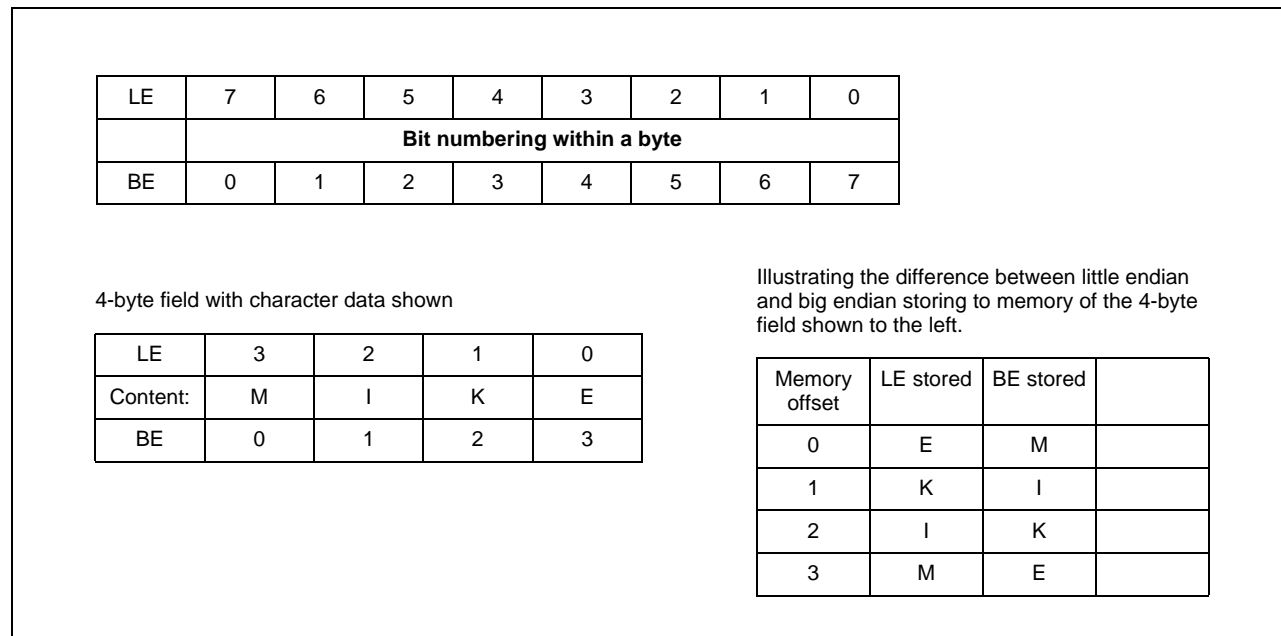
Thus, in the description of a 4-byte field, bit 31 is the most significant bit (MSb) and bit 0 is the least significant bit (LSb). The corresponding byte numbering is also shown.

MSb	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	LSb
	byte 3								byte 2								byte 1								byte 0								

Approved

The big-endian and little-endian byte ordering are described in the *POWER ISA, version 3.0, Book I. Figure 1* compares big-endian and little-endian notation.

Figure 1. Big- and little-endian comparisons



Representation of numbers

The notation for bit encoding is as follows:

- Hexadecimal values are preceded by an x and enclosed in single quotation marks. For example x'0A00'. Bit numbering is little endian and, in this example, is 15 to 0.
- Binary values in sentences are shown in single quotation marks. For example '1010'. Bit numbering is little endian and, in this example, is 3 to 0.
- ⁿx means the replication of x, n times. That is, x is concatenated to itself n-1 times. ⁿ0 and ⁿ1 are special cases:
 - ⁿ0 means a field of n bits with each bit equal to 0. For example, ⁵0 is equivalent to '00000'.
 - ⁿ1 means a field of n bits with each bit equal to 1. For example, ⁵1 is equivalent to '11111'.

RTL notation

RTL notations are used to specify the architectural transformation performed by the execution of a command.

Notation	Meaning
←	Assignment.
	Concatenation.
=, ≠	Equal, not equal relations.
≥, ≤	Greater than or equal to, less than or equal to relations.

Approved

Notation	Meaning
$>, <$	Greater than or less than relations.
$+$	Two's complement addition.
$-$	Two's complement subtraction, unary minus
\vee	Bitwise logical OR
\wedge	Bitwise logical AND
\oplus	Bitwise logical exclusive OR
$\text{Max}(x,y)$	Returns x when $x \geq y$; otherwise returns y
$\text{Min}(x,y)$	Returns x when $x \leq y$; otherwise returns y .
$\{x...y\}$	All integer values from x through y .
$A = \{x...y\}$	Returns true when A is a member of the set of integer values in the range of x through y .

Notes

This document contains engineering and developer notes.

Engineering notes

Engineering notes provide additional implementation details and recommendations not found elsewhere. The notes might include architectural compliance requirements. That is, the text might include *Architecture compliance terminology*. These notes should be read by all implementation and verification teams to ensure architectural compliance.

Engineering note

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin cursus hendrerit enim, vel tempus nibh ornare ut. Quisque ac augue eu augue convallis hendrerit. Mauris iaculis viverra ipsum nec dapibus. Nunc at porta libero. Curabitur luctus ultrices augue non pulvinar. Vestibulum mattis non ipsum at venenatis. Suspendisse euismod, neque et suscipit luctus, odio metus semper lectus, quis volutpat est libero quis nunc. Vivamus rutrum mauris sed tristique malesuada. Vivamus at augue vitae nisl cursus feugiat. Pellentesque efficitur sed nisi in dapibus. Curabitur vestibulum cursus arcu, ut mattis nisl.

Developer notes

Developer notes are used to document the reasoning and discussions that led to the current version of the architecture. These notes might also include recommended changes for future versions of the architecture, or warnings of approaches that have failed in the past. These notes should be read by verification teams and contributors to the architecture.

Developer note

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin cursus hendrerit enim, vel tempus nibh ornare ut. Quisque ac augue eu augue convallis hendrerit. Mauris iaculis viverra ipsum nec dapibus. Nunc at porta libero. Curabitur luctus ultrices augue non pulvinar. Vestibulum mattis non ipsum at venenatis. Suspendisse euismod, neque et suscipit luctus, odio metus semper lectus, quis volutpat est libero quis nunc. Vivamus rutrum mauris sed tristique malesuada. Vivamus at augue vitae nisl cursus feugiat. Pellentesque efficitur sed nisi in dapibus. Curabitur vestibulum cursus arcu, ut mattis nisl.

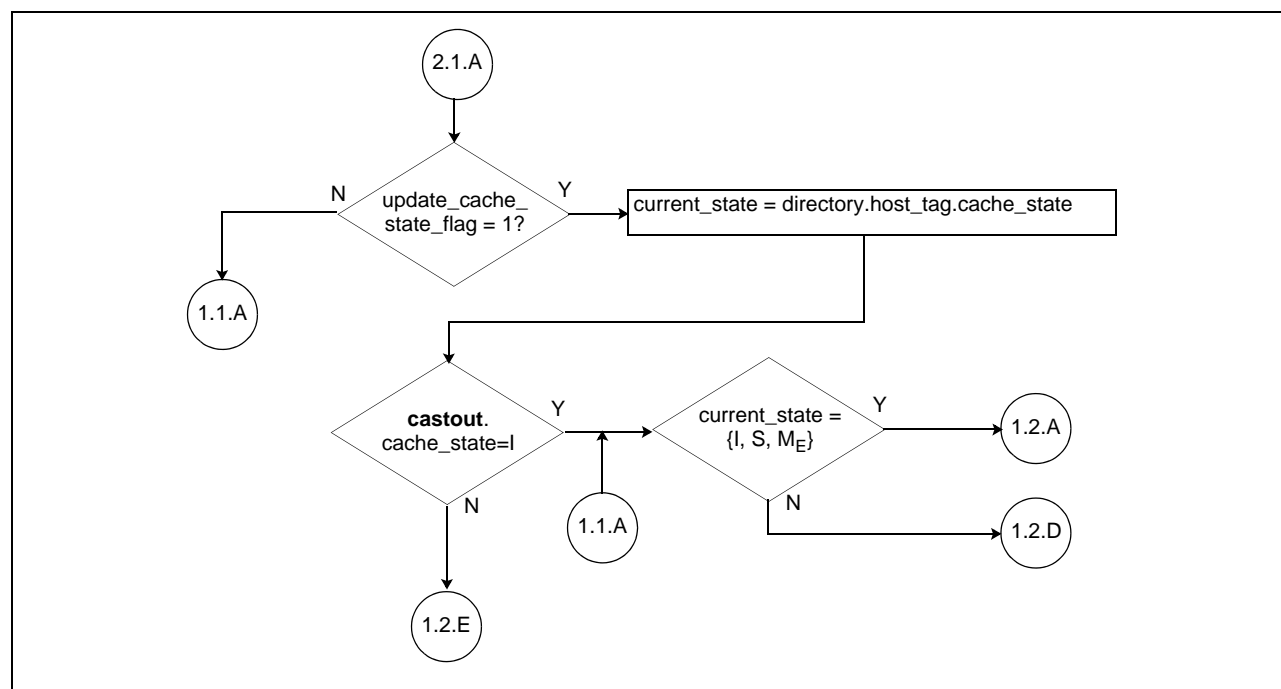
Command flows and transaction diagrams

Command flow diagrams

Command-flow diagrams show interactions within and across the different levels of the OpenCAPI protocol stack. Command flows use diamonds for decision blocks and rectangles for actions taken. Circles are used for on-page and off-page connectors and indicate a from-to direction based on the text content of the circle.

In *Figure 2*, a simple decision block with a state change and an off-page connector is shown. The text within the off-page connector has the format of “source page”.destination page”.”instance”. The off-page connectors shown in the figure is on page 1 of the figure¹ and is connecting to page 2 of the figure. On page 2, identical off-page connectors can be found. The instance indication allows for multiple connections to be shown between two pages. Connector 2.1.A illustrates a connection from page 2 to page 1 of *Figure 2*. An off-page connector can also be used to “connect” two spots on the same page as illustrated by connector 1.1.A. The direction of the arrow, into or out of a connector, decision block, or assignment-action block, indicates the direction of the sequence within the flow diagram.

Figure 2. Command flow example



Transaction diagrams

Transaction diagrams show the interaction between the TL and TLX layers and provide some illustrative notes for actions taken at the host protocol layer and the attached functional unit (AFU) protocol layer. In *Figure 3* on page 16, the diagram is broken into three vertical sections. From left to right, these are the AFU protocol layer notes, transactions between the TL and TLX layers, which are typically command and response packets, and the host protocol layer notes. Arrows indicate the direction in which the packet or action flows; for example, towards or away from the host (TL) layer.

1. All multi-page figures contain a “page n of y” notation in the figure description.

Approved

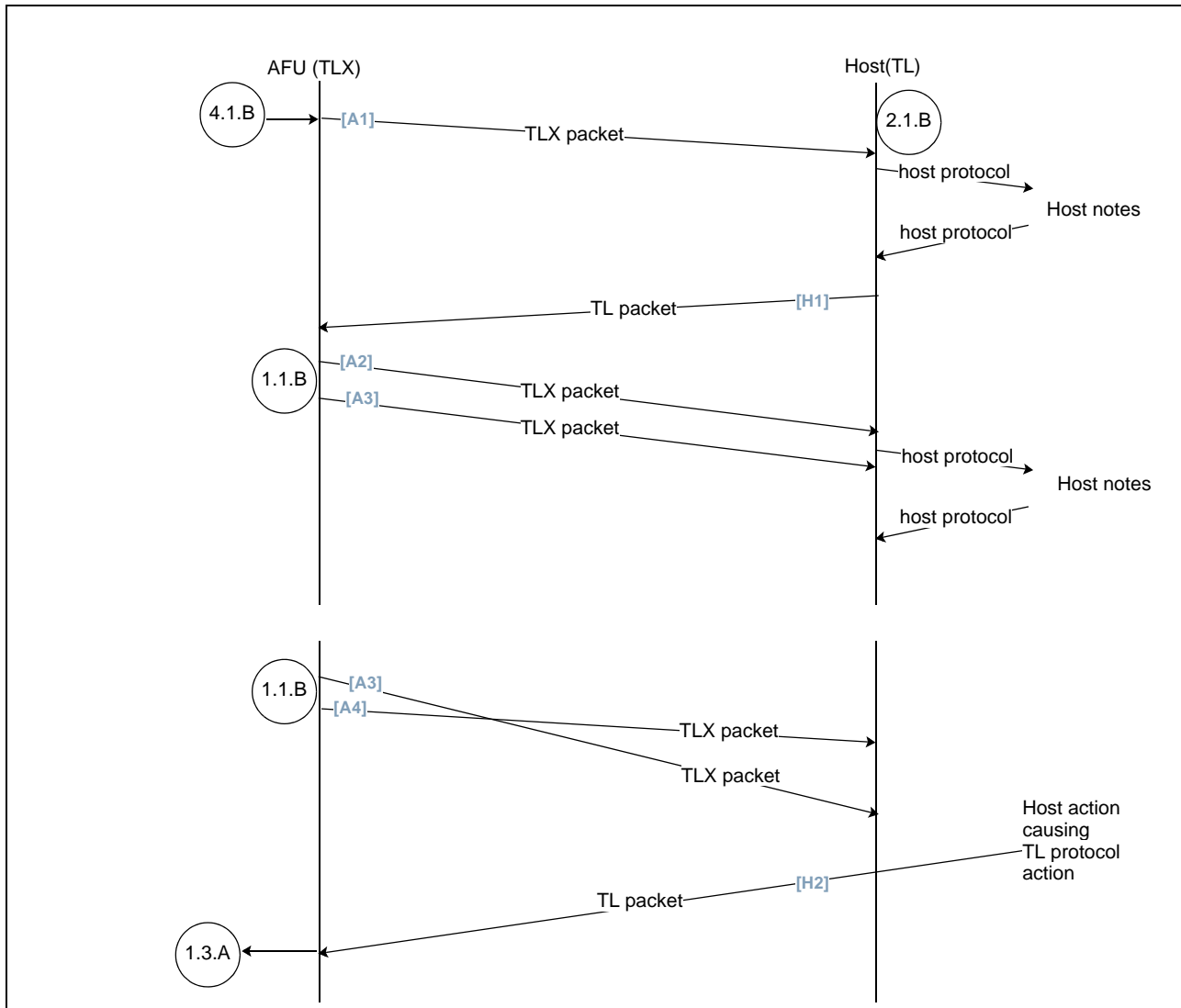
Circles are used for on-page and off-page connectors and indicate a from-to direction based on the text content of the circle. The text content is specified in the same manner for transaction diagrams as previously described for command flows. In addition to the specification of how connectors are used in command flows, in transaction diagrams, when a connector is used without an arrow, the transaction shown is one of multiple possible transaction outcomes. The use of this technique reduces the size of the transaction figure because the preceding set of transactions do not have to be repeated.

In *Figure 3*, connector 1.1.B illustrates an on-page connection without an arrow to indicate a different transaction outcome. The prior events are assumed to have occurred when looking at the second instance of the 1.1.B connector. In the second case, one TLX packet has passed a previously issued TLX packet; this is something that can occur when two packets use different virtual channels. Connector 1.3.A shows an off-page connection to page 3, and connector 4.1.B shows an off-page connection from page 4.

Arrow numbering is included in transaction diagrams to simplify references to transactions. The form of the arrow references indicates the source of the transaction (AFU or Host) and the instance of the arrow. As seen in *Figure 3*, [A1] is the first arrow from the TLX packet transaction and [H1] is the first TL transaction.

A break in the vertical lines indicates where a new transaction illustration starts or ends.

Figure 3. Example TLX and TL transaction diagram



Approved

Terms

The following terms are used in this document.

{EA, address context}	<p>A short hand notation to indicate an EA and <i>address context</i> pair.</p> <ul style="list-style-type: none"> Used when specifying an entry in an AFU L1 directory Use in discussions about address translation from EA to either an RA or PA.
acLookup(acTag)	<p>This is a function call used in command flows and transaction diagrams. It converts an acTag found in a TLX command packet into the address context (ac) used by the host's platform architecture to authenticate and provide the function requested by the TLX command.</p> <p>The result of an acLookup provides the error state of the address context provided. The state is shown as addressContext.state in the flows. The states are:</p> <ol style="list-style-type: none"> Good. The address context provided is valid. Invalid acTag. The acTag entry in the acTag table is not valid, or the acTag is specified outside the acTag table range. See <i>Table 7-1</i> on page 199. Invalid address context. The BDF and PASID associated with the acTag are invalid. The address context returned by the look up is not valid. See <i>Table 7-1</i> on page 199. <p>The function description is host specific.</p>
ACK	Acknowledgment.
address context	<p>(ac or addressContext). Address context is the information associated with a particular BDF and PASID pair. The association is formed by actions specified by the host's platform architecture.</p> <p>For TLX commands, the acTag and the acTag table provide the BDF and PASID. See <i>Section 4 The acTag table</i> on page 180 for additional details.</p>
address context space	A PASID paired with a BDF uniquely identifies the address space associated with a request. In OpenCAPI, a request is a TLX command.
address tenure	In a split transaction bus protocol, the commands and addresses are sent on the bus by the master before any data that might be associated with the transaction is moved. After the address tenure is completed, the status of the completion is examined. The data, if any is specified, is sent conditionally based on the status.
AFU	Attached functional unit. Architecturally, AFU refers to an end point unit or resource. Communication from the processor to the AFU goes through a protocol stack, transaction layer (TL), data link layer (DL), and physical medium layer (PHY). Command and data packets at the AFU interface are specified by the AFU command/data interface, which is the interface between the AFU protocol stack and the AFU.
AFU protocol	<p>AFU protocol layer. This layer currently consists of:</p> <ul style="list-style-type: none"> AFU_C protocol layer AFU_M protocol
AFU_C	<p>A processing element that is able to generate and receive commands to obtain data either in a cached state (using an attached L1), or in a checked-out (non-cached) state.</p> <p>It uses the AFU command/data interface to communicate with the AFU_C protocol stack. All addressing to the AFU_C protocol uses an EA only. It uses the AFU_C protocol stack to send and receive commands through the TLX. If the device contains an EA L1 cache, commands may result in cache line installation (cacheable operations).</p> <p>Non caching operations also specified for this device (non-allocating cache operations).</p> <p>See <i>AFU type</i> on page 28 for the different sub-types of an AFU_C.</p>
AFU_C protocol	AFU _C protocol layer. This protocol specifies the sequences on the AFU command/data interface and the OpenCAPI packet interface (TLX boundary) for an AFU _C -defined AFU.
AFU_M	<p>A processing element that receives commands to either provide or receive data. This element is a memory storage device and may be mapped to the system's memory address range.</p> <p>The attributes of the memory held by an AFU_M are managed by the operating system.</p> <p>It uses the AFU interface to communicate with the AFU_M protocol stack. All addressing to the AFU_M uses a PA only.</p> <p>See <i>AFU type</i> on page 28 for the different subtypes of an AFU_M.</p>

Approved

AFU_M protocol	AFU _M protocol layer. This protocol specifies the sequences on the AFU command/data interface and the OpenCAPI packet interface (TLX boundary) for an AFU _M -defined AFU.
alias	When address translation from one address type to another results in a many to one mapping, the set of addresses that map into the single address are referred to as alias of each other. During address translation, an alias is formed when two different addresses translate into the same address. For example: <ul style="list-style-type: none"> Two or more physical addresses (PA) of an OpenCAPI device map to the same host real address (RA). Two or more host RA map to a single attached OpenCAPI device's physical address.
Allocate command	Class of commands on the AFU command interface that indicates the allocation of a cache line. That is, a read operation that results in a cache line installed in the AFU L1.
AMO	Atomic memory operation. This operation performs an atomic update to a naturally aligned memory location. In some cases, this type of operation returns the original value of the memory location.
AP	Attached processor. Synonymous with AFU.
ATC	Address translation cache. The architecture describes a model for both a host ATC and an AFU ATC. See <i>Section 1.8 Address translation</i> on page 45.
BAR	Base Address Register.
back-off event	An event that causes a retry of an operation at some future time. The architecture specifies two types of back off events: short and long. The back off duration is controlled by configuration space registers specified in the OpenCAPI platform architecture.
BE	Byte enable.
cache block segment	A 64-byte block of memory held in a cache. A single cache state is associated with the cache block segment.
cache line	A cache line consists of one, two, or four cache block segments. A cache line is the size of the data block requested by a processor when moving data from a memory to a cache and is typically fixed in size. In many machine implementations, a cache line is associated with a single cache state. In this architecture, the host and AFU cache may not have the same cache line size. Due to this, there is no assurance that the cache state obtained by the AFU is consistent across the AFU cache line data block. The minimum granularity of cache state is assured at the 64-byte cache block segment. The TL architecture permits an AFU to request cacheable copies of one or more cache lines with a single command; for example, using read_me . Atomicity of all cacheable requests from the AFU are at the 64 byte cache block segment only.
CAPI	Coherent Accelerator Processor Interface.
CAPP	Coherent accelerator processor proxy.
command packet	TL/TLX construct. Contains command information for TL-to-TLX and TLX-to-TL communication.
convert2PA(RA)	This is a function call used in command flows and transaction diagrams. This converts an RA seen on the host processor bus into a PA used by the attached OpenCAPI device. The mapping of an RA to a device PA is device and host platform dependent.
CRC	Cyclic redundancy check.
critical OW request	The following commands are provisioned to support a critical octword (OW) request: <ul style="list-style-type: none"> rd_wntc, rd_wntc.t, and all dot variants of these commands read_mes, read_mes.t read_s, read_s.t read_me, read_me.t rd_mem A critical OW request is made when the address specified by the command is on a 32-byte (octword) boundary. Based on the size of the data block requested by the dLength specified, the address may not be naturally aligned. The requester is indicating that the OW specified by the command's address is latency critical. The requester is asking that the first data transfer associated with the first response for this command contain the critical OW. Responding with the critical OW first is optional.

Approved

data carrier	Data is transported between the TL and TLX in data carriers, which are defined as 64-byte data flits, or as 32- or 8-byte data fields found in control flits.
DCP	<p>Data credit pool. Each command or response specified with immediate data consumes one or more data credits.</p> <p>To add a command or response to a DL frame's control flit, both the VC credit and the DCP credit must be atomically obtained. That is, you must have both to proceed to insert the command or response into the DL frame.</p> <p>Adding a command or response specified with immediate data to a DL control flit defines the order the data is sent towards its destination.</p> <p>See <i>Section 5.1.3 Data transport, order, and alignment</i> on page 184 for full details.</p>
dError	Data error.
Device	The device refers to hardware and software attached via an OpenCAPI interface comprised of the PHYX, DLX, TLX Framer/Parser, TLX, AFU protocol stack, AFU protocol layer AFU interface and the AFU itself. See <i>Figure 1-1 OpenCAPI stack</i> on page 27.
DL	OpenCAPI data link layer found on the host processor.
dLength	Data length (dL).
DLX	OpenCAPI data link layer found on the external OpenCAPI device.
DMA	Direct memory access. A technique for using a special-purpose controller to generate the source and destination addresses for a memory or I/O transfer.
dP, dPart	Data part (dP).
EA	Effective address. This is the address as seen by a program. Some host architectures refer to this as a virtual address (VA). Mapping from an EA to an RA requires address translation services.
EA cache	A cache that is indexed by EA and <i>address context</i> pair. Access to the data in an EA cache requires that the address translation for the {EA, <i>address context</i> } is valid. See <i>Section 1.3.4 AFU Cache state transition reporting, initiation, and characteristics</i> on page 33 for details.
ECC	Error correction code. A code appended to a data block that can detect and correct bit errors within the block.
Flit	<p>An acronym for FLOW control digITs. Typically used in networking to specify the smaller pieces that a larger network layer packet is broken into. See FLITs.</p> <p>In this architecture specification, a flit is associated with the specification of a DL frame and is defined as a 64-byte unit of data. Control and data flits are specified.</p>
flit-cycle	The amount of time it takes 64-bytes to be either sent or received at the DL/TL or DLX/TLX interface.
host	The host refers to the host processor attached via an OpenCAPI link. It is comprised of the OpenCAPI PHY, DL, TL Framer/Parser, TL, the Host bus protocol stack interface and the hosts processors and other components that are implementation dependent on the host connected. See <i>Figure 1-1 OpenCAPI stack</i> on page 27.
host bus protocol layer	<p>Specifies the sequences on both the host bus and at the host bus protocol layer and the OpenCAPI packet interface to:</p> <ul style="list-style-type: none"> Respond to snooped host bus commands from the OpenCAPI packet to the OpenCAPI transaction layer to initiate action at the target AFU. Master commands on the host bus, per the specification found in the OpenCAPI packet, from the OpenCAPI transaction layer. Respond back to the source AFU at the conclusion of the host bus operation via an OpenCAPI packet to the TL layer.

Approved

host_tag arithmetic	<p>One <i>host_tag</i> is associated with each 64-byte data segment of a line held in an AFU_{C2} cache. That is, a <i>host_tag</i> is assigned to each cache block segment a line is composed of. The specification of a command or response shall ensure that the <i>host_tags</i> are adjacent when more than one 64-byte block of data is specified. Adjacent in this context means numerically adjacent. That is, add 1 to the value of a <i>host_tag</i> to get to the next <i>host_tag</i>. An operation that specifies 128 or 256 bytes shall be broken into multiple commands or responses when the <i>host_tags</i> associated with the data block are not adjacent.</p> <ul style="list-style-type: none"> When <i>dLength</i> is greater than one in a command that specifies a <i>host_tag</i>, the next <i>host_tag</i> is determined by incrementing the <i>host_tag</i> value by one. The command requester shall ensure that <i>host_tags</i> are sequential. When the host is returning cacheable data to an AFU_{C2} cache and the <i>dLength</i> is greater than one, the <i>host_tag</i> value in the response packet is associated with the 64-byte address determined by the command's address and the <i>dPart</i> specified in the response packet. The <i>host_tag</i> for the remaining 64-byte data blocks specified by this response is determined by incrementing the <i>host_tag</i> value by one for each 64-byte block. The host shall ensure that the <i>host_tags</i> specified by this response are sequential.
host_tag database	This is an architectural model construct managed by the AFU _{C2} . The <i>host_tag</i> database contains <i>host_tag</i> entries. See <i>Section 1.3.2.1 host_tag database</i> on page 30.
host_tag entry	An entry in the <i>host_tag database</i> . See <i>host_tag database</i> on page 30.
immediate data	Data associated with a command or response. Immediate data is the data specified for a write operation (the command and the data travel in the same direction). A read response has immediate data (the response and the data travel in the same direction). A read command does not have immediate data; the data arrives with the response.
inbound	The direction from the attached OpenCAPI device towards the attached processor chip.
LRU	Least recently used. A policy for a caching algorithm that removes from the cache the item that has the longest elapsed time since its last access. An algorithm used to identify and make available the cache space that contains the data that was least recently used.
MEM	The memory-mapped owner of the line. The owner could be the memory controller or an the owner of a memory-mapped I/O space. Some coherency protocols refer to this as a point of coherency (POC).
metadata	Refers to information associated with a <i>naturally aligned data block</i> . This architecture specifies a 7-bit metadata field and a 72 bit extended-metadata field. Metadata is found in control flits where the template specifies the association of the metadata with the data. 7-bit metadata fields are found in templates x'04' through x'09'. Extended metadata is found in templates x'0A' and x'0B'.
minimum signed integer value	<p>4-byte value: x'8000_0000'</p> <p>8-byte value: x'8000_0000_0000_0000'</p>
MMIO	Memory-mapped input/output. Refers to the mapping of the address space required by an I/O device for Load or Store operations into the system's address space.

Approved

mnemonic specification	<p>The general format of a mnemonic for either commands or responses is based on a base command/response type and “dotted” subtypes.</p> <p>The following subtypes are currently specified:</p> <p>When a mnemonic is only a base format (no additional dots), see the command specification to determine if a command is posted or non-posted.</p> <p>.be Byte enable field specified (dot-be).</p> <p>.d Data transfer (dot-d). Used only for intrp_req commands.</p> <p>.n Used for commands that require address translation (dot-n). If the address translation results in a miss in the ATC, the results of the address translation are used for the current operation, but are not loaded into the ATC.</p> <p>An implementation may:</p> <ul style="list-style-type: none"> Ignore this directive. Store the results in a TLB. <div data-bbox="521 646 1438 760" style="border: 1px solid black; padding: 5px;"> <p>Engineering Note</p> <p>The dot-n form is expected to be used with a host implementation that has a multi-level ATC. This form of the command allows <i>warming up</i> the higher levels of the ATC hierarchy without installing into the more resource-precious level 1 ATC.</p> </div> <p>.ow Octword data specified (dot-ow). Used for responses with immediate data consists of one or more control flits containing a 32-byte data field. The TL and TLX templates that support these control flit forms are specified in <i>Section 6 TL and TLX template specifications</i> on page 189. Responses with this form contain a dPart field with 32-byte address granularity.</p> <p>.p posted (dot-p). The absence of this sub-type designation for commands indicates that the command might be non-posted (that is, a response is expected).</p> <p>.s Indicates a presync is required prior to execution of the command. (dot-s)</p> <p>.t Indicates the address specified by the command consists of a translated address (TA) instead of an EA. Applies to TLX commands only. (dot-t)</p> <div data-bbox="537 1045 1446 1247" style="border: 1px solid black; padding: 5px;"> <p>Developer note</p> <p>To eliminate errors in the expected use cases for dot-s specified commands, the following restrictions were applied when forming the TLX command set:</p> <ul style="list-style-type: none"> All dot-s commands must also be dot-t. This ensures that address translation does not cause the operation to fail. Any other failure is expected to be fatal that causes the link to go down. All dot-p write commands must also be dot-t. This is done to simplify the architecture. </div> <p>.xw X-word data specified (dot-xw). Used for responses with immediate data consists of one control flit containing an 8-byte data field. The TL and TLX templates that support these control flit forms are specified in <i>Section 6 TL and TLX template specifications</i> on page 189.</p> <div data-bbox="542 1381 1451 1509" style="border: 1px solid black; padding: 5px;"> <p>Developer note</p> <p>The current set of TL/TLX templates limits the specification of dot-xw responses to 8-byte transfers. A future version could provide additional templates that support 16-byte data fields. The response encodes for the current set of dot-xw responses has bit 24 specified as ‘0’. To specify 16 bytes, bit 24 would be set to ‘1’.</p> </div>
MRU	Most recently used. One of the results of an <i>LRU</i> algorithm. The cache entry that has the shortest amount of elapsed time since its last access.
NACK	Negative acknowledgment.

Approved

naturally aligned data block	<p>A data block containing L bytes is naturally aligned when the address specifying the location of the data block is an integer multiple of the length of the block.</p> <p>Where:</p> $i = \{0, 1, 2, 3, \dots\}$ $L = \{64, 128, 256\}$ <p>A naturally aligned data block is located from byte address $i * L$ through $(i + 1) * L - 1$.</p> <p>A command's address specification may not be aligned as specified above. An unaligned address points to a naturally aligned data block of length L, with a starting address of $\text{adr}(63:(\log_2 L)) \parallel (\log_2 L)0$.</p>
nMMU	<p>An abstraction of a host implementation-dependent construct that performs page-table walks based on the underlying page-table architecture as specified by the host architecture and host's platform architecture.</p> <p>In the command flows and transaction diagrams, it returns:</p> <ul style="list-style-type: none"> nMMU_response.status = 0 when an address translation is successful. It returns page_size and access permissions. nMMU_response.status <> 0 when software must be invoked to complete the requested address translation.
Non-allocate command	<p>Class of commands on the AFU command interface that indicates the operation will not result in the allocation of a cache line. That is, a non-allocate read operation shall not result in the installation of a cache line in the AFU L1. However if a cache line has already been allocated, the line is already resident in the AFU L1, then the operation results with the line remaining in the cache. See the command description for details.</p>
null control flit	<p>A null control flit is defined as using template x'00'. The 6-slot packet contains a 1-slot null command, and the remaining five slots are undefined. A return credit response found in slots 0 and 1 may be used to return credits. See <i>Section 6 TL and TLX template specifications</i> on page 189 for the specification of template x'00'.</p>
OMI	<p>OpenCAPI memory interface. Additions to the OpenCAPI 3.0 TL specification:</p> <ul style="list-style-type: none"> Provide dot-ow and dot-xw formats of commands and responses Provide TL and TLX template specifications x'04' through x'08' Expand data carrier types to include 32- and 8-byte data fields found in some control flit formats Specify metadata and extended metadata Add critical OW specification for some read commands Add MAD fields to some read commands
outbound	<p>The direction from the processor chip to the attached OpenCAPI device.</p>
PA	<p>Physical address. This refers to the address space owned by an AFU_M device. The host converts the RA to the AFU_M device's physical address space using configuration settings in the host that are determined during initialization of the attached OpenCAPI device.</p> <p><i>A PA is not the result of address translation of an EA as might be the case in some host architectures. The host maps the device's PA into its own (RA) address space.</i></p>
packet	<p>TL/TLX unit of information. A command packet contains commands. A response packet contains response information. See the specification of command and response packets in <i>Section 2 TL and TLX command and response specifications</i> on page 48.</p> <p>Data is transferred in address-aligned:</p> <ul style="list-style-type: none"> 64-byte data flits 8-byte data fields specified in some template specifications found in <i>Section 6</i> 32-byte data fields specified in some template specifications found in <i>Section 6</i>
PHY	<p>The PHY layer interfaces to the DL and the network.</p> <p>This is the bit stream level specifying the electrical and optical transmission medium as well as the network interconnect topology.</p> <p>The current specification for the network is a point-to-point connection.</p>
PHYX	<p>On the OpenCAPI device, the PHYX layer interfaces to the DLX and the network. This is the bit stream level that specifies the electrical and optical transmission medium as well as the network interconnect topology. The current specification for the network is a point-to-point connection.</p>
pL, pLength	<p>Partial length.</p>
POC	<p>Point of coherency. See definition of MEM.</p>

Approved

presync	<p>dot-s formatted commands cause a presync event prior to execution.</p> <p>When a dot-s command reaches the head of the service queue and is eligible to be removed from the head of the service queue as described in <i>Section 3.3 TL Virtual channel and service queues</i> on page 168, it is enqueued into the service queue's corresponding presync queue.</p> <p>As described in <i>Section 3.4 TL Presync queues</i> on page 171, the command is dispatched to the host only when it reaches the head of the presync queue and when all prior commands have completed.</p> <p>A command is defined as being completed by the host bus/AFU protocol layer when service by the protocol layer ends. The operation completes. When, for example, a write operation's data is globally visible. A read operation completes when it provides the data. An address translation operation completes when it provides the necessary address translation.</p>
presync queue	<p>A presync queue is a queue placed after the service queue. A dot-s command at the head of a service queue is dequeued into the presync queue specified for the service queue.</p> <p>Presync queues are based on a VC. They use the same hash rules specified for service queues. An implementation is not required to use the same hash for a presync queue as it does for its service queue. The hash may be less or more perfect. However, it is constrained by the definition of a VC and the perfect hash specified for the VC as defined in the definition of a <i>service queue</i>.</p>
RA	Real address. A real address is the result of address translation of an EA. Some host architectures refer to this as a physical address; this specification reserves the term physical address for other purposes. See the definition of <i>PA</i> .
Reserved/R	Indicates that a field or bit specification is reserved. A reserved field is set to zero and shall not be examined by an implementation. See <i>Architecture compliance terminology</i> on page 11.
response packet	TL construct that contains response information to commands. Used for TL-to-TLX and TLX-to-TL communication.
responder	TL or TLX that accepts a command, services the command, and sends back a response TL/TLX packet that provides data, when required, and status of the service to the command.
requester	TL or TLX that issues a command. The requester collects all responses returned by the responder, if any, to determine the status of the service provided by the command. When the command is posted, responses are not returned.
RTL	Register transfer language.
segment	When used in reference to data, a segment refers to a naturally aligned 64-byte portion of a data transfer. For example, a 256-byte data transfer contains four segments.
service queue	<p>The members of a service queue are an ordered set of commands. The commands are selected by applying a hash against the VC, BDF, PASID and stream_id associated with the command. The hash results in the selection of a specific service queue. The hash is both implementation and command dependent. The hash is command dependent because not all commands are specified with a BDF, PASID and stream_id. Commands that are not specified with a VC do not enter a service queue.</p> <p>Per VC, the following operands may be included in the hash:</p> <p>TLX.vc.0 This VC is used for most responses, and the hash is the VC.</p> <p>TLX.vc.2 Contains only castout and castout.push TLX commands. The hash is the VC.</p> <p>TLX.vc.3 Contains various read and write TLX commands as well as assign_actag and kill_xlate_done.</p> <p>The assign_actag command is serviced before entering into a service queue. All other commands are sorted using a hash based on the VC, BDF, PASID and stream_id.</p> <p>The kill_xlate_done response is removed from the VC and is serviced as described in <i>Section 3.4.1 TL queuing and service of kill_xlate_done</i> on page 173.</p> <p>When the hash specified is used for a VC, the hash is perfect and the resulting service queue is identical to the definition of a <i>virtual queue</i>.</p> <p>When an implementation removes hash terms from the VC-specific specification, the hash is not perfect. For example, if the stream_id term is removed from TLX.vc.1, all commands, regardless of stream_id, occupy the same service queue.</p> <p>There is at least one service queue per VC supported by the implementation.</p>
slot	A slot is a 28-bit granule used to specify a TL or TLX command or response packet.
SUE	Special uncorrectable error. Refers to error detection and attempted correction to a block of data. A SUE indicates that an error was detected upstream from the present error detection logic. The use of SUE indications aids in determining error origination as part of a first error incident reporting scheme.

Approved

synonym	<p>In this term relates to address translation during cacheable transactions. A synonym is formed when two or more effective addresses (EAs), regardless of address context, map into the same host real address (RA).</p> <p>A synonym is detected by the host during the execution of a TLX read_me, read_me.t, read_mes, read_mes.t, read_s, read_s.t, upgrade_state, or upgrade_state.t command. When the EA is translated or the TA used, a synonym is detected if the host's proxy directory is holding the RA in a state other than invalid and indicates that the line is inclusive of the AFU's cache.</p> <p>Detection of a synonym results in a TL synonym_detected response. The host_tag is returned to the requester, which is required to manage the synonym detection event.</p> <p>The AFU is not required to fully support synonyms. At a minimum, on receipt of a synonym_detected response, the AFU shall:</p> <ul style="list-style-type: none"> • Wait until all responses for the current operation have been received. • Schedule a castout or castout.push of the data block specified by the original read request. • Schedule a synonym_done command once the cast out has been added to its VC. <p>After the synonym_done has been placed into its VC, the AFU may re-issue the original request.</p>
TL	<p>OpenCAPI transaction layer found on the host processor.</p> <ul style="list-style-type: none"> • Interfaces to the DL and the protocol layer. Responsible for command-packet formation and response-packet handling and formation. Ensures that the order of data sent to the DL matches the command- and response-packet order sent to the DL. • Manages data flits, 8- and 32-byte data carriers specified in some control flits from the DL. Associates the data with the command or response packet that was received prior to the arrival of the data. The command- and response-packets contain data descriptors that enable this association. • Performs flow control. • Performs error handling and control. • Manages all <i>service queues</i> and <i>presync queues</i> associated with each virtual channel. Order is retained within virtual channels.
TLB	<p>Translation lookaside buffer. An on-chip cache that holds the translation of an effective address (EA) to a real address (RA). A TLB caches page-table entries for the most recently accessed pages, thereby eliminating the necessity to access the page table from memory during load-store operations.</p>
TLX	<p>OpenCAPI transaction layer found on the external OpenCAPI device.</p> <ul style="list-style-type: none"> • Interfaces to the DLX and the protocol layer. Responsible for command packet formation and response packet handling and formation. Ensures that the order of data sent to the DLX matches the command and response packet order sent to the DLX. • Manages data flits, 8-, and 32-byte data carried in some control flits from the DLX and associates the data with the command or response packet that was received prior to the arrival of the data. The command and response packets contain data descriptors that enable this association. • Flow control. • Error handling and control.
UE	<p>Uncorrectable error. Refers to error detection and attempted correction to a block of data. An uncorrectable error indicates that an error was detected and the attempted correction failed.</p>
VC	<p>Virtual channel. See <i>Section 3 Virtual channel and data credit pool specification</i> on page 163, and the specification of all commands and responses in this section.</p>
virtual queue	<p>The specification of a <i>service queue</i> describes the VC-specific hash required to form a service queue from a virtual queue.</p> <p>The members of a virtual queue are an ordered set of commands received from a VC. That is, the ordering of the commands found in the VC shall be retained when adding commands from the VC to a virtual queue.</p>
warming up	<p>The process of loading or populating a cache with a set of valid data.</p>
write class command	<p>A command that is used to write data to a destination. The source of a write class command is also the source of the data.</p>

Approved

<p>xlate_result = adr_xlate(EA, addressContext)</p>	<p>This is a function call used in command flows and transaction diagrams. This returns the host's results from an address translation. The function returns an RA (xlate_result.RA) and a status (xlate_result.status). The status returned is:</p> <ol style="list-style-type: none"> 1. Complete. Address translation completed successfully with an RA provided. The ATC may have been updated with the result. 2. rty_req. Indicates that the address translation could not be completed at this time. The operation may be attempted at a later time. 3. xlate_pending. Indicates that the address translation could not be completed. The ATC did not contain the translation and software was invoked. An asynchronous xlate_done TL command is sent when the software actions have completed. <p>Engineering note</p> <p>The Resp_code=xlate_pending is sent in a read_failed, touch_resp, or write_failed response packets. These TL responses shall precede the xlate_done command in the TL.vc.0 virtual channel.</p>
--	--

1. Overview

The OpenCAPI transaction layer specifies the control and response packets passed between a host and an OpenCAPI device. The transaction layer implemented on the host is referred to as the TL. The transaction layer implemented on the OpenCAPI device is referred to as the TLX.

On the host, the transaction layer converts:

- Host-specific protocol requests into transaction-layer-defined commands.
- TLX commands into host-specific protocol requests. When the host protocol completes, it provides responses to the TLX commands when required.
- TLX responses into responses for host-initiated requests.

On the OpenCAPI device, the transaction layer converts:

- AFU-specific protocol requests into transaction-layer-defined commands.
- TL commands into AFU-specific protocol requests. When the AFU protocol completes, it provides responses to the TL commands when required.
- TL responses into responses for AFU-initiated requests.

Working together, the TL and TLX provide a standard method to bridge between a host protocol architecture and an AFU protocol architecture. This is accomplished by the exchange of command and response packets specified by the OpenCAPI transaction layer specification.

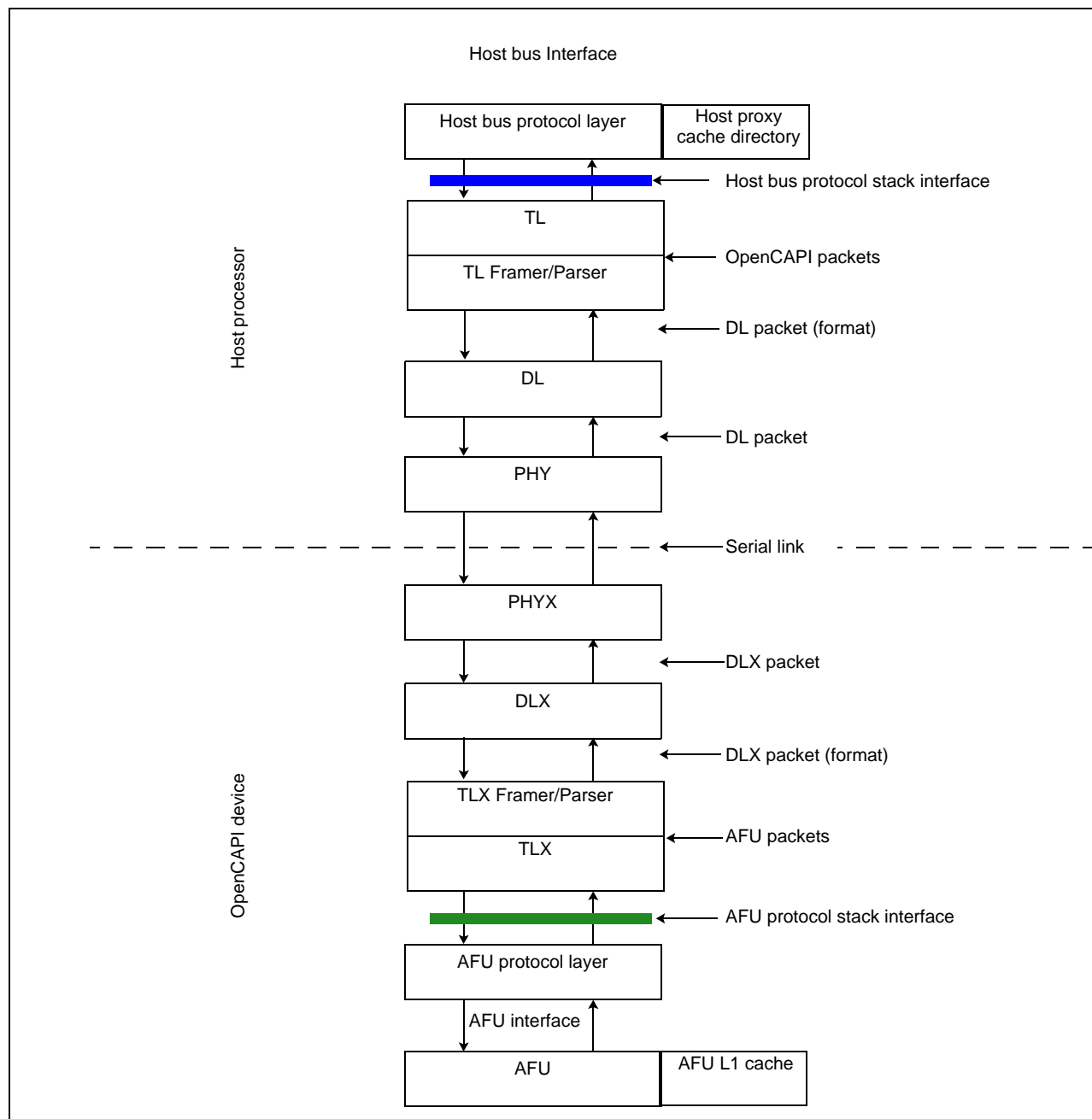
Version 4.0 of this specification builds on the version 3.1 base and adds the following extensions:

- New *AFU type*: AFU_{C2} . The AFU_{C2} adds an AFU processing element that supports a local cache.
- The AFU_{C2} introduces an EA AFU cache and specifies cache states. See *Table 1-1 Cache state descriptions* on page 32. Commands for the host to manage access between the AFU processing element and its cache are added. Commands for the host to manage the AFU cache contents are added.
- Adds a host proxy cache to the host protocol layer. The architecture specifies that the host proxy cache is inclusive of the AFU_{C2} cache contents. See *Section 1.3.4 AFU Cache state transition reporting, initiation, and characteristics* on page 33.
- The AFU_{C2} adds the concept of a *host_tag* that associates the AFU's cache to the host's proxy cache.
- TL.vc.2 and TLX.vc.2 virtual queues
- TLX.dcp.2 data credit pools
- New commands and responses to manage cacheable data.
- AFU ATC architecture models and commands to manage them.
- Adds *Translated addresses, AFU ATC, and dot-t commands*.
- dot-p forms of commands that are posted.
- dot-s forms of commands that enable the use of *TL Presync queues*.
- Modifications to *Write fragmentation ordering and atomicity* and the addition of the ordered segment (Os) directive to some write commands.
- TL versions of **amo_rd**, **amo_rw**, and **amo_w** commands
- Additional *Error event specifications* for the new extensions.

1.1 OpenCAPI protocol stack

Figure 1-1 on page 27 shows the OpenCAPI protocol layers.

Figure 1-1. OpenCAPI stack



1.2 Host operation modes

The interface between the host and the AFU can be implemented with varying levels of complexity. Interoperability with an AFU implementation is dependent on the operation mode supported by the host and the requirements of the AFU.

The various combinations of AFU capabilities are broken into two subclasses, AFU_C and AFU_M . There are three sub-classes of AFU_C , and two sub-classes of AFU_M .

AFU type	Description
AFU_{C0}	(C0 or none). There is no visible-to-the-host processing element. The host never sees any commands sourced by the TLX. While a processor element might not be visible to the host, it may still be present. If it is present, it shall not cache any lines in any coherent data valid state and shall not rely on the host's coherency protocol for correct operation.
AFU_{C1}	(C1 or type 1 processing element). A processing element with no cache. An AFU_{C1} may issue TLX commands to the host. It uses an EA to access host system memory. The host provides address translation and access to system memory.
AFU_{C2}	(C2 or type 2 processing element). A processing element with a EA L1 cache. An AFU_{C2} uses EA to access host system memory. Host provides address translation, access to system memory, and host_tag with cache state for each 64 byte naturally aligned data block (cache line or partial cache line depending on the host's cache line size). See Section 1.3 AFU_{C2} on page 30 for the requirements this mode places on the host and device behaviors.
AFU_{M0}	(M0 or none). There is no host system address space mapped to this device. That is, host system address space shall not be mapped to this device. Configuration space may be specified for this device.
AFU_{M1}	(M1 or type 1 MEM). A range of host system address space shall be assigned to this device. This address range shall be accessible only through the host (TL-to-TLX interactions). The host shall use the PA to access data. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;">Engineering Note The address range assigned to this type of device may be limited to MMIO space only or may include memory that is managed by the operating system; for example, memory that is backed by DRAM and that can be migrated to disk as needed.</div>

The following sections describe the combinations of AFU_C and AFU_M devices on a single OpenCAPI device.

1.2.1 No attached device (C0, M0)

No device is attached to the OpenCAPI interface. No transactions occur.

1.2.2 MEM-only mode (C0, M1)

In this mode of operation, the AFU appears to be a memory controller with an address space mapped into the host's system address space. Access by the host uses the PA. See the specification of an AFU_M .

Developer note

While a processor element might not be visible (C0), it might still be present. If it is present, it cannot cache any lines in any coherent data valid state and cannot rely on the host's coherency protocol for correct operation. Examples of this type of configuration that meet the above requirements are:

- An encrypted memory device. In this device, the data is encrypted/decrypted when the data is written to the M1 or read from the M1. The processing element that performs the encryption/decryption is not visible to the host. Topologically, the processing element is between the memory and the host.
- A memory cache device. The cache is managed by a processing element. The cache exists to reduce latency, and the cache states are not related to the host's coherency protocol. Data might be fetched or stored into the cache. The host cannot tell this is happening except for an improvement in performance.

1.2.3 Checkout mode (C1, M0)

In this mode of operation, the AFU appears as a processing element without a cache. It may have a non-coherent scratch pad memory, which is used for local processing only. Access to system memory is permitted as coherent, no-intent-to-cache actions. The AFU shall use an EA for these requests. The host shall perform address translation to enable access to system memory.

1.2.4 Checkout with MEM (C1, M1)

In this mode of operation, the AFU appears as a processing element without a cache. It may have a non-coherent scratch pad memory, which is used for local processing only. Access to system memory is permitted as coherent, no-intent-to-cache actions. The AFU shall use an EA for these requests. The host shall perform address translation to enable access to system memory.

In addition, the AFU provides a memory controller function with an address space mapped into the host's system address space. Access by the host uses a PA. See the specification of an AFU_M .

1.2.5 Cache-only mode (C2, M0)

In this mode of operation, the AFU appears to be a processor element with a coherent EA based L1 cache. Access to system memory is permitted using EA. The host performs address translation to enable access to system memory.

This combination is not likely to be viable since there must be some access method to configure the OpenCAPI device.

1.2.6 Cache + MEM mode (C2, M1)

In this mode of operation the AFU appears to be a processor element with a coherent EA based L1 cache and a separate PA accessed memory controller. The processor element is unable to directly access the contents of the memory controller; that is, any access by the processor element requires address translation through the host and access to the data is provided by the host.

1.3 AFU_{C2}

As described in *Host operation modes*, support for an AFU_{C2} places requirements on the behavior of the device and on the host. The following sections describe the requirements placed on the host's and device's behavior.

1.3.1 Host proxy cache

Supporting an AFU_{C2} places a requirement on the host to provide a host proxy cache. The host proxy cache shall appear to be inclusive of the AFU L1 EA cache.

The host proxy cache appears to be an RA cache holding the coherence state of the RA as known by the host. The states held by a host proxy cache entry appear to be the same as those specified for the AFU cache which are specified in *Table 1-1 Cache state descriptions* on page 32. An implementation may choose to provide additional or fewer states. Any implementation specific changes to the states specified by *Table 1-1* shall be done in such a manner that the differences are not externally visible.

A host proxy cache entry shall appear to have a 64-byte data block granularity. That is, each entry in the host proxy cache contains the state of a 64-byte naturally aligned block of data. A host tag is assigned to each entry in the host proxy cache. An implementation may choose to form a host proxy cache differently. Any implementation specific differences to the host proxy cache shall be done in such a manner that the differences are not externally visible.

The assignment of the `host_tag` value to the host proxy cache entry is implementation dependent.

The state held by the host proxy cache entry may not match the state of the corresponding entry or entries in the AFU_{C2} EA L1 cache² pointed to by the `host_tag`.

1.3.2 AFU_{C2} model

The architectural model of a AFU_{C2} contains the following structures

- *host_tag database*
- L1 EA cache directory
- data cache

An AFU_{C2} may choose to support synonyms or may provide only the minimum support for synonyms.

1.3.2.1 *host_tag database*

The `host_tag` database is comprised of a number of `host_tag` entries. The number of entries is determined during discovery and configuration time. The host tag value is provided by the host and appears in commands and responses. The host tag value is used to index into the `host_tag` database.

The architecture model of a host tag entry is comprised of a valid field, a L1 cache id and a set and way id. The L1 cache id allows for multiple L1 cache directories to be maintained, and the set and way id specifies the entry in the L1 directory that is described below. When a new L1 cache directory entry is created, the host

2. See *Section 1.3.2.2 L1 EA cache directory* on page 31.

Approved

provides a host_tag. The AFU_{C2} provides the L1 cache id and the set and way id. The host_tag entry contains a single host_tag dirty bit. This bit is set either during the validation of the host_tag entry or when the data associated with the host_tag entry has been modified by an action taken by the AFU_{C2}.

To support synonyms, the host_tag entry is expanded by providing multiple valid, L1 cache id, and set and way fields.

An implementation may choose to form a host tag database differently. Any implementation specific differences to the host_tag database cache shall be done in such a manner that the differences are not externally visible.

See *Section 1.5 Host tags* on page 39.

1.3.2.2 L1 EA cache directory

The architectural model of the L1 EA cache directory is comprised of entries that each have a 64-byte data block granularity. The cache line for the L1 EA may be 64-, 128-, or 256-bytes. Since the cache line of the host may also be 64-, 128- or 256-bytes, both the host proxy cache and the L1 cache directory are required to appear to maintain 64-byte granularity.

The L1 EA cache directory entry contains an address tag³, an address context identifier, the host_tag associated with the 64-byte naturally aligned data block associated with this entry, the cache state of the entry as defined in *Table 1-1 Cache state descriptions* on page 32, and a data cache set and way identifier.

The data cache set and way identifier is an explicit pointer to the location of the data. An alternative is an implied pointer based on the location of the cache directory entry.

An implementation may choose to form the L1 EA cache directory differently. Any implementation specific differences to the L1 EA cache directory shall be done in such a manner that the differences are not externally visible.

1.3.2.3 data cache

The data cache holds the data referenced by the host tag data base and the one or more L1 cache directory entries. The implementation shall ensure when synonyms are supported, that changes made to the data by processes that have write authority is seen by all synonyms. That is, any accesses using a synonym and holding a state that indicates valid data, has access to the same data state as any other synonym holding a state that indicates valid data⁴. This could be accomplished by a using either a single data cache, or distributed data caches based on either the L1 cache id or by the address context used.

An implementation may choose to form the data cache differently. Any implementation specific differences to the data cache shall be done in such a manner that the differences are not externally visible.

3. A set and way cache structure is assumed

4. I and E_i states indicate invalid data

Approved

1.3.3 AFU cache states

Cache state specifications and legal downgrade states are provided in *Table 1-1* on page 32.

Table 1-1. Cache state descriptions

Mnemonic	State name	Unique		Description
I	Invalid	No		Cache line data and L1 EA entry are not valid
E	Exclusive	Yes		Cache line data and L1 EA entry are valid. Data is not modified with respect to MEM/POC contents unless the dirty bit in the host_tag data base is set. Castout does not require a memory write operation unless the dirty bit in the host_tag database is set. Write permission is implicitly indicated for this state.
E _I	Exclusive - no data	Yes		Cache line data is not valid, and the L1 EA entry is valid. This is a special E state where the line has been forced back to the memory owning the line. This cache state occurs when exclusive ownership of the line is granted using upgrade_state requesting a I → E _I transition.
M	Modified	Yes		Cache line data and L1 EA entry are valid. Data may be modified with respect to MEM/POC contents. Castout requires a memory write operation. Write permission is implicitly indicated for this state.
S	Shared	No		Cache line data and L1 EA entry are valid. Write permission is unknown and shall not be assumed. Eviction actions from the L1 EA cache shall not include a memory write operation unless the dirty bit in the host_tag database is set. That is, a castout.push shall not be issued when the cache state transitions from an S to an I state and the dirty bit in the host_tag data base is not set.

In *Table 1-2* below the relationship between the host proxy state corresponding to a single host_tag entry and the one or more L1 EA entries that point to the same host_tag entry in the *host_tag database* is illustrated.

In *Table 1-2* below, there are separate columns for an implementation that supports synonyms and for implementations that do not. AFU_{C2} Implementations that do not support synonyms have a one to one relationship between a single host_tag entry and a single L1 EA cache directory entry. AFU_{C2} implementations that do support synonyms have a one to many relationship between a single host_tag entry and multiple L1 EA cache directory entries. AFU_{C2} implementations that support synonyms are required to track the clean/dirty state of the data associated with the host_tag entry. This is used by the AFU_{C2} to determine if a **castout.push** is required when invalidating the host_tag entry. That is, when the last L1 EA cache directory entry associated with a host_tag entry is evicted, the clean or dirty state of the data (dirty bit set column in *Table 1-2*) is examined to determine if a **castout** or **castout.push** is used to inform the host.

Table 1-2. Concurrent host proxy cache (L2) and L1 EA cache states (L1)

L2 States	No synonym support		Synonyms supported	
	L1 states	dirty bit set	L1 states (may be concurrent in the L1) ³	dirty bit set
I	I	No	I	No
S	S	No	S, I	No, No
E	E, E _I , M	No, No, Yes	E, E _I , M, S, I	No ² , No ² , Yes, No ² , No ²
E _I	E _I , M	No, Yes	E _I , M, S, I	No ² , Yes, No ² , No ²
M	M	Yes	E, M, S, I	Yes, Yes, No ¹ , No ¹

1. L2 state is M. This means the data was at one time handed to the AFU_{C2} in a dirty state. The dirty bit is set.
2. Dirty bit set if data has been modified by other synonyms that have write authority.
3. When the L2 is not in an I state, a state other than I is expected in the L1. The only time that the L1 state of a line is I and the state found in the L2 is not I, is during the window where the line has been cast out by the AFU_{C2}, that is, invalidating the host tag, and the time the host sees and acts on the cast out.

1.3.4 AFU Cache state transition reporting, initiation, and characteristics

As described above, host_tags are used to index into the host_tag database and are assigned by the host. The reader of this specification should be aware that this specification refers to one host_tag per coherence block for descriptive simplicity. When a coherence block uses multiple host tags, the intent of the architecture is to apply all rules and comments to the set of host tags associated with the coherence block, or in the case of command and response descriptions, with the aligned data block specified by the command or response. Section 1.5 Host tags on page 39 describes host tags in detail.

The AFU L1 cache is an EA cache. When the AFU processor accesses its L1 cache, the address translation for the cache entry must be valid. This requires that the host inform the AFU_{C2} when address translation is lost for an entry in the AFU L1 cache. This may be accomplished using any of the following techniques.

- The host may issue **force_evict** to clear the entries out of the L1 EA cache.
- When the AFU maintains an ATC⁵, the AFU shall access both its ATC and its L1 EA cache. Both the ATC and the L1 cache must be valid in order for the AFU processor to access the L1 EA contents. If either is invalid, the processor shall not gain access to the requested data.

Note that the AFU_{C2} L1 EA cache shall continue to respond to **force_evict** commands from the host when there is no valid address translation in the AFU's ATC.

Engineering note

In reference to the last bullet above. The requirement that the AFU's ATC contain a valid address translation in order for access to the L1 EA content allows the host to quickly invalidate address translations and stop the AFU_{C2} processor from accessing lines that have lost their address translation.

An AFU implementation may support *synonyms*. A synonym is detected by the host when the AFU requests a line for its cache for which the host has already provided a host_tag. In these synonym detected cases, the host_tag is returned without data to the AFU using the TL **synonym_detected** response. The AFU locks the host_tag provided with the **synonym_detected** response. Additional actions are taken by the AFU to

5. See Section 1.8 Address translation on page 45.

Approved

complete the TLX command that was originally issued. **synonym_done** shall be issued as part of the actions taken. It is the responsibility of the AFU to determine if the current state of the line held in its cache that is associated with the **host_tag** allows the completion of the operation. For example when **synonym_detected** is returned and:

- The **host_tag** is found to be invalid. The AFU shall issue **synonym_done** and may retry the operation to obtain the data.
- The **host_tag** is found to be valid and the data is invalid and the operation required valid data. The AFU shall first evict the line using the TLX **castout** command to release the **host_tag** and then issue **synonym_done**. The implementation shall ensure that **castout** precedes **synonym_done** in the VC⁶. Once **synonym_done** is issued, the AFU may retry the operation.
- The **host_tag** is found to be valid, the data is valid, and the state indicated by the **synonym_detected** response does not match the current state of the line.
 - For example, when the AFU cache holds the line in an S state and issues a **read_me** to obtain write authority for the line. The synonym detected is for the existing {EA, address context}. The state of the existing entry is updated with the new state specified by the **synonym_detected** **cache_state** field.
 - For example, when a process is attempting to gain access of a cache line and takes a cache miss, it may issue a **read_me**, **read_mes** or **read_s** command. The receipt of **synonym_detected** indicates that the host proxy cache's state indicates that the line requested has already been provided. Since the current state of the L1 cache entry is invalid, the AFU_{C2} takes the following actions:
 - If the AFU_{C2} does not support synonyms, the line specified by the **host_tag** found in the **synonym_detected** response is cast out and invalidates the **host_tag** entry. The **castout** or **castout.push** and the **synonym_done** commands are both in TLX.vc.2. The **castout** or **castout.push** shall precede the **synonym_done** command.
 - If the AFU_{C2} supports synonyms, then the new cache entry, which is starting the process in an Invalid state, shall be updated using the cache state provided by **synonym_detected**. See Table 2-21 on page 127 in the description of **synonym_detected**.
- The AFU supports mapping a single **host_tag** entry to multiple {EA, address context} entries as described in Section 1.3.2.1 *host_tag database* on page 30. The number of synonyms supported is implementation dependent and the host is not aware of the AFU's capabilities. The AFU_{C2} assigns another pointer in the **host_tag** entry to the new {L1 cache, EA, address_context} and the state of the cache entry is provided by the cache state found in the **synonym_detected** command. The dirty bit in the **host_tag** database entry is set when the M state is specified.

Cache state transitions that are initiated at the AFU are shown in Table 1-3 on page 35. Not all transitions at the L1 are required to be immediately reported to the host.

- As a consequence of an AFU_{C2} EA L1 cache holding synonyms, there are cases where an eviction of a synonym does not result in the AFU informing the host of the eviction. For example, if an AFU cache were to evict one synonym, leaving other synonyms in the AFU_{C2} EA L1 cache, the **host_tag** is still in use because the line is still present in one or more locations in the AFU_{C2} EA L1 cache. A more detailed examination of actions taken when a command from the AFU results in a **synonym_detected** response is found in Table 2-21 on page 127 which is part of the description of **synonym_detected**.

6. TLX.vc.2

Approved

Table 1-3. L1 EA Cache state change request and notification

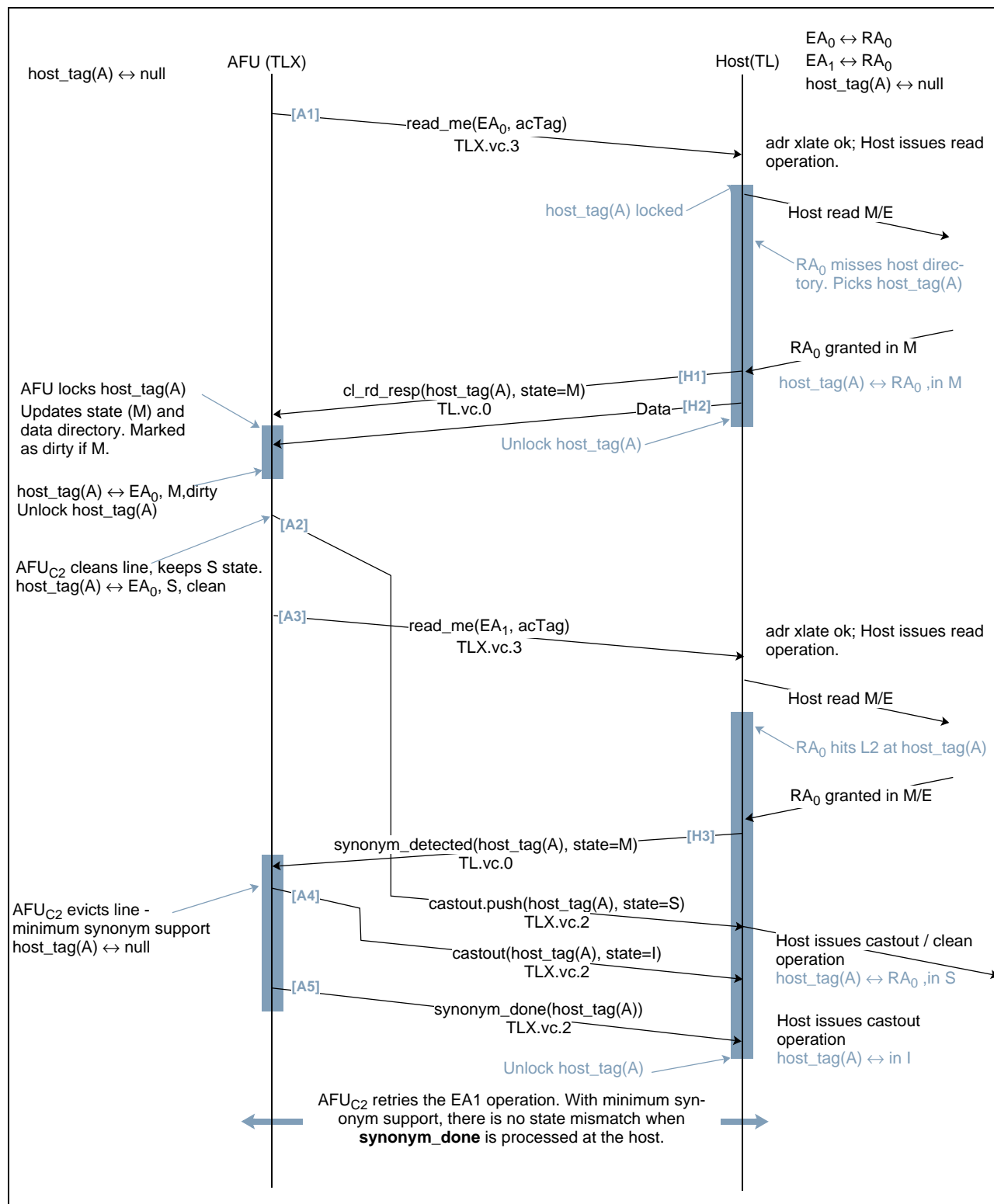
Starting State → Transition to ↓	I	E	E _I	M	S
I	Note 1	castout castout.push Note 2	castout ^{Note 2} castout.push ^{Note 5}	castout.push Note 2	castout Note 2
E	read_me read_mes Note 3	castout ^{Note 4} castout.push ^{Note 5}	castout ^{Note 6} castout.push ^{Note 5}	castout.push Note 5	read_me Note 7
E _I	upgrade_state (I → E _I) Note 3	castout ^{Note 4} castout.push ^{Note 5}	castout ^{Note 4} castout.push ^{Note 5}	castout.push Note 5	Note 8
M	read_me read_mes upgrade_state (I → M) Note 3	castout ^{Note 4} castout.push ^{Note 5}	castout ^{Note 4} castout.push ^{Note 5}	castout ^{Note 4} castout.push ^{Note 5}	read_me Note 7
S	read_mes read_s Note 3	Note 9	Note 9	Note 9	castout Note 6

See Table 7-2 Cache state transition errors on page 208 for illegal state transitions as seen by the host's L2.

Notes:

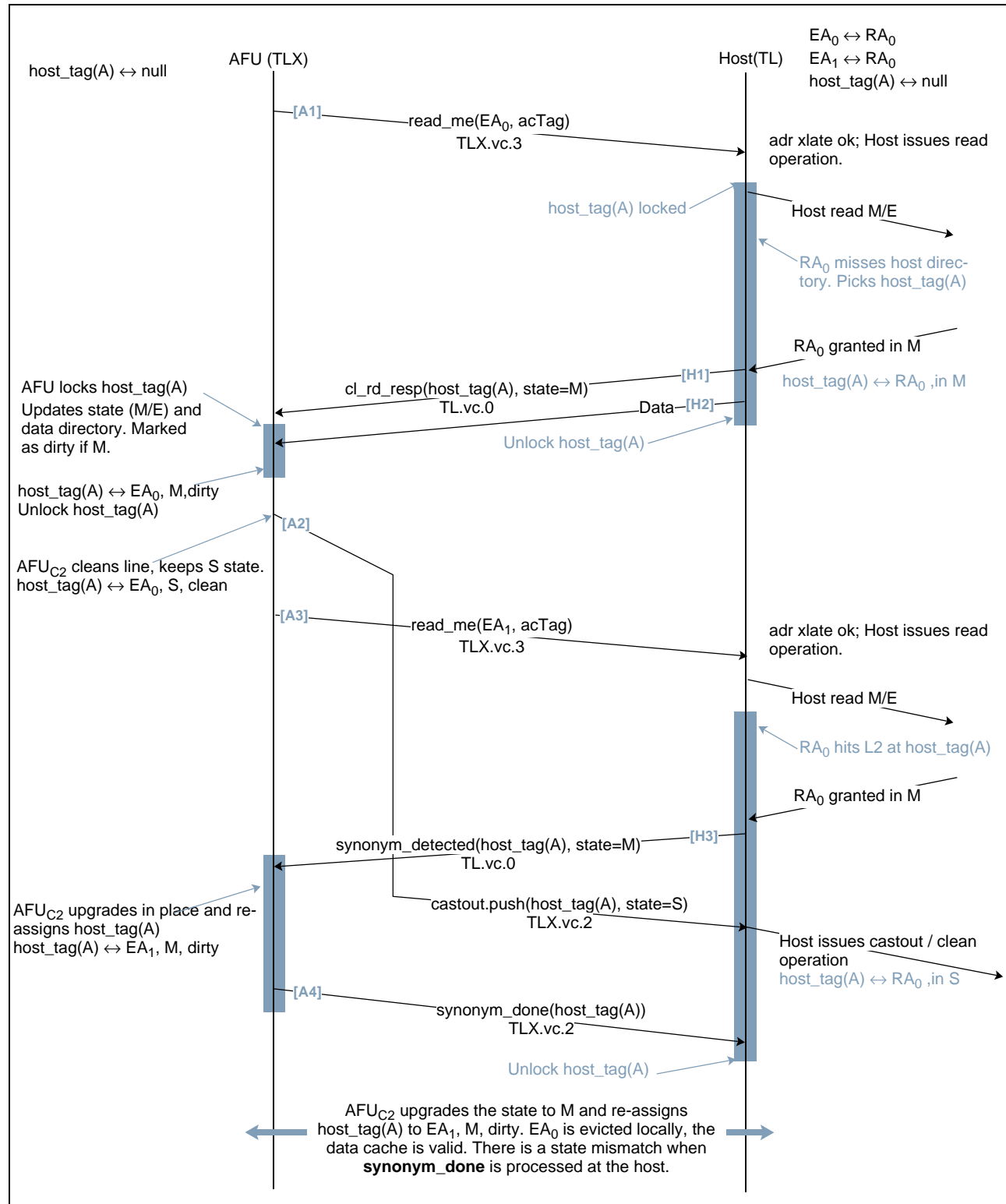
1. Notification not required.
2. Downgrade to an I state shall be reported.
3. The command response specifies the state the line is granted in to the AFU_{C2}.
4. State change is neither an upgrade or downgrade. Immediate notification not required when a data update (clean) is not required.
5. State change is neither an upgrade or downgrade. Immediate notification required when data update (clean) is required.
 - Some multi-step transitions are not visible at the L2, but occur at the L1. For example: E_I → M → I, E → M → E, E_I → M → E
6. State change is neither an upgrade or downgrade. Immediate notification is not required. Allowed only when **castout** indicates an mru_update (cmd_flag = '0001').
7. Upgrade request required response from host granting the upgrade in state. This action results in a **synonym_detected** response.
8. No direct transition. Must castout the line before reacquiring the line in the desired state.
9. A direct downgrade to S state not allowed by architecture due to an unresolved race condition. See Figure on page 36.

Downgrade to S Developer note showing state mismatch condition (Page 1 of 2)



Approved

Downgrade to S Developer note showing state mismatch condition (Page 2 of 2)



1.3.5 Design considerations when the AFU_{C2} and host cache line sizes are different

The architecture supports 64-, 128-, and 256-byte cache line sizes. This allows for a system where the AFU and the host have different cache line size specifications. Since the cache line size specification of the host and the AFU_{C2} may be different, *an AFU_{C2} shall not assume that obtaining a cache line greater than 64 bytes is performed in an atomic fashion by the host.*

1.3.5.1 Read commands

As noted in *Section 2.1.1*, a difference in cache line size may result in a single TLX **read_me**, **read_mes** or **read_s** command getting multiple responses.

read_me, **read_mes** and **read_s** cases:

- A mix of **cl_rd_resp** and **read_failed** responses are received.

The AFU_{C2} received a **cl_rd_resp** which indicates the host has allocated one or more host_tags and assigned a state in its proxy directory. The AFU_{C2} is permitted to

- retry the failed portion of the operation
- Cast out the segment of the cache line that was successfully obtained, and retry the operation. Casting out the segments obtained shall be done indicating a final state of I. This releases the host_tags in the host's proxy directory.

- A mix of states due to multiple **cl_rd_resp** or **synonym_detected** responses.

Multiple responses and a mix of states is an indication that the host has allocated multiple host tags

- A mix of E and M states is resolved by the AFU_{C2} setting the dirty bit in the host_tag data base entries corresponding to the responses indicating an M state. Locally the AFU_{C2} might treat the cache line as being in an M state.
- A mix of M, E, and S states requires that the AFU cast out the cache line segments⁷ held in M and E. The cast out of the segments shall be done indicating a final state of I. The segments can be reacquired by issuing a **read_s** command to ensure that an S state is obtained.

1.3.5.2 Force evict

When a **force_evict** command specifies only a segment of the AFU_{C2} cache line, the AFU_{C2} shall castout the segment of the cache line specified by the **force_evict** command. It may cast out all of the L1 EA cache line.

1.3.5.3 Upgrade state command

As noted in *Section 2.1.1*, a difference in cache line size may result in a single **upgrade_state** command getting multiple responses.

- A mix of **upgrade_resp**, **synonym_detected** and **read_failed** responses are received.

The AFU_{C2} received an **upgrade_resp**, which indicates the host has allocated one or more host_tags and assigned a state in its proxy directory. The AFU_{C2} is permitted to

7. Multiple responses were received. Each response may indicate one or more 64-byte blocks, each having an entry in the host_tag database. A segment refers to block referenced by the response.

- retry the failed portion of the operation as specified by the **read_failed** response.
- Cast out the segment of the cache line that was successfully obtained, and retry the operation. Casting out the segments obtained shall be done indicating a final state of I. This releases the **host_tags** in the host's proxy directory.

1.4 Command ordering

Ordering within a VC is maintained through the TL/TLX, but it is not assured after the command has moved to the upper protocol layers (host and AFU) as described in *Section 3 Virtual channel and data credit pool specification* on page 163.

Developer note

The TLX command set provides posted (dot-p) and non-posted commands with presync directives (dot-s). See *Section 3.3 TL Virtual channel and service queues* and *Section 3.4 TL Presync queues*.

1.5 Host tags

A **host_tag** is a host specified identifier used by an AFU_{C2} when evicting data blocks from the AFU L1 EA cache. The **host_tag** is specified in **synonym_done**, **castout**, and **castout.push** TLX commands. The **host_tag** is specified in the **force_evict** TL command and in the **synonym_detected**, **cl_rd_resp**, **upgrade_resp**, and **cl_rd_resp.ow** TL responses.

While a single **host_tag** field is specified in the above commands and responses, the **host_tag** is associated with a single 64-byte naturally aligned block of data. The commands and responses with **dLength** and **dPart** fields may specify data blocks of 128 or 256 bytes. The **host_tag** found in the command or response is associated with the 64-byte block at offset 0 within the naturally aligned block of data specified by the **dLength** and **dPart**. The **host_tags** associated with the remaining 64-byte blocks are determined using *host_tag arithmetic* defined on page 20. Restrictions on the **host_tag** arithmetic are specified by the host and are provided to the AFU_{C2} through the *host_tag run-length-capability*.

An AFU_{C2} retains the association between a **host_tag** and at least a single {EA, address context} L1 EA cache directory entry within a *host_tag database*. A **host_tag** may be associated with multiple {EA, address context} L1 EA cache directory entries. The AFU_{C2} is managing *synonyms* when a single **host_tag** is associated with multiple {EA, address context} L1 EA cache directory entries. See *host_tag database* on page 30.

The *host_tag database* tracks the association between each **host_tag** and the association to one or more {EA, address context} L1 EA cache directory entries maintained in the AFU_{C2} cache. The host specifies its own restrictions on **host_tags** and the applicability of **host tag arithmetic**.

- Responses with **host_tag** fields and **dPart** fields are a response to a command specifying a **dLength**. For each response, the response's **dLength** refers to the immediate data and/or address range associated with the response. The **dPart** specifies the offset from the address specified by the command the response is associated with. That is, either the immediate data provided by the response or the address range.
- TL responses **synonym_detected**, **cl_rd_resp**, and **upgrade_resp** are formed by the host. The host specifies its own restrictions on **host_tags** and the applicability of **host tag arithmetic**.

- TL response **cl_rd_resp.ow** specifies a single 32-byte naturally aligned data block. The 3 bit dPart field specifies the offset within the command's dLength sized naturally aligned data block. The host_tag is repeated for dPart pairs: {(0,1), (2, 3), (4, 5), (6, 7)}. The host specifies its own restrictions on host_tags and the applicability of host tag arithmetic.
- TL command **force_evict** specifies a naturally aligned block of data to be evicted from an AFU_{C2} cache. The host_tag field specifies the first naturally aligned 64-byte data block. All {EA, address context} copies of the data block associated with the host_tag shall be evicted from the AFU_{C2} cache using a **castout** or **castout.push** command⁸. When the dLength is greater than 64-bytes, host tag arithmetic is used to determine the next host_tag value to be used to determine the data blocks to be evicted.
- TLX command **synonym_done** is a response to a TL response **synonym_detected**. **synonym_detected** specifies a host_tag and dLength. The AFU shall use the same dLength and host tag in forming the **synonym_done** command⁹.
- TLX **castout** or **castout.push** commands shall be issued in response to a TL **force_evict** or due to the *EF* directive found in **upgrade_resp**, **cl_rd_resp**, and **cl_rd_resp.ow** responses. TLX **castout** or **castout.push** commands may be issued as part of normal AFU_{C2} cache management, or the unique AFU processor architecture that may specify pushing a line out of the cache.
 - When the **castout** or **castout.push** is the result of a **force_evict** or due to the *EF* directive found in **upgrade_resp**, **cl_rd_resp**, and **cl_rd_resp.ow**, the AFU_{C2}
 - May use the same dLength and host tag specification for the **castout** or **castout.push** command that was found in the **force_evict** command or due to the *EF* directive found in **upgrade_resp**, **cl_rd_resp** responses.
 - May issue multiple **castout** or **castout.push** commands in response to a **force_evict** or due to the *EF* directive found in **upgrade_resp**, **cl_rd_resp** using smaller dLength field values than the value found in the **force_evict** or due to the *EF* directive found in **upgrade_resp**, **cl_rd_resp** responses and may assume that the set of host tags implied by host_tag arithmetic can be returned individually, in even-odd pairs (offset from the original host tag), or a combination of individual and pairs.
 - Shall issue a single **castout** or **castout.push** using only the host tag in response to a **cl_rd_resp.ow** with an *EF* directive specified.
 - When the **castout** or **castout.push** is the result of normal AFU_{C2} cache management, or the unique AFU processor architecture that may specify pushing a line out of the cache, the AFU_{C2} may issue **castout** or **castout.push** commands with dLength specifying a 64-byte block regardless of the setting of the *host_tag run-length-capability*. That is, specifying a single host tag with each command. Using the *host_tag run-length-capability*, the AFU_{C2} shall issue **castout** or **castout.push** commands using dLength field specifications of
 - 64- or 128-bytes when the run length is 2
 - 64-, 128-, or 256-bytes when the run length is 4.

That is, the run length places a restriction on the dLength specification used by the AFU_{C2}. The AFU_{C2} shall not exceed this restriction.

8. All copies of the line specified by the **force_evict** command are locally evicted and a single castout operation occurs that is visible to the host. The castout operation takes the line to an I state.

9. **synonym_detected** also has a dPart field, so multiple **synonym_detected** responses can be received at the AFU. For each **synonym_detected** response received, the AFU shall respond with a **synonym_done** once all actions required by the **synonym_detected** have been completed.

1.5.1 host_tag run-length-capability

An AFU_{C2} uses the host_tag run-length-capability when specifying **castout** and **castout.push** commands that are not due to a **force_evict** or due to the **EF** directive found in **upgrade_resp**, **cl_rd_resp**, and **cl_rd_resp.ow**. See the discussion above.

The configuration space specification of the host_tag run-length-capability is found in the OpenCAPI Discovery and configuration specification. The description of the capability is shown below.

The host_tag run-length-capability specifies the run length of the host_tag specification. The run length indicates the size of the naturally aligned data block that the hosts assigns to consecutive host tags.

Table 1-4. host_tag run-length-capability definition

run length	Description
1	Indicates that the host assigns a host tag to each 64-byte naturally aligned data block.
2	Indicates that the host assigns {host_tag, host_tag+1} to each 128 byte naturally aligned data block. Within an AFU _{C2} cache two 64-byte data blocks are consecutive only when EA ₀ is on a 128-byte boundary and EA ₀ +64 uses the same address context as EA ₀ .
4	Indicates that the host assigns {host_tag, host_tag+1, host_tag+2, host_tag+3} to each 256-byte naturally aligned data block. Within an AFU _{C2} cache four 64-byte data blocks are consecutive only when EA ₀ is on a 256-byte boundary and EA ₀ +64, EA ₀ +128, and EA ₀ +192 all use the same address context as EA ₀ .
all other values	Reserved.

1.5.2 host_tag update ordering

1.5.2.1 TL and host rules

With respect to **castout** and **castout.push** commands (TLX.vc.2):

- **castout** and **castout.push** commands with different host_tag specifications may be completed by the host protocol layer in any order.
- **castout** and **castout.push** with the same host_tag specification shall be completed by the host protocol layer in the order specified by the command order found in the VC (TLX.vc.2).

This does not place a direct requirement on the host protocol. In cases where the host protocol does not support the above requirements, the TL implementation of the host shall enforce the above rule by any method and in such a manner that the difference between host direct compliance or additional TL implementation actions are not externally observable.

See Figure A-16. *castout.push* example showing host_tag ordering at the host on page 238.

1.5.2.2 TLX and AFU rules

The TLX locks the host_tag entry prior to dispatching the TL packet to the AFU dispatch interface for the following TL commands and responses: **force_evict**, **cl_rd_resp(EF=1)**, **cl_rd_resp.ow(EF=1)** and **synonym_detected**. The AFU responds with a TLX **castout** or **castout.push** command when the TL command is **force_evict**, **cl_rd_resp(EF=1)**, or **cl_rd_resp.ow(EF=1)**. The AFU responds with a TLX **synonym_done** command when the response is to a TL **read_me**, **read_mes**, or **read_s** or **upgrade_state** is **synonym_detected**. There is no response expected from the AFU when the TL response is **cl_rd_resp(EF=0)**.

See *Section 3.5 Device TL virtual channel queues* on page 174

1.6 Write fragmentation ordering and atomicity

1.6.1 Write fragmentation ordering and atomicity at the host

Write commands issued by the AFU may be fragmented by the host. The following sections specify the atomicity of the fragments and the order in which the updates become globally visible.

1.6.1.1 Partial write operations

These are TLX commands found in the following command classifications: `pr_dma_write`, `atomics.r`, `atomics.rw`, and `atomics.w`.

Minimum guaranteed write atomicity is specified as 16 bytes when aligned on a 16-byte address boundary. When the partial write operation is not specified with a naturally aligned address, atomicity may be reduced to a single byte. Data shall be globally visible in increasing address order.

Engineering note

The architecture does not currently provide TLX commands that specify partial write operations where the data and address are not naturally aligned. Unaligned partial writes *can* be specified using, for example, the `dma_w.be` command.

If an implementation breaks the writes specified by the byte enable mask into a series of host write commands, and when a host implementation does not directly support the byte enable mask, the implementation issues the commands in increasing address order and ensures that the results are globally visible in the order the commands are issued.

1.6.1.2 64-, 128-, 256-byte write operations

These are TLX commands restricted to 64-, 128-, or 256-byte naturally aligned write operations. These are TLX commands found in the following command classification: `dma_write`.

Developer Note

`castout.push` does not have an `Os` bit assigned and is not included in this discussion. A process attempting to obtain data held in the L1 cache obtains the data in the L1 cache and are not able to obtain stale data from the MEM as long as the data block is treated coherently.

Minimum guaranteed write atomicity is specified as 64 bytes. When the write operation specifies 128 or 256 bytes and the host fragments the write operation, ordering is controlled by the ordered segment bit found in the command.

When the ordered segment bit is set to 0, there is no ordering guarantee for the data segments written.

- When the ordered segment bit is set to 1, each segment is written atomically and increasing address order.

Developer note

Applications are expected to use 64-byte or smaller writes for synchronizing required events; for example, when writing a semaphore. Larger data transfer sizes are expected to be used for data transfer efficiency. Synchronizing control events are expected to use presync (dot-s) variants of write commands.

Ordering is provided between data segments of a single command as well as between commands using presync versions of write commands. Both segment ordering and presync have performance implications and should be used only when necessary.

1.6.2 Write fragmentation ordering and atomicity at the AFU

Write commands issued by the host may be fragmented by the AFU. The following sections specify the atomicity of the fragments and the order in which the updates become globally visible.

1.6.2.1 Partial write operations

These are TL commands found in the following command classifications: `mem_atomics.r`, `mem_atomics.rw`, `mem_atomics.w`, `pr_mem_write`, and configuration.

Minimum guaranteed write atomicity is specified as 16 bytes when aligned on a 16-byte address boundary. When the partial write operation is not specified with a naturally aligned address, atomicity may be reduced to a single byte. Data shall be globally visible in increasing address order.

Developer note

The architecture does not currently provide TL commands that are able to specify partial write operations where the data and address are not naturally aligned.

1.6.2.2 64-, 128-, 256-byte write operations

These are TL commands restricted to 64-, 128-, or 256-byte naturally aligned write operations. These are TL commands found in the following command classification: `mem_write`.

Minimum guaranteed write atomicity is specified as 64 bytes. When the write operation specifies 128 or 256 bytes and the AFU fragments the write operation, ordering is controlled by the ordered segment bit found in the command.

- When the ordered segment bit is set to 0, there is no ordering guarantee for the data segments written.
- When the ordered segment bit is set to 1, each segment is written atomically and increasing address order.

1.7 OpenCAPI device PA space specification

An OpenCAPI device may have the following three PA spaces specified:

1. Configuration space shall be specified for the device.
2. System memory space may be specified for the device.
3. MMIO space may be specified for the device.

Engineering note

MMIO space notes: Access to MMIO should be non-blocking. Retrying access to MMIO space is not prohibited by the architecture.

The configuration space is accessed by using the **config_read** or **config_write** commands. The PA specified for this space is separate from the system memory space and the MMIO space. The host may:

- Provide a configuration address BAR to access this space using a direct access load/store model.
- Provide an MMIO register set to access this space using an indirect access method.

This architecture does not specify the application of *metadata* to a device's configuration space.

The system memory space is memory space owned by the OpenCAPI device that is mapped to the host's system memory. The PA for system memory space is defined to start at offset 0. The host differentiates between the different system memory spaces of different OpenCAPI devices by providing a configuration address BAR for each attached device.

The MMIO space shares the PA space used by the system memory space. It is specified by a fixed offset from PA 0 which is specified in the OpenCAPI device's configuration space. The host differentiates the MMIO spaces of different OpenCAPI devices by providing a configuration address BAR for each attached device. Access to MMIO space is sensitive to the operand length and the command specified. The device literature should provide information on how to correctly access MMIO space.

- A device may not support all operand lengths provided by the architecture when accessing a specific address found in MMIO space. If an MMIO access does not use a correct operand size for the address specified, an unsupported-operand-length Resp_code shall result.
- A device may not support all commands provided by the architecture when accessing a specific address found in MMIO space. If an MMIO access does not use a correct command for the address specified, a Failed Resp_code shall result.
- All accesses to MMIO space shall result in a single response from the device. That is, when a dLength of 128 or 256 bytes is permitted, the device shall respond with the same dLength used in the command.

System and MMIO spaces are expected to be contiguous based on the configured starting PA and size. Access to unimplemented addresses results in the following:

- Read access to an unimplemented PA shall return all 1s data.
- Write access to an unimplemented PA shall result in discarded data.

1.7.1 PA-to-RA mapping rules

Real addresses (RA) are mapped into the physical address (PA) space specified for a device. This eliminates any requirement placed on the OpenCAPI device to have knowledge of the host's real address space or how the OpenCAPI device's PA space is mapped into it. The following rules place restrictions on the OpenCAPI device's specification of its PA space.

1. No address aliasing for PA-to-RA translation. That is, a PA for any specific device attached to an OpenCAPI link (PA + unique interface) translates into a unique RA. The address translation is specified by address ranges configured by software.
2. No address aliasing for RA-to-PA translation. The host protocol is provided with a single POC for each RA.

1.8 Address translation

1.8.1 Effective to real address translation

Effective to real address translation is specified by the host's architecture. The TL architecture model assumes a host address translation cache (ATC) that holds valid effective (EA) to real address (RA) translations. The ATC contains, at a minimum, a valid indication, the page size, the page size aligned starting effective address (EA), the address context of the translation in the form of the BDF and PASID, page write permission (W) and the host's corresponding real address.

The architecture supports multiple page sizes. All implementations shall support the minimum page size of 4K-bytes. See the specification of *log₂_page_size* on page 52 and page size capability recommendations found in *Table 8-9 Profile specifications supported page size* on page 217. During configuration, the set of page sizes used by host and AFU is determined by taking the intersection of the set of page sizes supported by the host and the set of page sizes supported by the AFU.

An implementation may choose to provide additional fields, or may replace some of the fields listed above with other host specific content. Since the architecture assumes that the contents of an ATC entry contain the fields specified by the TL architectural model, any implementation specific alterations shall be done in such a manner that the differences are not externally observable.

The architecture model does not require, but allows for, a multi-level ATC. Higher level ATC might have a smaller capacity and have faster access than a lower level ATC that have more capacity and longer access latency. The structure of a host's ATC is outside the scope of this architecture.

The TL architectural model assumes that TLX commands with an EA specified go through effective to real address translation before execution on the host protocol bus. See *Section 3.3 TL Virtual channel and service queues* on page 168 for additional details.

An AFU can warm up the host's ATC by using the TLX **xlite_touch** command. See the command description for additional details.

1.8.2 Translated addresses, AFU ATC, and dot-t commands

All dot-t commands have a translated address (TA) specified. To obtain a TA, the AFU may¹⁰ issue a **xlite_touch** command with the translated address request option asserted as specified by the command flag bit 3 being set to '1'.

A successful address translation results in the TL returning a **touch_resp.t** that provides a page size aligned translated address (TA), page write permission (W), memory hit (*mem_hit*) indication, and the size of the page. The AFU uses the provided base TA and adds in the offset into the page when accessing data within the range of the page.

The TA returned is unique to the address context associated with the **xlite_touch**. The same TA value may be used for different {EA, address context} translations. **touch_resp.t** returns a {TA, address context} pair. The AFU shall make the association between the TA returned and the address context when updating its ATC on receipt of the TL **touch_resp.t** response.

10. Other methods are permitted by the architecture. Those methods are outside the scope of the architecture.

The TL architecture model assumes an AFU address translation cache (ATC) that holds valid effective {EA, address context} to {TA, address context} address translations. The AFU ATC contains, at a minimum, a valid indication, the page size, the page size aligned starting effective address (EA), the address context of the translation in the form of the BDF and PASID, page write permission, and the page size aligned translated address (TA). An implementation may choose to provide additional fields, or may replace some of the fields listed above with other AFU specific content. Since the architecture assumes that the contents of an AFU ATC entry contain the minimum set of fields specified by the TL architectural model, any implementation specific alterations shall be done in such a manner that the differences are not externally observable.

The architecture supports multiple page sizes. All implementations shall support the minimum page size of 4K-bytes. See the specification of *log₂_page_size* on page 52 and page size recommendations found in *Table 8-9 Profile specifications supported page size* on page 217. During configuration, the set of page sizes used by host and AFU is determined by taking the intersection of the set of page sizes supported by the host and the set of page sizes supported by the AFU.

When supporting an AFU ATC, the host address translation cache is modified from the description provided in *Section 1.8.1* beginning on page 45 by the addition of translated addresses (TA). The TA field is added to the host's ATC in order to support the use of TA by the AFU. The host's ATC shall be accessible by either EA or TA. Since the architecture assumes that the contents of an ATC entry contain the fields specified by the TL architectural model, and assumes the access methods described, any implementation specific alterations shall be done in a manner such that the differences are not externally observable.

Engineering note

Multiple host ATC and TA implementation approaches are possible to meet the architectural model's requirements. For example:

- Pin the host ATC entry. When implemented in this manner, the TA has the same value as the EA. A field is added to the host ATC entry indicating that the ATC entry is pinned.
- TA database. This is a database containing the same fields as an ATC entry where the TA value and the contents found in a host ATC entry are provided. The TA is the index into the TA database.

When requesting and releasing translated addresses:

- An AFU making multiple requests for the same {EA, address context} may either receive the same TA value or may receive different TA values. The address context is the same.
- Regardless of the number of times the AFU obtains the same TA value, a single **xlata_release** releases the {TA, address context}.

1.8.2.1 AFU initiated AFU ATC entry invalidation

The AFU manages its ATC without direction from the host. The implementation dependent topology of the ATC may require that an entry must be removed before a new entry can be added. This may be accomplished by the AFU issuing a **xlata_release** command or the AFU may evict the ATC entry without notifying the host. When using **xlata_release**, the command shall be enqueued into its VC after all commands dependent on the translation.

Since **xlata_release** is assigned to TLX.vc.3 the service queue hash¹¹ used by the host may include the stream_id. In cases where the address translation is in use by multiple stream_id, the AFU shall stop issuing commands using the address translation and then issue a **sync** command to all stream_id using the translation. Once all **sync** commands have received a response¹², the AFU sends **xlata_release** to any *one* of the stream_id to return the translation to the host.

1.8.2.2 Host initiated AFU ATC entry invalidation

For various reasons¹³ the host might require that an address translation held by the AFU ATC be invalidated. This is accomplished by the host requesting the TL to issue a **kill_xlate** command. The AFU shall complete all currently active operations using the address translation specified by the command and invalidate its AFU ATC entry. New operations shall not be started using the ATC entries specified by the **kill_xlate** command. All TLX commands using these address translations shall be placed into the TLX.vc.3 before the **kill_xlate** command can be completed by sending a **kill_xlate_done**. See *Section 3.4.1 TL queuing and service of kill_xlate_done* on page 173 for additional detail.

11. See the specification of service queue hash in the definition of a *service queue* on page 23.

12. **sync_done**

13. For example, invalidation of the address translation by the host's operating system, or the host's management of its ATC requires an entry be removed before adding a new host ATC entry.

2. TL and TLX command and response specifications

This section specifies all command and response types originated in the TL and the TLX. Commands originating in the TL are referred to as CAPP command packets (CAPP_cmd). Commands originating in the TLX are referred to as AP command packets (AP_cmd). Responses originating in the TL are referred to as CAPP response packets (CAPP_response). Responses originating in the TLX are referred to as AP response packets (AP_response)

In the subsections of this chapter descriptions use the following format:

Command descriptive name	mnemonic	Assigned opcode
command classification	VC used, DCP used (immediate data)	28-bit slot count

Table 2-1 lists the command operands used in the TL and TLX command and response specifications. See *Terms* on page 17 for definitions of terms used in these specifications.

Table 2-1. TL and TLX command operands (Page 1 of 6)

Operand mnemonic	Field width	Description
acTag	12	Address context tag. The address context tag is managed by the AFU. The acTag is used as an index into a host table that contains the BDF and PASID associated with the acTag. The OpenCAPI device learns its Bus number during a config_write , T=0 operations. The function and device numbers are assigned by the attached OpenCAPI device's implementation and cannot be modified by any configuration actions. The OpenCAPI device shall be assigned at least one PASID, and may be assigned more than one PASID, by host software during the initialization and operation of the device. The BDF and PASID are used for address translation authorization and operation validation.
AFUTag	16	<p>Unique handle specifying the AFU and command instance. Provided by the AFU that is requesting command services of the TLX. A TL response to a single TLX command may be broken into multiple TL response packets. When this occurs, all responses associated with the TLX command shall return the same AFU Tag value.</p> <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <p style="text-align: center;">Engineering Note</p> <p>The TL shall not use the AFUTag for any purpose other than as data to complete the contents of a response packet, or when forming an xlata_done or intrp_rdy TL command packet. Any retirement rules specified by a device implementation for the AFUTag shall not be checked by the TL.</p> <p>AFU tag retirement recommendations:</p> <ul style="list-style-type: none"> For non-posted commands, the AFU should not reuse an AFUTag until all responses for the command have been received. For posted commands, the AFU tag is not used by the TL and is not returned to the TLX. There are no recommendations when AFUTag values can be re-used. </div>
BDF	16	Bus device function. This is the identifier of a TLX requester. See <i>acTag</i> on page 48 for additional details.
Byte enable	64	(BE) This field is found in commands with dot-be mnemonic specifications. Valid only for write class commands.

Approved

Table 2-1. TL and TLX command operands (Page 2 of 6)

Operand mnemonic	Field width	Description
cache_state	3	<p>Specifies the cache state the cache line has obtained. The minimum data block size granularity associated with a cache state assignment is 64 bytes. This specification defines 4 states:</p> <p>000 Invalid (I). 001 Shared (S). Read only cached copy of the line. To modify the line, the state must be upgraded to an M or E state. 010 Exclusive (E). No other cached copies exist in the system. The line is unmodified with respect to the copy held in the MEM (memory). 011 Modified state (M). An exclusive state where the line is modified with respect to the copy held in the MEM (memory). 100 Exclusive with no valid data held (E_I). No other cached copies exist in the system. Data is held by the MEM.</p> <p>All other encodes are reserved. See <i>Table 1-1 Cache state descriptions</i> on page 32 for a full description of the cache states. See <i>Table 7-2 Cache state transition errors</i> on page 208 for the specification of legal and illegal cache state transitions due to castout and castout.push commands. See the descriptions of upgrade_state and read_me TLX commands for supported upgrade cache transitions.</p> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <p>Engineering Note</p> <p>cl_rd_resp.ow</p> <p>A single read request for a data block greater than 32 bytes results in multiple responses when cl_rd_resp.ow is used to return the data. The minimum granularity of a cache state assignment to a data block in memory is 64 bytes. For multiple cl_rd_resp.ow responses to a single command, the value of the cache_state field shall be the same for dPart field values of {0, 1}, {2, 3}, {4, 5}, and {6, 7}.</p> </div>
CAPPTag	16	<p>Unique handle specifying the host CAPP and command instance. Provided by the CAPP that is requesting command services of the TL.</p> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <p>Engineering Note</p> <p>The TLX shall not use the CAPPTag for any purpose other than as data to complete the contents of a response packet. Any retirement rules specified by the host implementation for the CAPPTag shall not be checked by the TLX.</p> <p>CAPPTag retirement recommendations:</p> <ul style="list-style-type: none"> For non-posted commands, the CAPP should not reuse a CAPPTag until all responses for the command have been received. For posted commands, the CAPP tag is not used by the TLX and is not returned to the TL. There are no recommendations when CAPPTag values can be re-used. </div>
cmd_flag	4	Specifies execution behavior for commands and responses specified with this field. The command or response specification includes the behavior specification for the cmd_flag when the field is specified.
cmd_opcode	8	Specifies the operation to be performed.
credit_return	48	<p>Specifies the number of credits returned to the VC and DCP credit pools. The credits are returned in fixed subfield locations in a 2-slot (56-bit) TL or TLX response packet. See the specification for return_tl_x_credits and return_tl_credits for the format of the field.</p> <p>Each VC credit allows for a single command or response to be sent in the virtual channel.</p> <p>Each DCP credit allows the sending of one <i>data carrier</i>.</p>

Approved

Table 2-1. TL and TLX command operands (Page 3 of 6)

Operand mnemonic	Field width	Description
dLength	2	<p>Data length (dL). Indicates the number of data bytes associated with a command or response packet. This 2-bit field indicates a length of:</p> <p>00 32 bytes when the command is pad_mem or when in response to a pad_mem command is mem_wr_response or mem_wr_fail. Reserved for all other commands and responses.</p> <p>01 64 bytes. This field value shall be used in a response packet when the command is a partial read or write operation.</p> <p>10 128 bytes. Reserved when the command is a partial read or write operation.</p> <p>11 256 bytes. Reserved when the command is a partial read or write operation.</p> <p>When the dLength field in the response packet does not match the full amount of data requested by the command, the dPart field is used to indicate the offset within the <i>naturally aligned data block</i> specified by the command's address. For example, in the multiple responses to a single TLX read command, the AFUtag is unchanged. That is, the dLength may vary and the dPart shall vary when multiple responses are returned for a single command.</p> <p>For multiple responses to a single command, there is no order requirement placed by the architecture. That is, continuing with the above example, the TLX may see the values of dPart returned in any order.</p> <p>Support for 256 bytes is optional for the TLX and AFU. See <i>Table 8-10 Profile specifications supported dLength by TLX</i> on page 217.</p>
dPart(1:0)	2	<p>Data part (dP(1:0) or dPart(1:0)). Indicates the data content of the current response packet. Read requests can be 64, 128, or 256 bytes in length. This field indicates the starting offset from the naturally aligned data block specified by the address provided in the read command. The amount of data transferred due to this response packet is found in the dLength field.</p> <p>00 Offset at 0 bytes. This field value shall be used for response packets when the command is a partial read or write operation.</p> <p>01 Offset at 64 bytes. This field value <i>shall not</i> be used when the dLength specifies 128 or 256 bytes. Reserved when the command is a partial read or write.</p> <p>10 Offset at 128 bytes. This field value <i>shall not</i> be used when the dLength specifies 256 bytes. Reserved when the command is a partial read or write.</p> <p>11 Offset at 192 bytes. This field value <i>shall not</i> be used when the dLength specifies 128 or 256 bytes. Reserved when the command is a partial read or write.</p> <p>The presence of this field in a command allows for multiple responses to be returned for a command. For example, a 256-byte read command such as rd_wntc may result in four responses with the dPart field taking on all four states.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>Developer note</p> <p>The constraints placed on dPart are to ensure that responses specify naturally aligned data blocks. This is intended to simplify the design and the verification state space.</p> </div>
dPart(2:0)	3	<p>Data part (dP(2:0) or dPart(2:0)). Indicates the data content of the current response packet. This field indicates the starting offset from the naturally aligned data block specified by the address provided in the read command. This extended version of dPart is used for dot-ow variants of cl_rd_resp, mem_rd_response, and read_response.</p> <p>The extended field width allows the specification of offsets with 32-byte granularity. The data is sent in 32-byte <i>data carriers</i>. For example, a 256-byte read command such as rd_wntc results in eight responses with the dPart(2:0) field taking all eight states. A pr_rd_wntc uses a single response and is restricted to an offset of 0 bytes.</p> <p>000 Offset at 0 bytes.</p> <p>001 Offset at 32 bytes. Reserved when the command is a partial read or write.</p> <p>010 Offset at 64 bytes. Reserved when the command is a partial read or write.</p> <p>011 Offset at 96 bytes. Reserved when the command is a partial read or write.</p> <p>100 Offset at 128 bytes. Reserved when the command is a partial read or write.</p> <p>101 Offset at 160 bytes. Reserved when the command is a partial read or write.</p> <p>110 Offset at 192 bytes. Reserved when the command is a partial read or write.</p> <p>111 Offset at 224 bytes. Reserved when the command is a partial read or write.</p>

Approved

Table 2-1. TL and TLX command operands (Page 4 of 6)

Operand mnemonic	Field width	Description
E	1	Operand endianness. Used for mem_atomics.* and atomics.* class commands to specify the endianness of the operands. For bitwise logical operations, the endianness of the operands does not change the result. The field is specified as follows: 0 Operands are little endian. 1 Operands are big endian.
EA	52, 59, 64	Effective address (also referred to as the VA or virtual address by some host architectures). Length specification is dependent on the command issued and is noted in the command specification.
EF	1	Evict and Fill. This directive is used in upgrade_resp , cl_rd_resp and cl_rd_resp.ow to indicate if the host_tag specified by the command is being re-assigned. When asserted, the entry in the L1 cache indicated by the current state of the <i>host_tag database host_tag entry</i> is to be evicted from the L1 cache before installing the new data and cache state specified by this response. Engineering Note cl_rd_resp.ow A single read request for a data block greater than 32 bytes results in multiple responses when cl_rd_resp.ow is used to return the data. The granularity of a host tag is 64 bytes. For multiple cl_rd_resp.ow responses to a single command, the value of the host_tag and EF field shall be the same for dPart field values of {0, 1}, {2, 3}, {4, 5}, and {6, 7}.
host_tag	24	Unique identifier provided by the host to the AFU associated with a 64-byte address aligned data block held in the AFU L1. <ul style="list-style-type: none"> The AFU L1 is required to retain the association between the host_tag provided and the AFU L1 entry or entries. The host uses this field when specifying cl_rd_resp, upgrade_resp, upgrade_resp, and force_evict. A single host_tag value applies to an address aligned 64-byte data granule. In combination with a dLength field, a single host_tag specification in a command may infer up to 4 host_tag values. See <i>host_tag arithmetic on page 20</i>. <p>The minimum supported host_tag width shall be 6 bits and the largest supported host_tag width shall be 24 bits. See <i>Section 1.3.2.1 host_tag database on page 30</i>.</p> Engineering Note cl_rd_resp.ow A single read request for a data block greater than 32 bytes results in multiple responses when cl_rd_resp.ow is used to return the data. The granularity of a host tag is 64 bytes. For multiple cl_rd_resp.ow responses to a single command, the value of the host_tag field shall be the same for dPart field values of {0, 1}, {2, 3}, {4, 5}, and {6, 7}. Engineering note The width of the host_tag is a capability specified by the OpenCAPI device. The device specifies the width of host_tag supported and the host constrains the host_tag based on the host's proxy cache implementation and the device's capability. The configuration space specification of the host_tag run-length-capability is found in the OpenCAPI Discovery and configuration specification. The description of the capability is found in <i>Section 1.5.1 host_tag run-length-capability on page 41</i> .

Approved

Table 2-1. TL and TLX command operands (Page 5 of 6)

Operand mnemonic	Field width	Description
log ₂ _page_size	6	<p>Log₂ value of the page size determined by the host when executing xlate_touch. The value of the page size touched in bytes is specified as $2^{\text{bin2dec}(\log_2_page_size)}$. The values supported by the host and the attached OpenCAPI device are capabilities determined during configuration. <i>Table 8-9 Profile specifications supported page size</i> on page 217 contains the recommended support for the host and the OpenCAPI device. Only the intersection of the host and the OpenCAPI device's supported page sizes are used during operation.</p> <p>A value of '00 0000' is used in a touch_resp when an age out ATC request results in a miss to the ATC. That is, the field is reserved. See <i>Figure 2-1 Address translation sequence: xlate_touch</i> on page 108.</p> <p>A minimum page size of 4KB shall be supported by all implementations. Additional page sizes may be supported and are included in the capability specifications of the host and the attached OpenCAPI device. See <i>Table 8-9 Profile specifications supported page size</i> on page 217. Values corresponding to a page size of 1-byte to 2K-bytes are reserved.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>Developer Note</p> <p>The specification of touch_resp.t is dependent on this minimum page size since the command format does not have enough bits to specify a smaller page size.</p> </div>
MAD	8	<p>Memory access directive. Specifies host directives when accessing OpenCAPI memory devices (AFU_M). The specification of this field is found in the host's platform architecture.</p> <p>The AFU_M may ignore this field.</p>
Meta	7	<p>Metadata. This 7-bit field specifies the metadata for a data block held in memory. The size of the data block is implementation dependent. The TL architecture specifications provides for 7 bits of metadata for 8-, 32- and 64-byte data blocks.</p> <p>The specification of the metadata is outside the scope of this architecture and is found in the host and OpenCAPI device's documentation.</p> <p>An implementation shall transform the metadata, if necessary, when 8- and 32-byte naturally aligned data blocks are aggregated into 64-byte naturally aligned data blocks.</p>
mem_hit	1	<p>MEM hit (mh). Used in touch_resp.t. When this field is set to '1', the host is indicating that during address translation it was determined that the requester is the MEM owner of the page.</p>
Object_handle	64/68	<p>Used by message class commands.</p> <p>For TL commands, the object handle is specified by the OpenCAPI device manufacturer and is loaded into a table maintained by the device software. It is accessed by the host based on a method specified by the host's OpenCAPI platform architecture.</p> <p>For TLX commands, the object handle is specified by the host architecture and is loaded into a table held in the device's MMIO space. The OpenCAPI device manufacturer specifies the location of the MMIO space, and it is provided to the host through the device software.</p>
Os	1	<p>Ordered segment. Controls ordering between segments for 128 or 256 byte write operations. This bit has no affect when the write command indicates a 64 byte transfer.</p> <ul style="list-style-type: none"> When the ordered segment bit is set to 0, there is no ordering guarantee for the data segments written. When the ordered segment bit is set to 1, each segment is written atomically and increasing address order.
PA	58, 59, 64	<p>Physical address. Translation from the host RA to the AFU_M PA is performed by the host and configured during device initialization.</p> <p>The AFU's configuration space provides the information about the topology of the physical address space held by the OpenCAPI device. Types of address spaces are:</p> <ul style="list-style-type: none"> Address space shared with the system. This excludes MMIO space. MMIO space. Configuration address space. This address space may be directly memory mapped and use a simple load/store model, or it may be accessed using indirect address methods. The choice is host dependent and is transparent to the OpenCAPI device and TL protocol.
PASID	20	<p>This term identifies the user process associated with a request. In OpenCAPI, a request is a TLX command. See <i>acTag on page 48</i> for additional details.</p>

Approved

Table 2-1. TL and TLX command operands (Page 6 of 6)

Operand mnemonic	Field width	Description
pLength	3	<p>(pL) Partial length. Specifies the number of data bytes specified for a partial write command. The address specified shall be naturally aligned based on the pLength specified. The data may be sent in a data flit, or an 8- or 32-byte data field specified for some control flits.</p> <p>000 1 byte. Reserved when the command is an amo*.</p> <p>001 2 bytes. Reserved when the command is an amo*.</p> <p>010 4 bytes. Reserved when the command is amo_rw and the operation is specified as a Fetch and swap. That is the command flag is {x'8'..x'A'}.</p> <p>011 8 bytes. Reserved when the command is amo_rw and the operation is specified as a Fetch and swap. That is the command flag is {x'8'..x'A'}.</p> <p>100 16 bytes. Reserved when the command is an amo*.</p> <p>101 32 bytes. Reserved when the command is an amo*.</p> <p>110 Specifies 4-byte operands when the command is amo_rw and the operation is specified as a Fetch and swap. That is, the command flag is {x'8'..x'A'}. Otherwise, this field is reserved.</p> <p>111 Specifies 8-byte operands when the command is amo_rw and the operation is specified as a Fetch and swap. That is, the command flag is {x'8'..x'A'}. Otherwise, this field is reserved.</p>
Resp_code	4	Response code. On a failed transaction, this field is found in a response packet reporting the reason the transaction failed. See the response packet for encoding and specifications. "Done" is not typically an included encoding because the response packet used is different for a failed transaction. For example, in response to a rd_wntc AP command, the read_failed (TL response) is sent when the read is not able to complete successfully. The read_response (TL response) is used to indicate a successful completion and that data is associated with the response. A response code of "done" is implied with the read_response .
stream_id	4	Stream identifier used by the AP. This is used as part of the virtual channel, virtual queue, service queue specification.
T	1	<p>Configuration read or write command type.</p> <p>0 Indicates a type 0 configuration read or write command. A config_write, T=0 shall be used by the AFU to learn its bus number. For config_read with TL=0, the bus number is unchecked.</p> <p>1 The operation shall result in a mem_wr_fail or mem_rd_fail TLX response with a Resp_code = Failed.</p>
TA	52, 58, 59, 64	<p>Translated Address. The AFU has obtained a translated address and shall use it with TLX commands that have a dot-t format. The translated address is associated with a specific page in memory and the size of the page is known. See <i>Section 1.8.2 Translated addresses, AFU ATC, and dot-t commands</i> on page 45.</p> <p>A translated address and page size is obtained using xlata_touch.</p>
W	1	Write permission. Returned with touch_resp and touch_resp.t . When asserted indicates that write authorization is granted for the specified page.

2.1 Handling multiple responses to a single command

As noted in the description of some responses, a single command may receive multiple responses. This might be due to a mismatch between the host's and OpenCAPI device's maximum data length specification.

For example, the host's or the device's internal bus protocol might be limited to atomically accessing 64 bytes of data. Read and write cases are examined in the following sections.

2.1.1 TLX Read request getting multiple TL responses

A 128-byte read request by the OpenCAPI device may be broken into two 64-byte read requests on the host protocol bus. This results in two TL responses returning data to the OpenCAPI device. The responses are not returned in any specified order. After all the responses are returned to the requester (the OpenCAPI device in this example), the requester examines the responses.

- When all responses indicate success, the command has completed successfully. In this example, each response provides 64 bytes of data, fulfilling the OpenCAPI device's request for 128 bytes.
- When all responses indicate failure, the command has failed. In this example, no data has been returned.
- When one response indicates success and the other indicates a failure, a non-cacheable command has failed. In this example, only 64 bytes of the requested 128 bytes have been returned. When this is a non-cacheable read request, the data may be discarded. Depending on the Resp_code and the TL response, the entire operation or just the failing portion may be retried. Refer to the specification of the TL response for when the operation may be retried.

When this is a cacheable read request, the AFU protocol is more complex and is discussed in *Section 1.3.5 Design considerations when the AFU_{C2} and host cache line sizes are different* on page 38.

The following TLX read commands may receive multiple responses.

- **rd_wntc, rd_wntc.n, rd_wntc.t, rd_wntc.t.s, upgrade_state, upgrade_state.t, read_me, read_me.t, read_mes, read_mes.t, read_s, read_s.t**

These read commands, when receiving multiple TL responses, shall see only the following responses¹⁴:

- **synonym_detected, read_response, read_response.ow, upgrade_resp, cl_rd_resp, cl_rd_resp.ow, read_failed**

2.1.2 TLX Write request getting multiple TL responses

A 128-byte write request by an OpenCAPI device may be broken into two 64-byte operations on the host protocol bus. When the write request is non-posted, This results in two TL responses indicating the status of the write operation in the host. The responses are not returned in any specified order. After all the responses are returned to the device, the device examines the responses.

- When all responses indicate success, the command has completed successfully. The write operation has completed and the changes to the specified memory locations are globally visible.
- When all responses indicate failure, the command has failed. The locations in memory specified by the command may have been modified by the failed operation. That is, the data at the locations may be unmodified, may contain undefined data, or may contain SUE data. The Resp_code field in the fail response indicates what might have occurred at the memory location specified by the write command.
- When one response indicates success and the other indicates failure, the command has failed. Only the data corresponding to the 64-byte block specified by the successful response has completed its operation in the host and the changes to the specified memory location are globally visible. The data corresponding to the 64-byte block specified by the failed response shall contain SUE data. Depending on the Resp_code and the TL response, the failing portion may be retried and the successful portion shall not be retried. Refer to the specification of the TL response for when the operation may be retried.

14. Not all responses apply to all commands. See the command descriptions for applicable responses.

Approved

The following TLX write commands may receive multiple responses:

- **dma_w, dma_w.n**

These write commands, when receiving multiple TL responses, shall see only the following responses.

- **write_response, write_failed**

2.1.3 TL read request getting multiple TLX responses.

A 128-byte read request by the host to an OpenCAPI device may be broken into two 64-byte read requests at the OpenCAPI device. This results in two TLX responses returning data to the host. Further, the responses are not returned in any specified order. After all responses are returned to the host, the host examines the responses.

- When all responses indicate success, the command has completed successfully. In this example, each response provides 64-bytes of data, fulfilling the host's request for 128 bytes.
- When all responses indicate failure, the command has failed. No data has been returned.
- When one response indicates success and the other indicates failure, the command has failed. In this example, only 64-bytes of the requested 128 bytes have been returned. The data obtained may be discarded. Depending on the Resp_code and the TLX response, the entire operation or just the failing portion may be retried.

The following TL read commands may receive multiple responses:

- **rd_mem**

These read commands, when receiving multiple TLX responses, shall see only the following responses:

- **mem_rd_response, mem_rd_response.ow, mem_rd_fail**

2.1.4 TL write request getting multiple TLX responses

A 128-byte write request by the host to an OpenCAPI device may be broken into two 64-byte write requests at the OpenCAPI device. This results in two TLX responses indicating the completion status of the write operation in the OpenCAPI device. Further, the responses are not returned in any specified order. After all the responses are returned to the host, the host examines the responses.

- When all responses indicate success, the command has completed successfully. The write operation has completed and the changes specified by the memory locations are globally visible,
- When all responses indicate failure, the command has failed. The locations in memory specified by the command may have been modified by the failed operation. That is, the data at the locations may be unmodified, may contain undefined data, or may contain SUE data. The Resp_code field in the fail response indicates what might have occurred at the memory location specified by the write command.
- When one response indicates success and the other indicates failure, the command has failed. Only the data corresponding to the 64-byte block specified by the successful response has completed its operation in the OpenCAPI device and the changes to the specified memory location are globally visible. The data corresponding to the 64-byte block specified by the failed response may be unmodified, may contain undefined data, or may contain SUE data. The contents of the data block is dependent on the address of the command. Depending on the Resp_code and the TLX response, the entire operation or just the failing portion may be retried. Refer to the specification of the TL response for when the operation may be retried and the state of the data block when the response indicates a failure.

Approved

The following TL write commands may receive multiple responses:

- **write_mem**

These commands, when receiving multiple TLX responses, shall see only the following responses:

- **mem_wr_response, mem_wr_fail.**

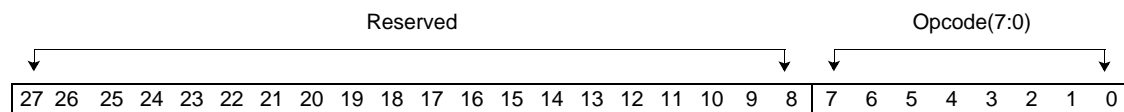
Approved

2.2 TL CAPP command packets

TL commands are sent from the host to the AFU. An alphabetical list of the TL commands follows; each command is hyperlinked to its specification. In this section, the TL command specifications are in opcode order.

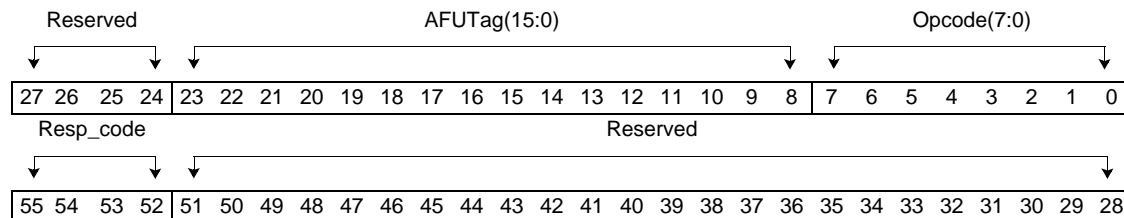
amo_rd	amo_rw	amo_w	config_read
config_write		disable_atc	disable_cache
enable_atc	enable_cache	force_evict	intrp_rdy
kill_xlate	mem_cntl	nop	pad_mem
pr_rd_mem	pr_wr_mem	rd_mem	rd_pf
write_mem	write_mem.be		xlate_done

No operation	nop	'0000 0000'
NA	NA	1



This command has no operands and performs no action. It is discarded at the TLX.

Address translation completed	xlate_done	'0001 1000'
async notification	TL.vc.0	2



The host is sending an asynchronous notification that an address translation requested by a prior TLX command has completed with the indicated response code. The remaining fields of the command identify the prior TLX command.

The TL is required to maintain the order of the matching **read_failed**, **write_failed**, or **touch_resp** response packets carrying the AFUtag and the Resp_code = intrp_pending when loading the VC. That is, the TL shall ensure that the response packets precede the **xlate_done** command in the VC.

The following illustrates how **xlate_done** is used:

1. The device issues a command that requires an address translation that the host is unable to complete.
2. The host responds with

Approved

- a. a **read_failed** response with a Resp_code = xlate_pending. See *Table 2-23 read_failed Resp_code use by TLX command* on page 136 for a list of the commands the device might have issued in step 1.
 - b. a **write_failed** response with a Resp_code = xlate_pending. See *Table 2-25 write_failed Resp_code use by TLX command* on page 143 for a list of the commands the device might have issued in step 1.
 - c. a **touch_resp** response with a Resp_code = xlate_pending. See *Table 2-20 touch_resp Resp_code use by TLX command* on page 124 for a list of the commands the device might have issued in step 1.
3. Once the host has completed the address translation, the host issues **xlate_done** indicating if the device should retry or abort the operation.

The Resp_code field is specified in *Table 2-2*.

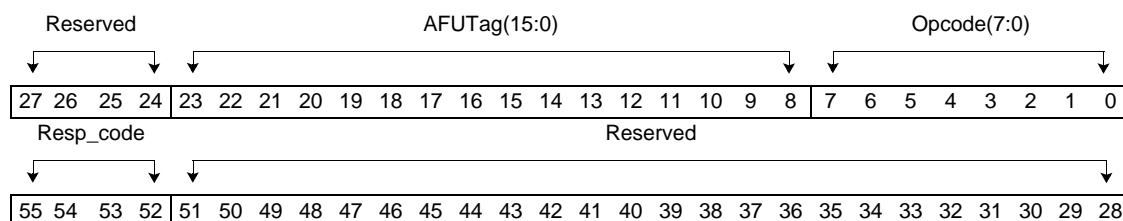
Table 2-2. The Resp_code specification for xlate_done

Resp_code encode	Description
'0000'	Completed. Address translation completed successfully
'0001'	Reserved.
'0010'	Retry request (rty_req). Indicates that the address translation could not be completed at this time. The AFU may make an address translation attempt at a later time. This is a long back-off event.
'0011' - '1110'	Reserved.
'1111'	Translation address error (adr_error). Indicates that the address translation requested resulted in an address translation error.

Note: The errors specified by Resp_code do not include the fatal error conditions described in *Table 7-1* on page 199.

This command is posted.

Interrupt ready	intrp_rdy	'0001 1010'
async notification	TL.vc.0	2



The host is sending an asynchronous status notification for a previously attempted interrupt. The AFU determines its actions based on the Resp_code received. The AFUTag field of the command identifies the prior TLX command.

The TL is required to maintain the order of the matching **intrp_resp** or **wake_host_resp** carrying the AFUTag and the Resp_code = intrp_pending when loading the VC. That is, the TL shall ensure that the response packets precede the **intrp_rdy** command in the VC.

This command is used by two protocol sequences with the following events.

1. Use by **intrp_req**:
 - a. The device issues an **intrp_req** using the *cmd_flag* and *Object_handle* specified in the device's MMIO space.

Approved

- b. The host protocol, using the *Object_handle* for some form of address translation, is unable to complete. The host returns an **intrp_resp** with a *Resp_code* of *intrp_pending* (4).
- c. Once the host has completed the address translation, the host issues **intrp_rdy** indicating if the device should retry or abort the **intrp_req** operation.

2. Used by **wake_host_thread**:

- a. The device issues a **wake_host_thread** using the *cmd_flag* and *Object_handle* specified in the device's configuration space.
- b. The host protocol using the *Object_handle* for some form of address translation, is unable to complete. The host returns an **wake_host_resp** with a *Resp_code* of *intrp_pending* (4).
- c. Once the host has completed the address translation, the host issues **intrp_rdy** indicating if the device should retry or abort the **wake_host_thread** operation.

The *Resp_code* field is specified in *Table 2-3*.

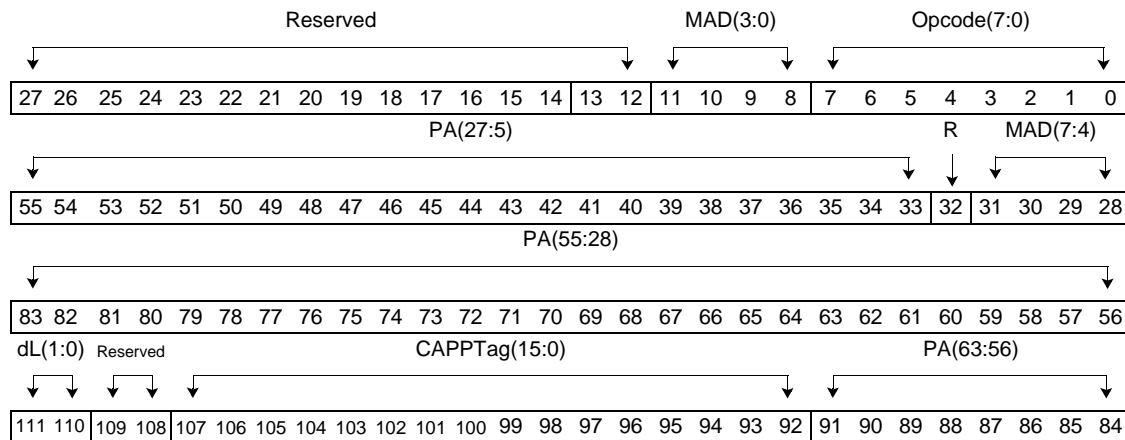
*Table 2-3. The *Resp_code* specification for **intrp_rdy***

Resp_code encode	Description
'0000'	Ready to service the interrupt. The AFU may retry the prior intrp_req , intrp_req.d , or wake_host_thread command.
'0001'	Reserved.
'0010'	Retry request (<i>rtty_req</i>). Indicates that the host is unable to service the interrupt at this time. The AFU may retry the prior interrupt request at a later time as specified by its long back off event timer. This is a long back-off event.
'0011' - '1101'	Reserved.
'1110'	Failed. The host is unable to service the interrupt specified by the prior command. Any future attempt specifying the same interrupt parameters shall fail. <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <p>Engineering Note</p> <p>Note that intrp_req is specified in a way that the <i>cmd_flag</i> and <i>Object_handle</i> are host specific and the values used are found in the device's configuration space. A device correctly using the <i>cmd_flag</i> and <i>Object_handle</i> should not normally see this <i>Resp_code</i>. A malicious device using values other than those provided in its MMIO space may see a failed <i>Resp_code</i>.</p> <p>It is strongly recommended that an implementation provide error collection facilities to indicate the reason for this <i>Resp_code</i>. The specification of the error collection facility should be documented in the host's platform architecture.</p> </div>
'1111'	Reserved.
Note: The errors specified by <i>Resp_code</i> do not include the fatal error conditions described in <i>Table 7-1</i> on page 199.	

This command is posted.

Approved

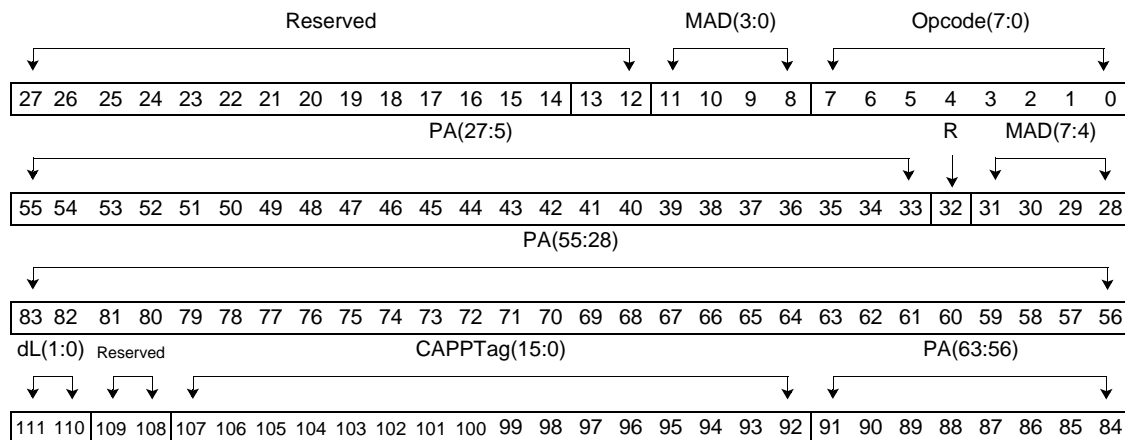
Read memory	rd_mem	'0010 0000'
mem_read	TL.vc.1	4



The host is requesting data to be read from the AFU memory. The starting address specified by the PA, supports a *critical OW request*. The data is a *naturally aligned data block* with a length specified by the dLength field (dL). See the host's platform architecture for the specification of the MAD field.

The response to this command is **mem_rd_response**, **mem_rd_response.ow**, or **mem_rd_fail**. The **mem_rd_fail** response indicates the operation failed. See the response packet for encoding and specifications.

Read Prefetch	rd_pf	'0010 0010'
mem_read	TL.vc.1	4



The host is requesting data to be prefetched by the AFU memory. The starting address, specified by the PA, supports the *critical OW request* format. The data requested is a *naturally aligned data block* with a length specified by the dLength field (dL). See the host's platform architecture for the specification of the MAD field. No data is returned. The AFU_M, when provisioned, may hold data in temporary buffers. The command is intended to provide hints to the AFU_M to access data with long access times that might be required at a later time. The host may or may not request the data at a later time.

Approved

The AFU_M may ignore the command's prefetch hints. The architecture does not specify any errors for this command. Any event occurring in the AFU_M that prohibits the operation from completing is not reported and the command execution is aborted.

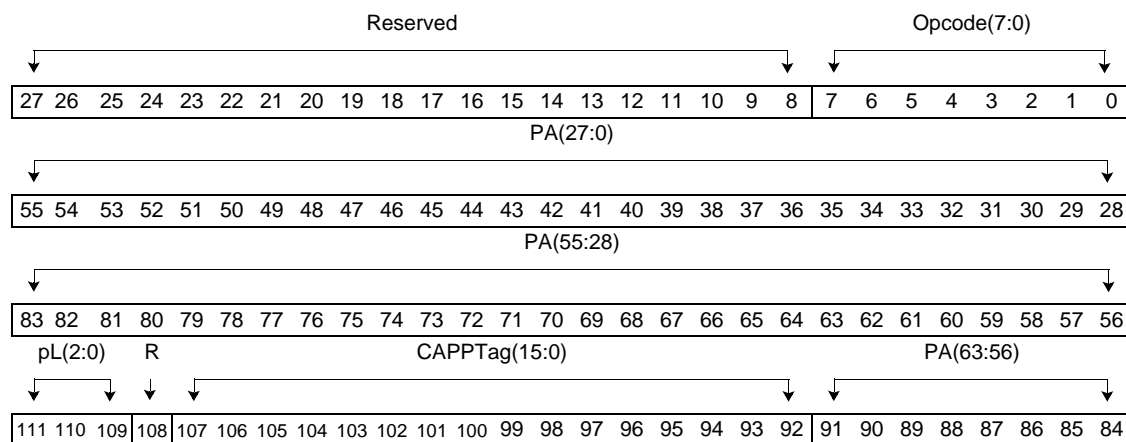
The command is posted.

Developer Note

A *Posted command error* is not specified for this command. If error checking were architected, the architecture would specify the same errors defined for **rd_mem**. Since there is no architected mechanism for a posted command to retry an operation, those types of Resp_code events specified by a response to a **rd_mem** are eliminated. Since data is not returned to the host, dError events are eliminated. This leaves only a Failed error.

Prefetch mechanisms tend to be somewhat inaccurate. Data that is not required by the application might be requested for prefetch and is never demand fetched by the application. The architecture for **rd_pf** pushes any error detection onto a subsequent demand fetch (**rd_mem**) using the same PA.

Partial memory read	pr_rd_mem	'0010 1000'
pr_mem_read	TL.vc.1	4

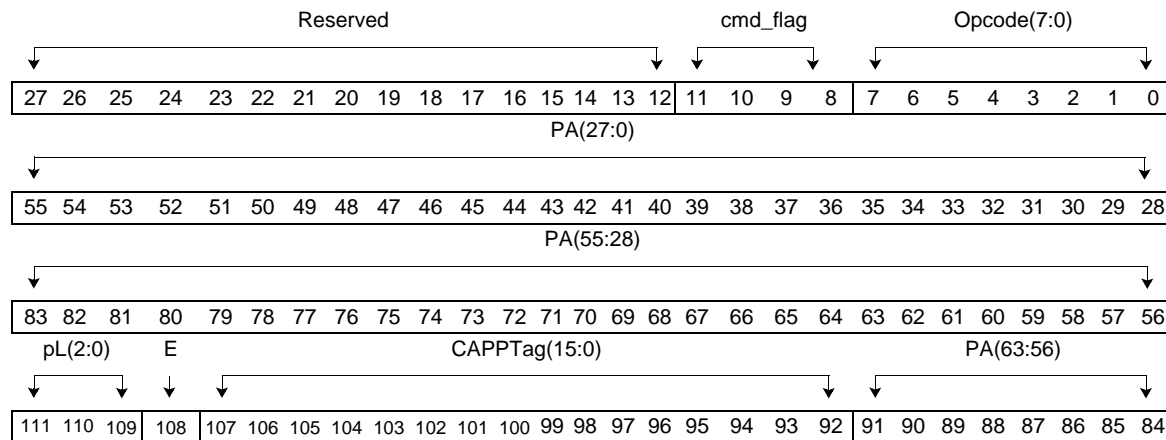


The host is requesting data to be read from the AFU memory. The number of bytes transferred is specified by pLength field(pL), and the starting address shall be naturally aligned based on the number of bytes requested. The pLength field limits the transfer size to 2^n bytes where $n = \{0..5\}$.

The response to this command is **mem_rd_response**, **mem_rd_response.ow**, **mem_rd_response.xw**, or **mem_rd_fail**. When a **mem_rd_response** is received, the data is found in the 64-byte data flit (only one is returned for this command). The data is address aligned as specified in *Section 5.1.3 Data transport, order, and alignment* on page 184. When a **mem_rd_response.ow** is received, the data is found in the 32-byte data field specified by some control flits. The data is address aligned within the data field. The **mem_rd_response.xw** response may be used only when the data length specified by pL is 8 or fewer bytes. That is, when **mem_rd_response.xw** is used to return data, the data length specified by pL shall be 8 or fewer bytes. The **mem_rd_fail** response indicates the operation failed. See the response packet for encoding and specifications.

Approved

AMO Read	amo_rd	'0011 0000'
mem_atomics.r	TL.vc.1	4



The Host is requesting an atomic memory operation specified by the `cmd_flag`. All operands for this request are found in memory as specified by the PA.

Developer note

This command is an in memory operation. Data returned, if any, is stale since the original contents of memory are returned to the host. While the operations specified by this command could be replicated in the host's cache, it would be overly complicated to require the host to maintain a coherent copy in this fashion.

A single response to this command is expected. The response to this command is **mem_rd_response**, **mem_rd_response.ow**, **mem_rd_response.xw**, or **mem_rd_fail**. See the response specification for additional details.

Operation:

The operand length, as specified by the `pLength` field is restricted to 4 and 8 byte operands (`pL` = {'010', '011'} all other values are reserved). There are two operands specified. The first operand "A" is found at the address specified by the command. The second operand "A2" is found at the address specified with an offset specified by the width of the operands and the operation; that is, as specified by `pLength` and by the `cmd_flag`.

- For Fetch and increment bounded, Fetch and increment equal; that is, `cmd_flag` = {'1100', '1101'}, A2 is found at the address specified **plus** the width of the operand.
- For Fetch and decrement bounded; that is, `cmd_flag` = {'1110'}, A2 is found at the address specified **minus** the width of the operand.

The specification of the address shall be naturally aligned and

- shall not target locations at $32n - 2^{\text{bin2dec}(pL)}$, where $n = 1, 2, 3 \dots$ (Fetch and increment bounded, Fetch and increment equal; that is, `cmd_flag` = {'1100', '1101'})
- shall not target locations at $32n$, when $n = 0, 1, 2, 3 \dots$ (Fetch and decrement bounded; that is, `cmd_flag` = {'1110'})

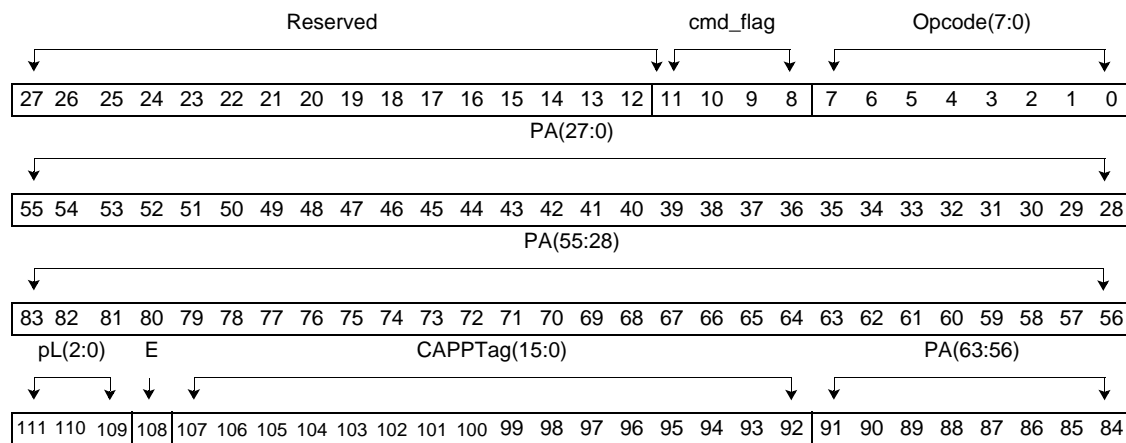
The original value from the memory location specified by the command, or the 4 or 8 byte minimum signed integer value is returned with the response. The operation performed is specified by the `cmd_flag`.

Approved

Table 2-4. The cmd_flag specification for **amo_rd**

cmd_flag	Operation name and description
'0000' through '1010'	Reserved
'1100'	Fetch and increment bounded t ← A; If A != A2 then {A <- A+1; return t} else {return minimum signed integer value}
'1101'	Fetch and increment equal t ← A; If A = A2 then {A <- A+1; return t} else {return minimum signed integer value}
'1110'	Fetch and decrement bounded t ← A; If A != A2 then {A <- A-1; return t} else {return minimum signed integer value}
'1111'	Reserved

AMO read write	amo_rw	'0011 1000'
mem_atomics.rw	TL.vc.1, TL.dcp.1	4



The Host is requesting an atomic memory operation specified by the cmd_flag. For this request, operands are provided with the command and are found in memory as specified by the PA.

This command is specified with immediate data. The data may be sent using data flits, 32-byte data carriers, or 8-byte data carriers as described below. Credits for both the VC and DCP shall be obtained before this command is serviced by the TL.

Developer note

This command is an in memory operation. Data returned, if any, is stale since the original contents of memory are returned to the host. While the operations specified by this command could be replicated in the host's cache, it would be overly complicated to require the host to maintain a coherent copy in this fashion.

Approved

The CAPP TL shall not service this command unless all data specified by pLength is available to be sent. A single response to this command is expected. The response to this command is **mem_rd_response**, **mem_rd_response.ow**, **mem_rd_response.xw**, or a **mem_rd_fail**. See the response specification for additional details.

Operation:

The command's address specified shall be naturally aligned based on the operand length. The operand length is restricted to 4 and 8 byte operands. The specification of pLength shall be specified as {'010', '011'} for all cmd_flag operations with the exception of fetch and swap operations where the cmd_flag is specified as {x'8'...x'A'} and pLength shall be specified as {'110', '111'}. Refer to the specification of *pLength* on page 53.

Operations specified by the cmd_flag use either two or three operands; additional classification of the operands can be found in the description of the operation in *Table 2-5*. The command's address specifies the location of a first operand, "A" which is operated on the second operand, "V", which is provided as the command's write data. V is aligned within

- the 64-byte data flit based on address bits 5:0 specified by the command.
- the 32-byte data field carried in a control flit based on address 4:0 specified by the command.
- an 8 byte data field carried in a control flit based on address 2:0 specified by the command. This shall not be used for fetch and swap operations where the cmd_flag is specified as {x'8'...x'A'}.

A third operand "W" is provided for compare and swap operations. "W" is placed in the same data carrier as "V". "W" is aligned within

- the 64-byte data flit based on the following equation:

$$\text{alignment}(5:0) \leftarrow \text{PA}(5:4) \parallel (\text{PA}(3:0) + '1000')$$
Any carryout from bit 3 is ignored.
- the 32-byte data field carried in a control flit based on the following equation:

$$\text{alignment}(4:0) \leftarrow \text{PA}(4) \parallel (\text{PA}(3:0) + '1000')$$

Any carryout from bit 3 is ignored.

The original value from the memory location specified by the command is returned with a **mem_rd_response**, **mem_rd_response.ow**, or **mem_rd_response.xw**.

The endianness of the operands is specified by the E bit. The value of E may not affect the result of the operation specified by the cmd_flag. This is noted in the operation description found in *Table 2-5*.

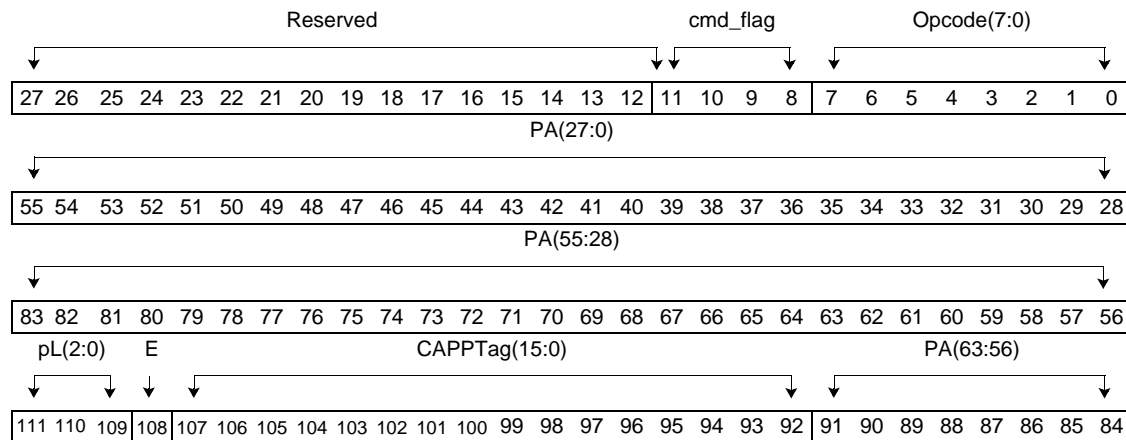
Approved

Table 2-5. The cmd_flag specification for **amo_rw**

cmd_flag	Operation name and description
'0000'	Fetch and Add Operands are unsigned integers. $t \leftarrow A; A \leftarrow A + V; \text{return } t$ Overflow conditions are not reported.
'0001'	Fetch and XOR Operands are bit-vectors. E has no affect on the operation. $t \leftarrow A; A \leftarrow V \oplus A; \text{return } t$
'0010'	Fetch and OR Operands are bit-vectors. E has no affect on the operation. $t \leftarrow A; A \leftarrow V \vee A; \text{return } t$
'0011'	Fetch and AND Operands are bit-vectors. E has no affect on the operation. $t \leftarrow A; A \leftarrow V \wedge A; \text{return } t$
'0100'	Fetch and maximum unsigned. Operands are unsigned integers. $t \leftarrow A; A \leftarrow \text{Max}(A, V); \text{return } t$ A is not modified when A is greater than or equal to V.
'0101'	Fetch and maximum signed Operands are signed 2's complement integers. $t \leftarrow A; A \leftarrow \text{Max}(A, V); \text{return } t$ A is not modified when A is greater than or equal to V.
'0110'	Fetch and minimum unsigned Operands are unsigned integers. $t \leftarrow A; A \leftarrow \text{Min}(A, V); \text{return } t$ A is not modified when A is less than or equal to V.
'0111'	Fetch and minimum signed Operands are signed 2's complement integers. $t \leftarrow A; A \leftarrow \text{Min}(A, V); \text{return } t$ A is not modified when A is less than or equal to V.
'1000'	Fetch and swap Operands are bit-vectors. E has no affect on the operation. V is not used. $t \leftarrow A; A \leftarrow W; \text{return } t$
'1001'	Fetch and swap equal Operands are bit-vectors. E has no affect on the operation. $t \leftarrow A; \text{When } V = A, \text{ then } A \leftarrow W; \text{return } t$
'1010'	Fetch and swap not equal Operands are bit-vectors. E has no affect on the operation. $t \leftarrow A; \text{when } V \neq A, \text{ then } A \leftarrow W; \text{return } t$
'1011' through '1111'	Reserved

Approved

AMO write	amo_w	'0100 0000'
mem_atomics.w	TL.vc.1, TL.dcp.1	4



The Host is requesting an atomic in-memory operation specified by the **cmd_flag**. For this request, operands are provided with the command and are found in memory as specified by the PA.

This command is specified with immediate data; Credits for both the VC and DCP shall be obtained before this command is serviced by the TL.

The CAPP TL shall not service this command unless all data specified by pLength is available to be sent.

Developer note

This command is an in memory operation. While the operations specified by this command could be replicated in the host's cache, it would be overly complicated to require the host to maintain a coherent copy in this fashion.

mem_wr_response and **mem_wr_fail** are the responses to this command which indicates the status of the write to memory operation. **mem_wr_response** indicates a successful completion of the operation. **mem_wr_fail** indicates the operation failed. See the response packet for encoding and specifications.

Operation:

The command's address specified shall be naturally aligned based on the operand length. The operand length, as specified by the pLength field shall be restricted to 4 and 8 byte operands. That is, the pLength shall be $pL = \{ '010', '011' \}$, with all other values specified as reserved.

The number of operands specified by this command is determined by an examination of the **cmd_flag**. Two or three operands may be specified as illustrated in *Table 2-6*. The operands are designated as "A", "A2" and "V". The command's address specifies the location of each operand as follows:

- "A" shall be located in memory at the address specified by the PA and shall be naturally aligned. When the **cmd_flag** indicates the use of "A2", the address of A is further constrained and shall not target locations at $32n - 2^{\text{bin2dec}(pL)}$, where $n = 1, 2, 3, \dots$
- "A2" is located in memory at the address specified by the PA plus an offset specified by the width of the operands.

Approved

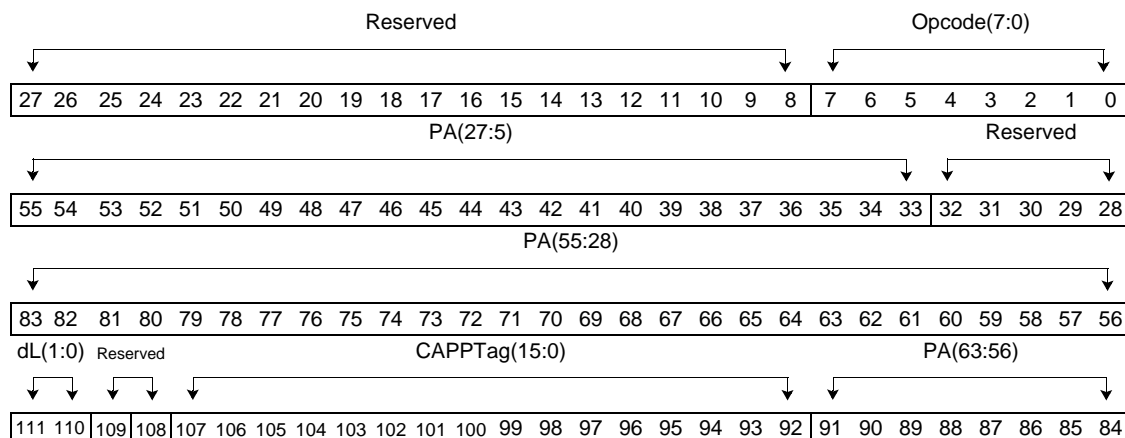
- “V” is provided as the command’s write data. V is aligned within
 - a 64 byte data flit based on PA bits 5:0 specified by the command, or
 - a 32-byte data field found within a control flit based on PA bits 4:0 specified by the command, or
 - an 8-byte data field found within a control flit based on PA bits 2:0 specified by the command.

Table 2-6. The *cmd_flag* specification for **amo_w**

cmd_flag	Operation name and description
'0000'	Store and Add Operands are unsigned integers. $A \leftarrow A + V$; Overflow conditions are not reported.
'0001'	Store and XOR Operands are bit-vectors. E has no affect on the operation. $A \leftarrow V \oplus A$
'0010'	Store and OR Operands are bit-vectors. E has no affect on the operation. $A \leftarrow V \vee A$
'0011'	Store and AND Operands are bit-vectors. E has no affect on the operation. $A \leftarrow V \wedge A$
'0100'	Store and maximum unsigned. Operands are unsigned integers. $A \leftarrow \text{Max}(A, V)$ A is unchanged when A is greater than or equal to V.
'0101'	Store and maximum signed Operands are signed 2's complement integers. $A \leftarrow \text{Max}(A, V)$ A is unchanged when A is greater than or equal to V.
'0110'	Store and minimum unsigned Operands are unsigned integers. $A \leftarrow \text{Min}(A, V)$ A is unchanged when A is less than or equal to V.
'0111'	Store and minimum signed Operands are signed 2's complement integers. $A \leftarrow \text{Min}(A, V)$ A is unchanged when A is less than or equal to V.
'1000 through '1011'	Reserved
'1100'	Store and compare twin Operands are bit-vectors. E has no affect on the operation. When $A=A2$, then $(A \leftarrow V, A2 \leftarrow V)$
'1101' through '1111'	Reserved

Approved

Pad memory	pad_mem	'1000 0000'
mem_write	TL.vc.1	4



The host is issuing a request to load memory using a data pattern set during the configuration of the device. The configuration of the data pattern in the device is device implementation dependent and is beyond the scope of this architecture. The pattern is used to fill a 32-, 64-, 128- or 256-byte naturally aligned block of data in the AFU memory. The starting address is specified by the PA field and shall be naturally aligned based on the length of the data as specified by the dLength (dL) field¹⁵.

Engineering Note

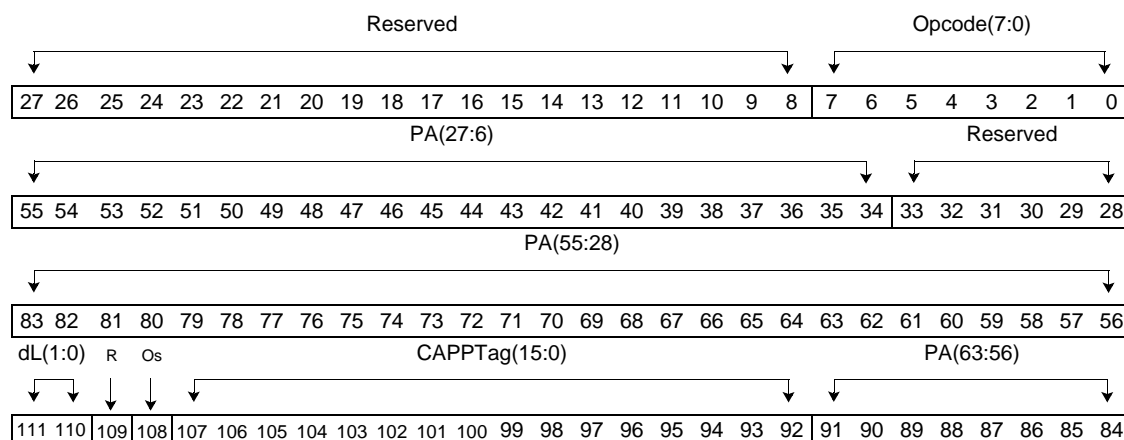
The pattern is expected to be found through the device's configuration space and may be located in the device's MMIO space. The width of the pattern shall be a power of 2 number of bits. The pattern is applied starting at offset 0 (bit 0, byte 0) of the data block. When the pattern is smaller than the operation width specified by the dLength field, the pattern is repeated. When the pattern is larger than the operation width specified by the dLength field, the pattern starting at offset 0 is applied and as space permits offset 1 is applied and so on.

The **mem_wr_response** and **mem_wr_fail** responses to this command indicate the status of the pad memory operation. The **mem_wr_response** indicates a successful completion of the operation. The **mem_wr_fail** response indicates that the operation failed. See the response packet description for encoding and specifications.

¹⁵. A dLength value of '00' specifies a 32-byte data block.

Approved

Write memory	write_mem	'1000 0001'
mem_write	TL.vc.1, TL.dcp.1	4



The host is writing a 64-, 128-, or 256-byte block of data to the AFU memory. The starting address is specified by the PA field and shall be naturally aligned based on the length of the data as specified by the dLength (dL) field.

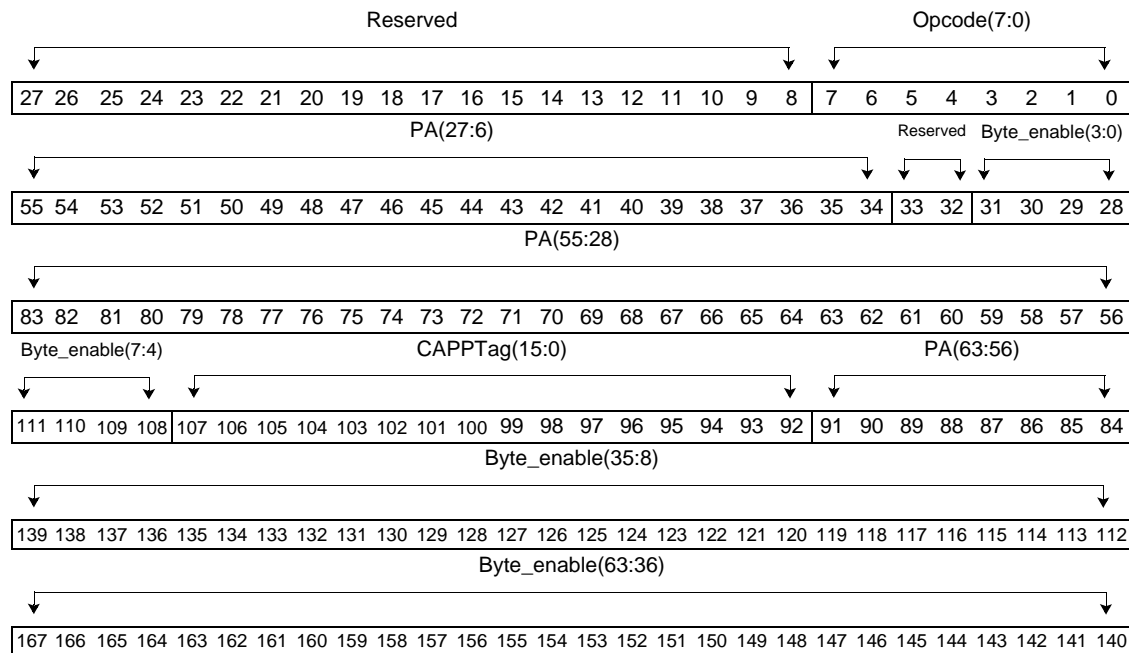
When the data length field specifies 128 or 256 bytes, the Os bit controls ordering between the data segments.

This command is specified with immediate data. The data may be transferred using data flits or 32-byte data fields found within a control flit. Credits for both the VC and DCP shall be obtained before this command is serviced by the TL.

The **mem_wr_response** and **mem_wr_fail** responses to this command indicate the status of the write to memory operation. The **mem_wr_response** indicates a successful completion of the operation. The **mem_wr_fail** response indicates that the operation failed. See the response packet description for encoding and specifications.

Approved

Byte enable memory write	write_mem.be	'1000 0010'
pr_mem_write	TL.vc.1, TL.dcp.1	6



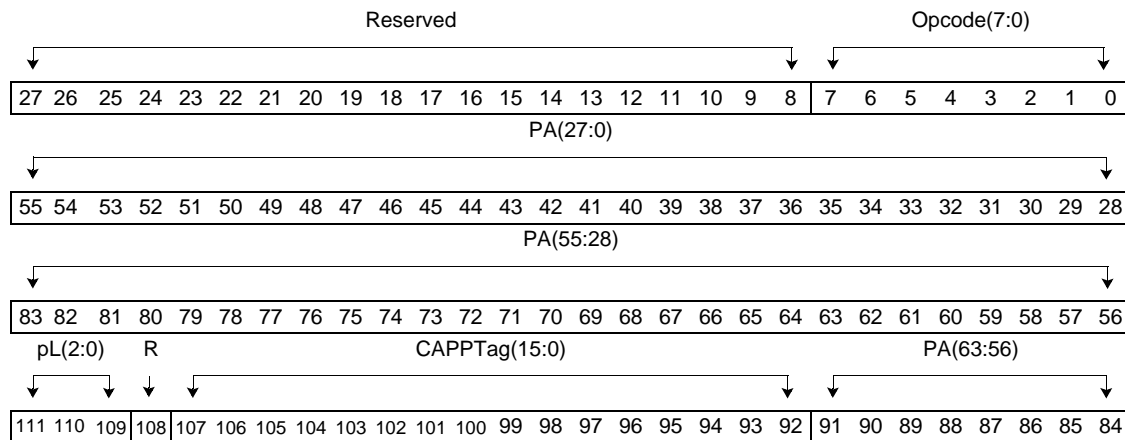
The host is writing a 64-byte data block to the AFU memory using a 64-bit byte-enable field. Each bit corresponds to one byte of the 64-byte aligned data block specified by the PA, where bit 0 of the BE determines if byte 0 of the data is written. When BE(n) is set to 1, byte n is written where n={0..63}.

This command is specified with immediate data. The data shall be sent using a single 64-byte data flit. Credits for both the VC and DCP shall be obtained before this command is serviced by the TL.

The **mem_wr_response** and **mem_wr_fail** responses to this command indicate the status of the write to memory operation. The **mem_wr_response** indicates a successful completion of the operation. The **mem_wr_fail** response indicates that the operation failed. See the response packet for encoding and specifications.

Approved

Partial cache line memory write	pr_wr_mem	'1000 0110'
pr_mem_write	TL.vc.1, TL.dcp.1	4

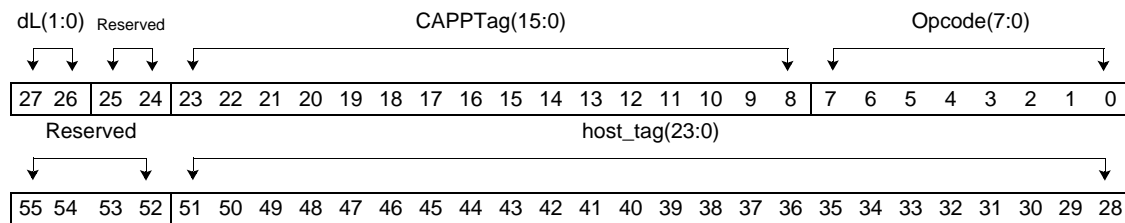


The host is writing the data to the AFU memory. The starting address is specified by the PA and shall be naturally aligned based on the length of the data as specified by the pLength (pL) field. The combination of the address and the pLength shall not cross a 64-byte address boundary.

This command is specified with immediate data. The data may be sent in a data flit, or it may be sent in a 32-byte data carrier. When the pL field indicates a length of 8 bytes or less, the data may be sent in an 8-byte data carrier. The data is address aligned as specified in *Section 5.1.3 Data transport, order, and alignment* on page 184. Credits for both the VC and DCP shall be obtained before this command is serviced by the TL.

The **mem_wr_response** and **mem_wr_fail** responses to this command indicate the status of the write to memory operation. The **mem_wr_response** indicates a successful completion of the operation. The **mem_wr_fail** response indicates that the operation failed. See the response packet for encoding and specifications.

Force eviction	force_evict	'1101 0000'
cache management	TL.vc.0	2



The host is requesting that the AFU_{C2} evict one, two, or four 64 byte cache block segments held in its cache.

The AFU shall evict the cache block segments specified by the *host_tag* using one or more **castout** or **castout.push** TLX commands specifying the *cache_state* as Invalid (I) within the time period specified by the host's platform architecture. This action shall invalidate all synonyms held by the AFU_{C2} pointed to by the *host_tags* specified by the command. The AFU_{C2} shall invalidate the *host_tag* entries specified by the command when completing the command.

Approved

The dLength field indicates the number of cache block segments by specifying the length of the block that the host is requesting the AFU cache to evict. A difference between the host's cache line size and the AFU's cache line size may result in one or more **castout*** commands. All **castout*** commands for the specified cache block segments must be received at the host prior to the expiration of the time out period. See the descriptions of *host_tag*, *host_tag arithmetic on page 20* and *cl_rd_resp*.

Engineering Note

A host implementation may choose to take down the link or otherwise disable the AFU when the AFU does not respond to the **force_evict** command within the host specified time period.

See *TL response timer expired on page 206* for the specification of the error class and error signature.

Developer note

The intent of this command is provide the host with a mechanism to force the AFU's cache to evict the line specified by the *host_tag*.

The AFU is expected to respond with either a **castout** or **castout.push** command indicating a transition to an I state within a host specified time-out period. **force_evict** is a posted command.

This is a posted command.

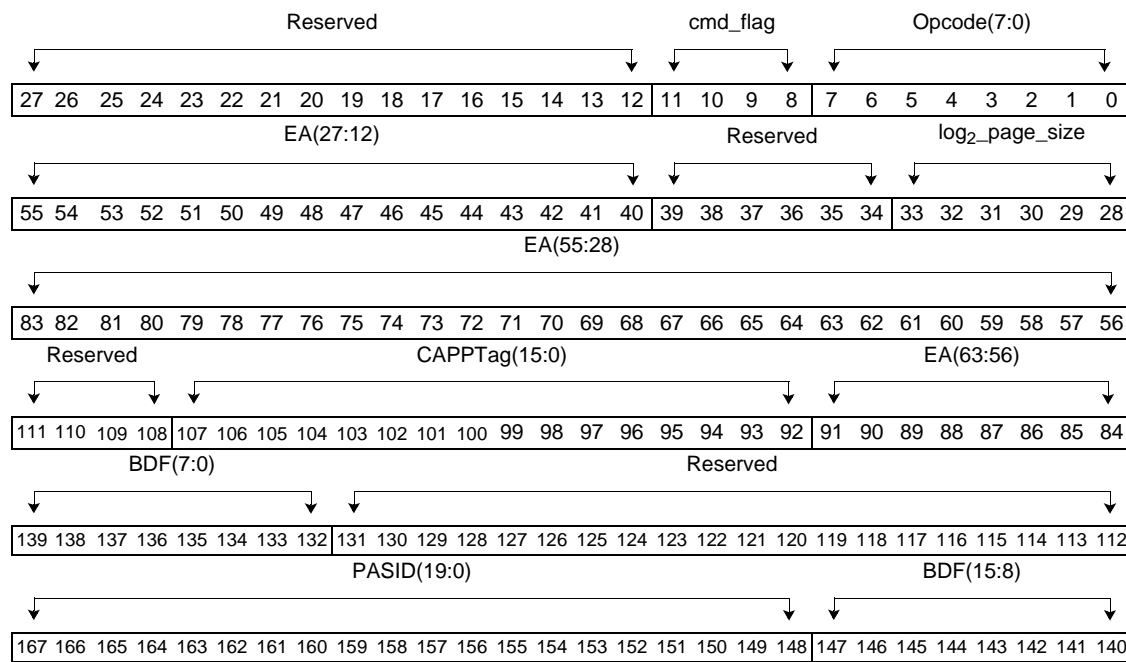
force_evict AFU collision rules

1. A **force_evict**(*host_tag*) from the host hits a running TLX **read_me/read_mes/read_s** (acTag, EA) in the AFU.

The AFU may detect the collision by converting *host_tag* to EA across all address contexts. The AFU shall service the **force_evict** regardless of the collision. The host determines the order and services the **read_me/read_mes/read_s** command.

Approved

Kill address translation entry	kill_xlate	'1101 0010'
address translation management	TL.vc.2	6



The host is requesting that the AFU kill one or more address translation ATC entries corresponding to the page specified by the EA, log₂_page_size, address context as specified by the PASID and BDF, and cmd_flag directive that is passed in this command. The address translation was obtained by the AFU using **xlate_touch** with a command flag indicating that a translated address (TA) was required. See *Section 1.8.2.2 Host initiated AFU ATC entry invalidation* on page 47.

The cmd_flag is specified in *Table 2-7*.

Table 2-7. The cmd_flag specification for **kill_xlate**

cmd_flag	Operation description
'0000'	The host is requesting that the AFU kill ATC entries corresponding to the page specified by the EA, log ₂ _page_size and address context as specified by the PASID and BDF fields. Support of this code point by the TLX is optional. When not supported, the TLX shall treat this code point as either '0001' or '1111'.
'0001'	The host is requesting that the AFU kill ATC entries corresponding to the address context specified by the PASID and BDF. The EA and log ₂ _page_size fields are reserved. Support of this code point by the TLX is optional. When not supported the TLX shall treat this code point as x'1111'.
'0010' - '1110'	Reserved
'1111'	The host is requesting that the AFU kill all ATC entries. The EA, log ₂ _page_size, PASID and BDF fields are reserved. All TLX shall support this code point.

The architectural model requires that the AFU shall invalidate one or more entries of the AFU ATC corresponding to the command's operands. An implementation may choose to provide an alternative to an AFU ATC. Since the architecture assumes the existence of these ATC based on the AFU's use of **xlate_touch**, alternative implementations shall be constructed in such a manner that the differences are not externally observable.

Approved

An AFU_{C2} holding lines in its cache using this address translation shall no longer modify or reference those lines until it obtains the address translation again. This applies to all the lines contained within the page specified by the address translation.

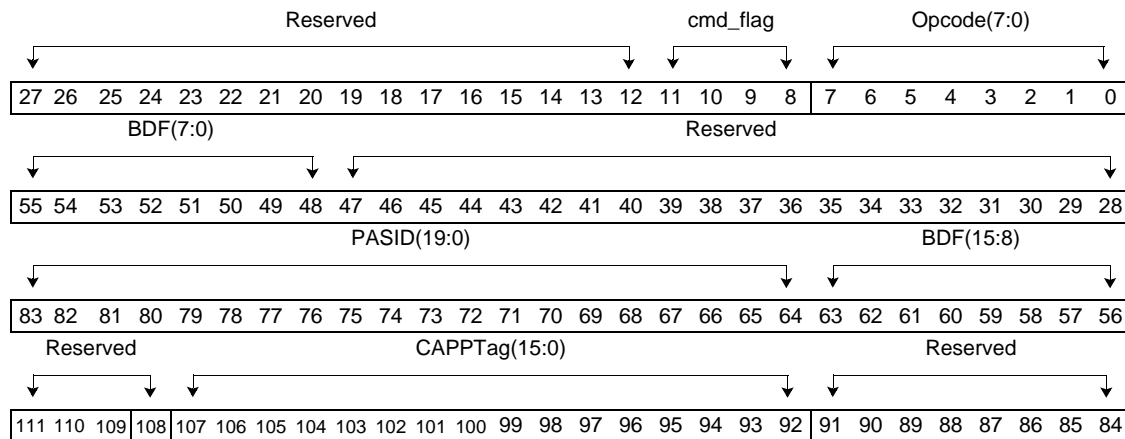
An AFU_{C1} shall stop using the TA in all TLX dot-t commands until it obtains the address translation again.

The EA shall be naturally aligned based on the page size specified by the log₂_page_size field when the cmd_flag is '0000'.

Once the AFU has taken actions to ensure that the address translation is no longer used as described above, the AFU may release one or more TA to the host using one or more TLX **xlata_release** commands. The AFU shall ensure that **xlata_release** is added to the TLX.vc.3 after all dot-t commands using the translation have been added to the VC. The AFU shall respond to the completion of all actions to kill the ATC entries corresponding to the command's operands by issuing a **kill_xlata_done**. The AFU shall ensure the **kill_xlata_done** is added to the TLX.vc.3 after all dot-t commands using the translations killed have been added to the VC.

See Figure A-8. *xlata_touch TLX and TL interaction on page 229.*

Disable AFU _{C2} cache	disable_cache	'1101 0100'
address translation management	TL.vc.2	1



The host is requesting that the AFU disable its processing element's access to the AFU_{C2} cache. Access to the cache and cache miss actions are requested to be suspended based on the address context criteria specified in the command flag (cmd_flag) field. It is expected that this will suspend most processing element operations using the address context specified.

The AFU_{C2} shall continue to serviced all TL commands and responses including those that may cause updates to the AFU ATC and AFU_{C2} cache.

The cmd_flag is specified in Table 2-8.

Approved

Table 2-8. The cmd_flag specification for **disable_cache**

cmd_flag	Operation name and description
'0000'	(disable_all) Disable access to the cache and miss actions regardless of address context. The BDF and PASID fields are undefined.
'0001'	(disable_by_BDF). Disable access to the cache and miss actions based on address context using the BDF specified in the command. The BDF field is valid. The PASID field is undefined.
'0010'	Reserved.
'0011'	(disable_by_ac). Disable access to the cache and miss actions based on the full address context using the BDF and PASID specified in the command. The BDF and PASID field are valid.
'0100' - '1111'	Reserved.

The AFU responds to this command with **cache_disabled**. The host shall wait for the AFU response before issuing a second **disable_cache**.

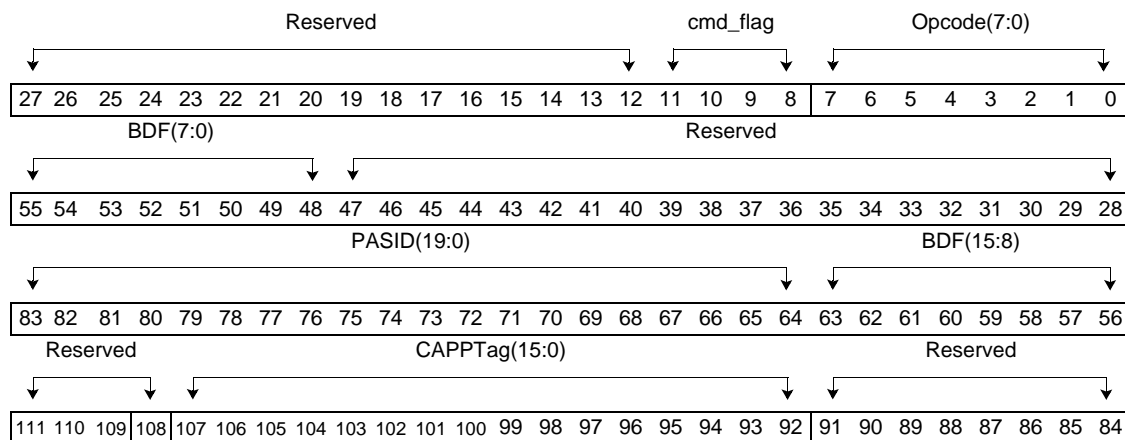
disable_cache actions are accumulative. If a **disable_cache** command finds that the specified cache and address context combination is already disabled, **cache_disabled** is returned and no error is indicated.

Support for cmd_flag x'1' and x'3' which specifies address context granularity, is optional¹⁶ and when not supported these values shall be treated as x'0' (disable_all). That is, an implementation shall support cmd_flag values of x'0' and may support x'1' and x'3'.

Developer note

This enables a host to disable the AFU_{C2} cache. Not all host protocols require the use of this command.

Enable AFU _{C2} cache	enable_cache	'1101 0101'
address translation management	TL.vc.2	1



The host is requesting that the AFU enable its processing element's access to the AFU_{C2} cache based on the address context criteria specified in the command flag (cmd_flag) field. This command is expected to be used only when a **disable_cache** command has been previously issued.

¹⁶. Device support is found the device's configuration space.

Approved

The AFU responds to this command with **cache_enabled**.

The cmd_flag is specified in Table 2-9.

Table 2-9. The cmd_flag specification for **enable_cache**

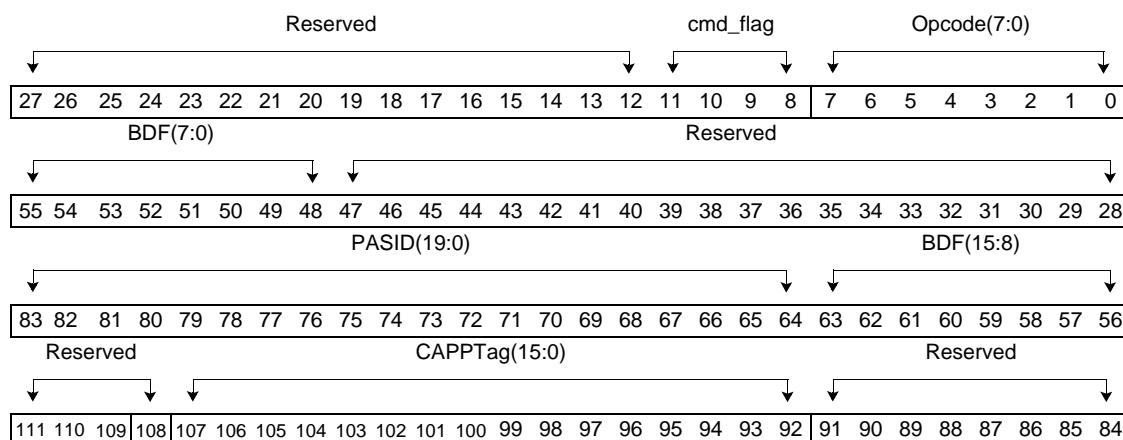
cmd_flag	Operation name and description(
'0000'	(enable_all) Enable access to the cache and miss actions regardless of address context. The BDF and PASID fields are undefined.
'0001'	(enable_by_BDF). Enable access to the cache and miss actions based on address context using the BDF specified in the command. The BDF field is valid. The PASID field is undefined.
'0010'	Reserved.
'0011'	(enable_by_ac). Enable access to the cache and miss actions based on the full address context using the BDF and PASID specified in the command. The BDF and PASID field are valid.
'0100' - '1111'	Reserved.

The AFU responds to this command with **cache_enabled**. The host shall wait for the AFU response before issuing a second **enable_cache**.

enable_cache actions are accumulative. If an **enable_cache** command finds that the specified cache and address context combination is already enabled, **cache_enabled** is returned and no error is indicated.

Support for cmd_flag x'1' and x'3' which specifies address context granularity, is optional¹⁷ and when not supported these values shall be treated as x'0' (enable_all). That is, an implementation shall support cmd_flag values of x'0' and may support x'1' and x'3'.

Disable AFU _{C2} cache	disable_atc	'1101 0110'
address translation management	TL.vc.2	1



The host is requesting that the AFU disable its processing element's access to the AFU's ATC. Access to the ATC and ATC miss actions are requested to be suspended based on the address context criteria specified in the command flag (cmd_flag) field. It is expected that this will suspend most processing element operations using the address context specified.

17. Device support is found the device's configuration space.

Approved

The AFU shall continue to serviced all TL commands and responses including those that may cause updates to the AFU ATC.

The cmd_flag is specified in *Table 2-10*.

*Table 2-10. The cmd_flag specification for **disable_atc***

cmd_flag	Operation name and description
'0000'	(disable_all) Disable access to the AFU's ATC and miss actions regardless of address context. The BDF and PASID fields are undefined.
'0001'	(disable_by_BDF). Disable access to the AFU's ATC and miss actions based on address context using the BDF specified in the command. The BDF field is valid. The PASID field is undefined.
'0010'	Reserved.
'0011'	(disable_by_ac). Disable access to the AFU's ATC and miss actions based on the full address context using the BDF and PASID specified in the command. The BDF and PASID field are valid.
'0100' - '1111'	Reserved.

The AFU responds to this command with **atc_disabled**. The host shall wait for the AFU response before issuing a second **disable_atc**.

disable_atc actions are accumulative. If a **disable_atc** command finds that the specified ATC and address context combination is already disabled, **atc_disabled** is returned and no error is indicated.

Support for cmd_flag x'1' and x'3', which specifies address context granularity, is optional¹⁸ and when not supported these values shall be treated as x'0' (disable_all). That is, an implementation shall support cmd_flag values of x'0' and may support x'1' and x'3'.

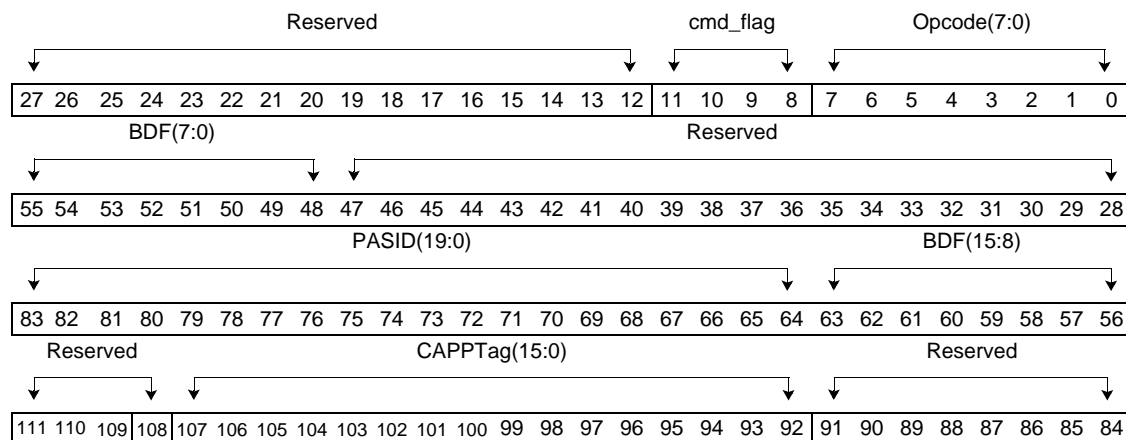
Developer note

This enables a host to disable the AFU ATC. Not all host protocols require the use of this command.

¹⁸. Device support is found the device's configuration space.

Approved

Enable AFU _{C2} cache	enable_atc	'1101 0111'
address translation management	TL.vc.2	1



The host is requesting that the AFU enable its processing element's access to the AFU's ATC based on the address context criteria specified in the command flag (cmd_flag) field. This command is expected to be used only when a **disable_atc** command has been previously issued.

The AFU responds to this command with **atc_enabled**.

The cmd_flag is specified in *Table 2-11*.

*Table 2-11. The cmd_flag specification for **enable_atc***

cmd_flag	Operation name and description(
'0000'	(enable_all) Enable access to the AFU's ATC and miss actions regardless of address context. The BDF and PASID fields are undefined.
'0001'	(enable_by_BDF). Enable access to the AFU's ATC and miss actions based on address context using the BDF specified in the command. The BDF field is valid. The PASID field is undefined.
'0010'	Reserved.
'0011'	(enable_by_ac). Enable access to the AFU's ATC and miss actions based on the full address context using the BDF and PASID specified in the command. The BDF and PASID field are valid.
'0100' - '1111'	Reserved.

The AFU responds to this command with **atc_enabled**. The host shall wait for the AFU response before issuing a second **enable_atc**.

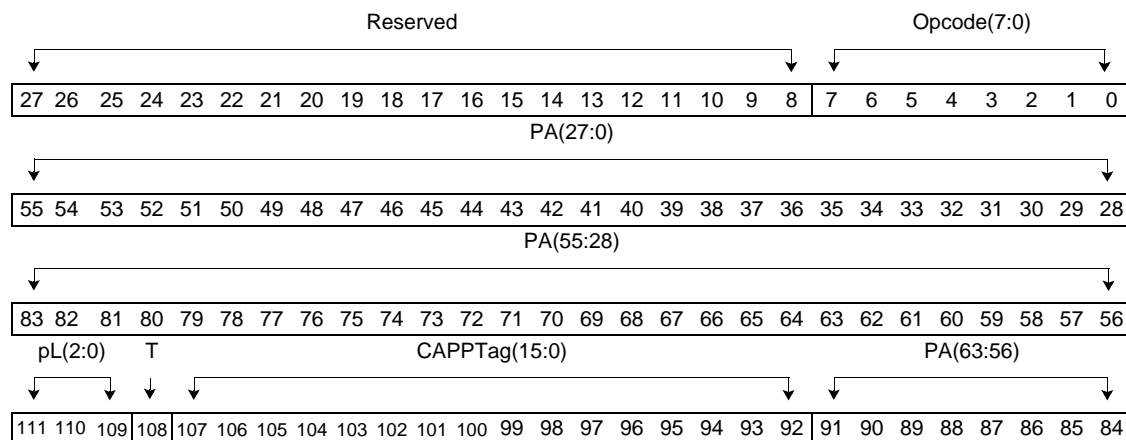
enable_atc actions are accumulative. If an **enable_atc** command finds that the specified AFU's ATC and address context combination is already enabled, **atc_enabled** is returned and no error is indicated.

Support for cmd_flag x'1' and x'3', which specifies address context granularity, is optional¹⁹ and when not supported these values shall be treated as x'0' (enable_all). That is, an implementation shall support cmd_flag values of x'0' and may support x'1' and x'3'.

¹⁹. Device support is found the device's configuration space.

Approved

Configuration read	config_read	'1110 0000'
configuration	TL.vc.1	4



The host is issuing a read to the AFU's configuration address space. The number of bytes transferred is specified by pLength (pL) field. The starting address shall be naturally aligned based on the number of bytes requested. For this command, the pLength value is limited to a transfer size of 1, 2, or 4 bytes. That is, the specification of pLength shall limit the transfer size to 2^n bytes where $n = \{0..2\}$.

The PA field is defined as follows:

PA bits	Description
63:32	Reserved. Shall be set to ³² 0.
31:24	Bus number (7:0).
23:19	Device number (4:0).
18:16	Function number (2:0).
15:12	Reserved. Shall be set to ⁴ 0.
11:2	Register number.
1:0	Byte offset within the register.

The T field, defined in *Table 2-1* on page 48, specifies the configuration type of the command.

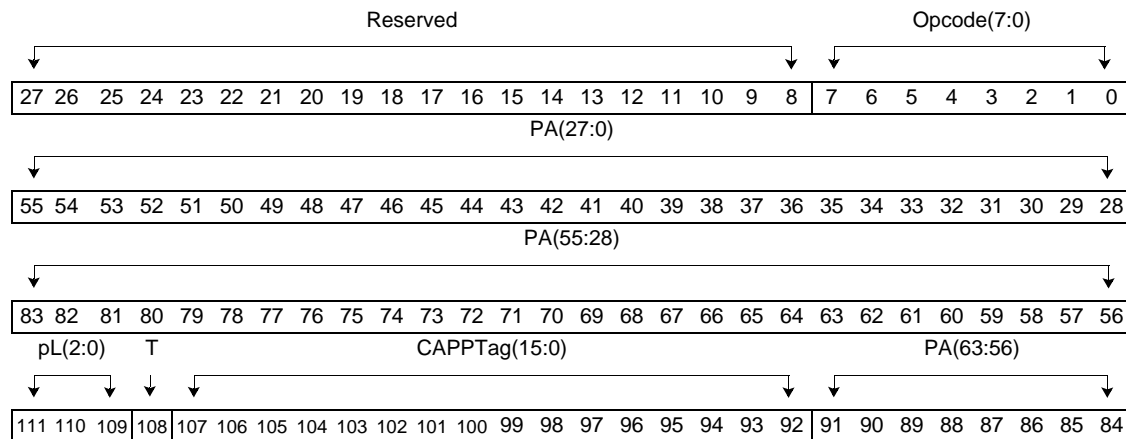
The response to this command is **mem_rd_response**, **mem_rd_response.ow**, **mem_rd_response.xw**, or **mem_rd_fail**. When a **mem_rd_response** is received, the data is found in the 64-byte data flit (only one is returned for this command). The data is address aligned as specified in *Section 5.1.3 Data transport, order, and alignment* on page 184. When a **mem_rd_response.ow** is received, the data is found in the 32-byte data field found in a control flit (only one is returned for this command). The data is address aligned. When a **mem_rd_response.xw** is received, the data is found in the 8-byte data field found in a control flit and is address aligned.

Neither metadata or extended-metadata are specified for this command. If the template used when returning the response specifies metadata or extended-metadata for the data carrier used, the metadata and extended-metadata is discarded by the host, an error shall not be reported, and the operation completes successfully.

Approved

The **mem_rd_fail** response indicates the operation failed. If the device or function number are not recognized by the AFU, the operation shall fail with a Resp_code = Failed. See the response packet for encoding and specifications.

Configuration write	config_write	'1110 0001'
configuration	TL.vc.1, TL.dcp.1	4



The host is issuing a write to the AFU's configuration address space. The number of bytes transferred is specified by pLength (pL) field. The starting address shall be naturally aligned based on the number of bytes requested. For this command, pLength is limited to a transfer size of 1, 2, or 4 bytes. That is, the specification of pLength shall limit the transfer size to 2^n bytes where $n = \{0..2\}$.

The PA field is defined as follows:

PA bits	Description
63:32	Reserved. Shall be set to ³² 0.
31:24	Bus number (7:0).
23:19	Device number (4:0).
18:16	Function number (2:0).
15:12	Reserved. Shall be set to ⁴ 0.
11:2	Register number.
1:0	Byte offset within the register.

The T field, defined in *Table 2-1* on page 48, specifies the configuration type of the command. When T is set to 0, the AFU learns its bus number located in the PA field. The device and function number are assigned by the attached OpenCAPI device and are not modified by any configuration actions. If the device or function numbers are not recognized, the operation shall fail and the data is discarded. The failure shall be reported using a TLX **mem_wr_fail** response with a Resp_code= Failed.

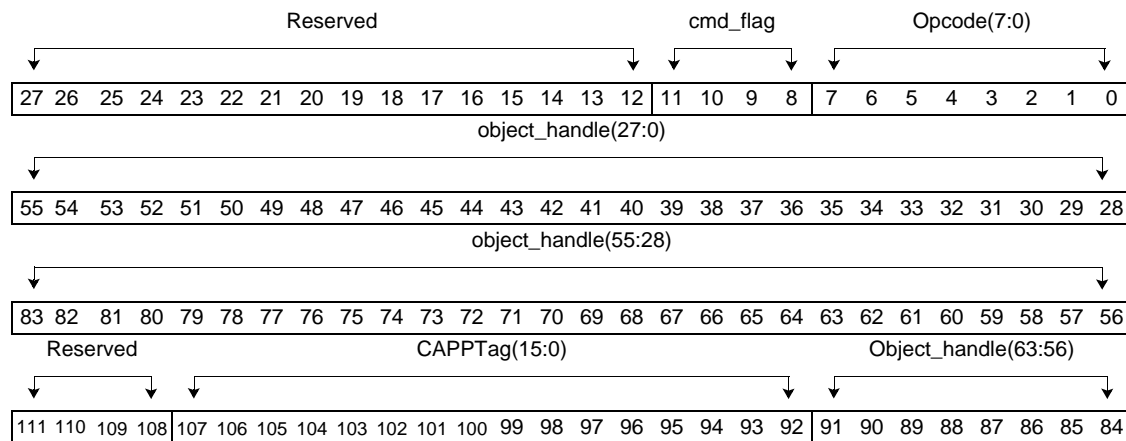
This command is specified with immediate data. The data is address aligned as specified in *Section 5.1.3 Data transport, order, and alignment* on page 184. The data may be sent in a data flit, or it may be sent in a 32- or 8-byte data carrier. Credits for both the VC and DCP shall be obtained before this command is serviced by the TL.

Approved

Neither metadata or extended-metadata are specified for this command. If the template used when issuing this command specifies metadata or extended-metadata for the data carrier used, the metadata shall be discarded by the AFU, an error shall not be reported, and the operation completes successfully.

The **mem_wr_response** and **mem_wr_fail** responses to this command indicate the status of the write to memory operation. The **mem_wr_response** indicates a successful completion of the operation. The **mem_wr_fail** response indicates that the operation failed. See the response packet for encoding and specifications.

Memory control	mem_cntl	'1110 1111'
message	TL.vc.0	4



This command is used for device defined functions specified by the device manufacturer. The **cmd_flag** and **object_handle** are defined by the device manufacturer. Any errors in the host's specification of these fields results in the operation failing.

The response to this command is a **mem_cntl_done**.

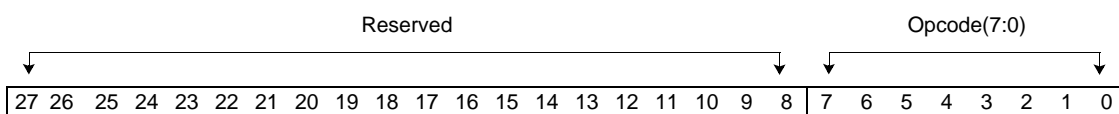
Approved

2.3 TLX AP command packets

TLX commands are sent from the AFU to the host. An alphabetical list of the TLX commands follows; each command is hyperlinked to its specification. In this section, the TLX command specifications are in opcode order.

amo_rd	amo_rd.n	amo_rd.t	amo_rd.t.s
amo_rw	amo_rw.n	amo_rw.t	amo_rw.t.s
amo_w	amo_w.n	amo_w.t.p	amo_w.t.p.s
assign_actag		castout	castout.push
dma_pr_w	dma_pr_w.n	dma_pr_w.t.p	dma_pr_w.t.p.s
dma_w	dma_w.be	dma_w.be.n	dma_w.be.t.p
dma_w.be.t.p.s	dma_w.n	dma_w.t.p	dma_w.t.p.s
intrp_req	intrp_req.d		nop
pr_rd_wnitc	pr_rd_wnitc.n	pr_rd_wnitc.t	pr_rd_wnitc.t.s
rd_wnitc	rd_wnitc.n	rd_wnitc.t	rd_wnitc.t.s
read_me	read_me.t	read_mes	read_mes.t
read_s	read_s.t	sync	synonym_done
upgrade_state	upgrade_state.t	wake_host_thread	xlate_release
xlate_touch	xlate_touch.n		

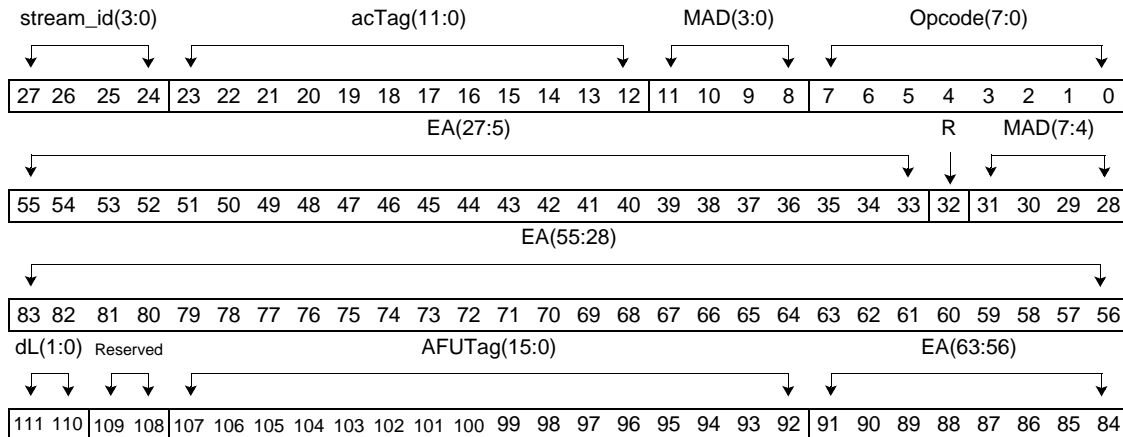
No operation	nop	'0000 0000'
NA	NA	1



This command has no operands and performs no action. It is discarded at the TL.

Approved

Read with no intent to cache	rd_wnltc rd_wnltc.n	'0001 0000' '0001 0100'
dma_read	TLX.vc.3	4



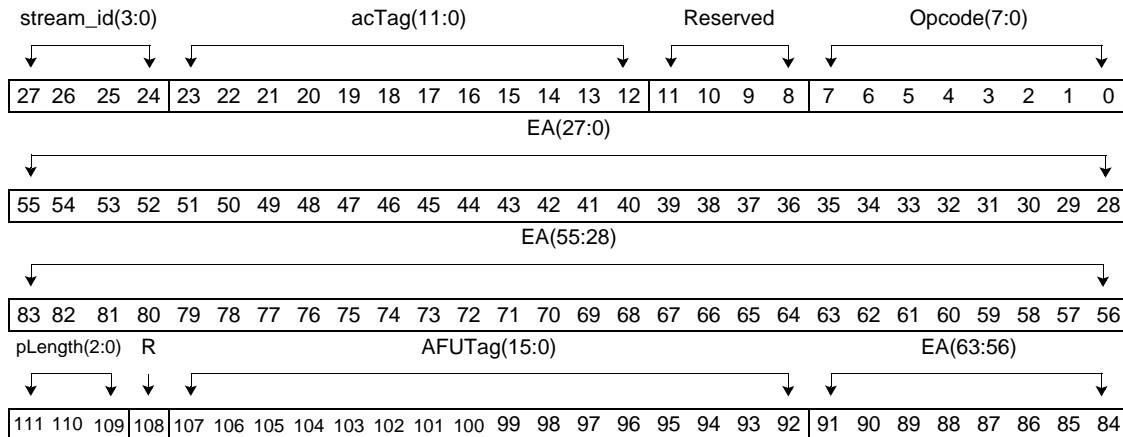
The AFU is requesting to read data with no intent to cache. The starting address specified by the EA supports a *critical OW request*. The data is a *naturally aligned data block* with a length specified by the dLength field (dL). See the host's platform architecture for the specification of the *MAD* field.

- The dot-n form indicates that the results of the address translation may not be installed into the host's ATC as part of ATC miss handling.

The response to this command is **read_response**, **read_response.ow**, or **read_failed**. Multiple responses to a single **rd_wnltc** may occur. The **read_failed** response indicates the operation failed. See the response packet for additional details.

Approved

Partial read with no intent to cache	pr_rd_wnrtc pr_rd_wnrtc.n	'0001 0010' '0001 0110'
pr_dma_read	TLX.vc.3	4



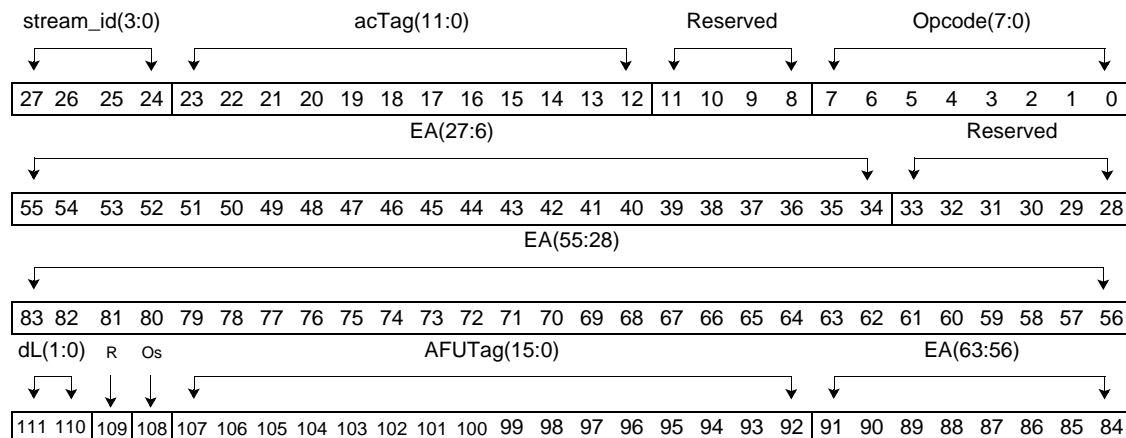
The AFU is requesting to read a partial cache line of data with no intent to cache at the address specified by the EA. The starting address shall be naturally aligned based on the length of the data specified by the pLength field. The pLength restricts this command to lengths of powers of 2 ranging from 1 to 32 bytes.

- The dot-n form indicates that the results of the address translation may not be installed into the host's ATC as part of ATC miss handling.

The response to this command is **read_response**, **read_response.ow**, or **read_failed**. When a **read_response** is received, the data is address aligned (address bits 5:0) in the 64-byte data flit (only one is returned for this command). When a **read_response.ow** is received, the data is address aligned (address bits(4:0)) in the 32-byte data carrier (only one is returned for this command). The **read_failed** response indicates the operation failed. See the response packet for additional details.

Approved

DMA Write	dma_w dma_w.n	'0010 0000' '0010 0100'
dma_write	TLX.vc.3, TLX.dcp.3	4



The AFU is requesting to write data at the address specified by the EA. The starting address is specified by the EA and shall be naturally aligned based on the length of the data as specified by the dLength field.

When the data length field specifies 128 or 256 bytes, the Os bit controls ordering between the data segments.

Developer note

The line may be held in the AFU's EA L1 cache and not be aware of it if the address used by the **dma_w** turns out to be a synonym.

- The host's coherency protocol could require that the proxy cache be checked after address translation of the EA presented with the **dma_w** to determine if the line (RA determined by translation) is already in the proxy cache. If the line is already in the proxy cache, a synonym has been detected.

If a synonym is detected, the host's coherency protocol might require that the AFU push the line (issue a **force_evict** command).

This command is specified with immediate data. The data may be sent in a data flit, or it may be sent in multiple 32-byte data carriers. Credits for both the VC and DCP shall be obtained before this command is serviced by the TLX.

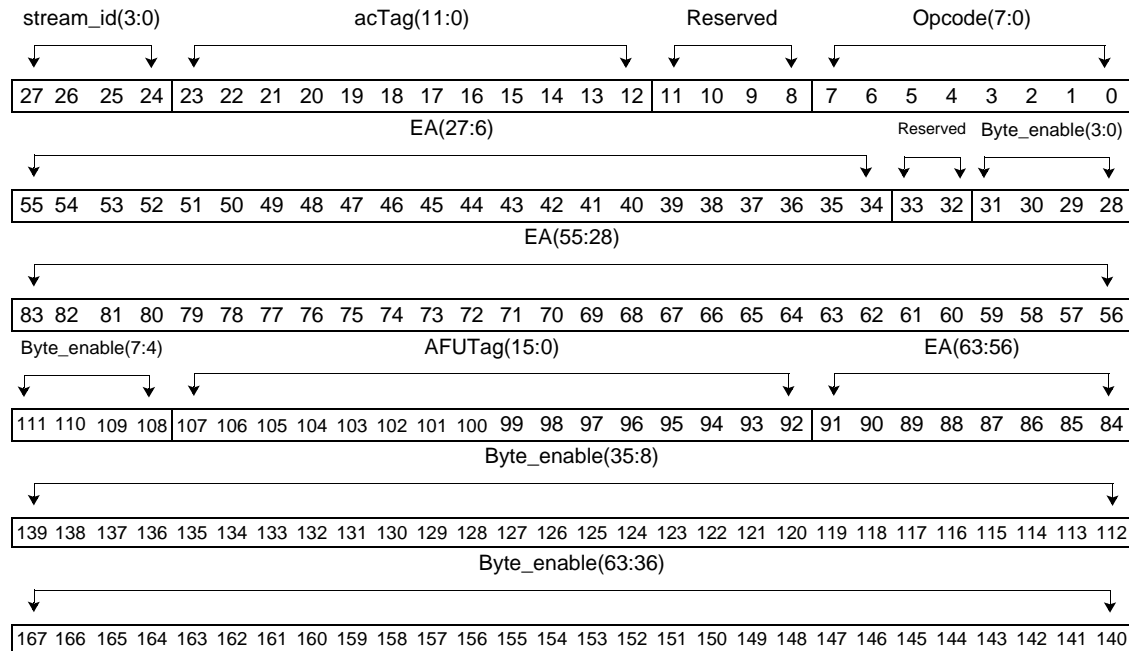
The AFU TLX shall not service this command unless all data specified by dLength is available to be sent.

- The dot-n form indicates that the results of the address translation may not be installed into the host's ATC as part of ATC miss handling.

The host shall respond with either a **write_response** or a **write_failed** response packet. The **write_failed** response indicates that the operation failed. See the response packet description for additional details.

Approved

Byte enable DMA Write	dma_w.be dma_w.be.n	'0010 1000' '0010 1100'
pr_dma_write	TLX.vc.3, TLX.dcp.3	6



The AFU is writing data at the address specified by the EA using a 64-bit byte-enable field. Each bit corresponds to one byte of the 64-byte aligned data block specified by the EA, where bit 0 of the BE determines if byte 0 of the data is written. When BE(n) is set to 1, byte n is written where $n=\{0..63\}$.

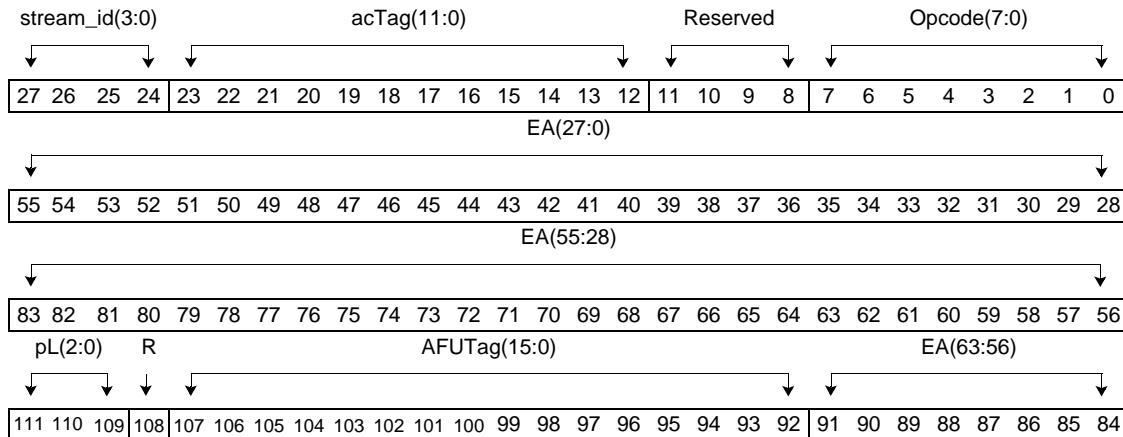
This command is specified with immediate data. The data shall be sent in a 64-byte data flit. Credits for both the VC and DCP shall be obtained before this command is serviced by the TLX.

- The dot-n form indicates that the results of the address translation may not be installed into the host's ATC as part of ATC miss handling.

The host shall respond with either a **write_response** or a **write_failed** response packet. The **write_failed** response indicates that the operation failed. See the response packet description for additional details.

Approved

DMA parital write	dma_pr_w dma_pr_w.n	'0011 0000' '0011 0100'
pr_dma_write	TLX.vc.3, TLX.dcp.3	4



The AFU is requesting to write data starting at the address specified by the EA. The starting address shall be naturally aligned based on the length of the data as specified by the pLength (pL) field. The pLength restricts this command to lengths of powers of 2 ranging from 1 to 32 bytes. The combination of the EA and the pLength shall not cross a 64-byte address boundary.

Only a single data carrier is associated with this command.

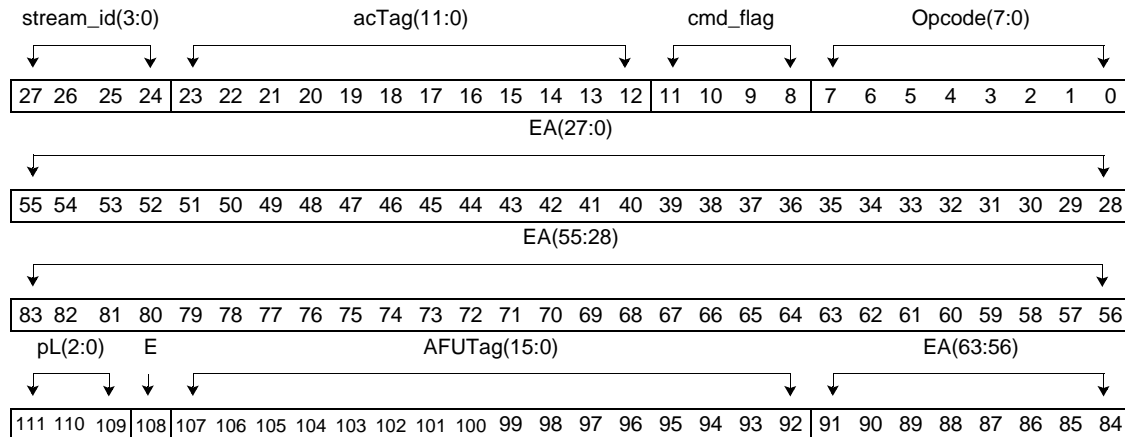
This command is specified with immediate data. The data is address aligned as specified in *Section 5.1.3 Data transport, order, and alignment* on page 184. The data may be sent in a data flit, or it may be sent in a 32-byte data carrier. When the length specified by pL is 8 or less bytes, the data may be sent in an 8-byte data carrier. Credits for both the VC and DCP shall be obtained before this command is serviced by the TLX.

The AFU TLX shall not service this command unless all data specified by pLength is available to be sent.

The dot-n form indicates that the results of the address translation may not be installed into the host's ATC as part of ATC miss handling. The host shall respond with either a **write_response** or a **write_failed** response packet. The **write_failed** response indicates that the operation failed. See the response packet description for additional details.

Approved

AMO read	amo_rd amo_rd.n	'0011 1000' '0011 1100'
atomics.r	TLX.vc.3	4



The AFU is requesting an atomic memory operation specified by the **cmd_flag**. All operands for this request are found in memory as specified by the **EA**.

- The dot-n form indicates that the results of the address translation may not be installed into the host's ATC as part of ATC miss handling.

There shall be a single response to this command. The response to this command shall be one of the following TL responses: **read_response**, **read_response.ow**, **read_response.xw**, or **read_failed**. The **read_failed** response indicates that the operation failed. See the response packet specification for additional details.

Operation:

The operand length, as specified by the **pLength** (**pL**) field is restricted to 4- and 8-byte operands. That is, the **pLength** shall be specified as {'010', '011'}; all other values are reserved. Two signed integer operands are specified. The first operand "A" is found at the address specified by the command. The second operand "A2" is found at the address specified with an offset specified by the width of the operands and the operation; that is, as specified by **pLength** and by the command flag.

- For Fetch and increment bounded and Fetch and increment equal (that is, **cmd_flag** = {'1100', '1101'}), A2 is found at the address specified *plus* the width of the operand.
- For Fetch and decrement bounded (that is, **cmd_flag** = {'1110'}), A2 is found at the address specified *minus* the width of the operand.

The specification of the address is constrained to be naturally aligned. In addition:

- It cannot target locations at $32n - 2^{\text{bin2dec}(\text{pL})}$, where $n = 1, 2, 3...$ (Fetch and increment bounded and Fetch and increment equal; that is, **cmd_flag** = {'1100', '1101'}).
- It cannot target locations at $32n$, where $n = 0, 1, 2, 3...$ (Fetch and decrement bounded; that is, **cmd_flag** = {'1110'}).

The original value from the memory location specified by the command, or the 4- or 8-byte minimum signed integer value, is returned with **read_response**.

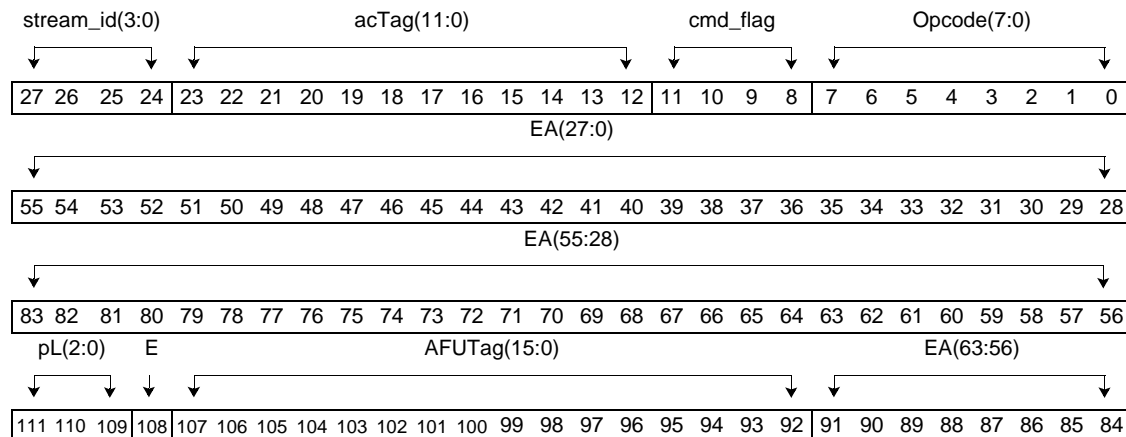
Approved

The operation performed is specified by the cmd_flag. The endianness of the operands is specified by the E bit.

Table 2-12. The cmd_flag specification for **amo_rd**

cmd_flag	Operation name and description
'0000' through '1010'	Reserved
'1100'	Fetch and increment bounded t ← A; If A != A2 then {A ← A+1; return t} else {return minimum signed integer value}
'1101'	Fetch and increment equal t ← A; If A = A2 then {A ← A+1; return t} else {return minimum signed integer value}
'1110'	Fetch and decrement bounded t ← A; If A != A2 then {A ← A-1; return t} else {return minimum signed integer value}
'1111'	Reserved

AMO read write	amo_rw amo_rw.n	'0100 0000' '0100 0100'
atomics.rw	TLX.vc.3, TLX.dcp.3	4



The AFU is requesting an atomic memory operation specified by the cmd_flag. For this request, operands are provided with the command and are found in memory as specified by the EA.

- The dot-n form indicates that the results of the address translation may not be installed into the host's ATC as part of ATC miss handling.

This command is specified with immediate data. The data may be sent using data flits, 32-byte data carriers, or 8-byte data carriers as described in the following operation description. Use of 8-byte data carriers is restricted as specified below. Credits for both the VC and DCP shall be obtained before this command is serviced by the TLX.

Approved

The AFU TLX shall not service this command unless all data specified by pLength is available to be sent.

There shall be a single response to this command. The response to this command shall be one of the following TL responses: **read_response**, **read_response.ow**, **read_response.xw**, or **read_failed**. The **read_failed** response indicates that the operation failed. See the response packet specification for additional details.

Operation:

The command address specified shall be naturally aligned based on the operand length. The operand length is restricted to 4- and 8-byte operands. The pLength (pL) shall be specified as {'010', '011'} for all cmd_flag operations with the exception of fetch and swap operations where the cmd_flag is specified as {x'8'...x'A'} and pLength shall be specified as {'110', '111'}. Refer to the specification of *pLength* on page 53.

Operations specified by the cmd_flag use either two or three operands; additional classification of the operands can be found in the description of the operation in *Table 2-13*. The command's address specifies the location of a first operand, "A".

Operand A is operated on by the second operand, "V", which is provided as the command's write data. Operand V is aligned within one of the following:

- a 64-byte data flit based on address bits 5:0 specified by the command.
- a 32-byte data field carried in a control flit based on address bits 4:0 specified by the command.
- an 8-byte data field carried in a control flit based on address 2:0 specified by the command. This option shall not be used for fetch and swap operations where the cmd_flag is specified as {x'8'...x'A'}.

A third operand "W" is provided for fetch and swap operations. Operand W is placed in the same data carrier as operand V. Operand W shall be aligned within one of the following:

- the 64-byte data flit based on the following equation:

$$\text{alignment (5:0)} \leftarrow \text{EA(5:4)} \parallel (\text{EA(3:0)} + '1000')$$
Any carryout from bit 3 is ignored.
- the 32-byte data field carried in a control flit based on the following equation:

$$\text{alignment (4:0)} \leftarrow \text{EA(4)} \parallel (\text{EA(3:0)} + '1000')$$
Any carryout from bit 3 is ignored.

The original value from the memory location specified by the command shall be returned with a **read_response**, **read_response.ow**, or **read_response.xw**.

The endianness of the operands is specified by the E bit. The value of E might not affect the result of the operation specified by the cmd_flag. This is noted in the operation description found in *Table 2-13*.

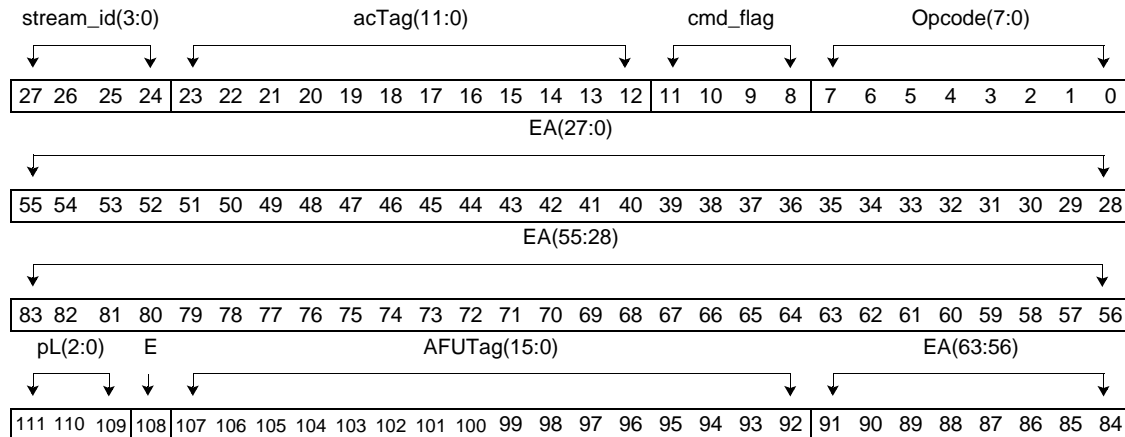
Approved

Table 2-13. The cmd_flag specification for **amo_rw**

cmd_flag	Operation name and description
'0000'	Fetch and Add Operands are unsigned integers. $t \leftarrow A; A \leftarrow A + V; \text{return } t$ Overflow conditions are not reported.
'0001'	Fetch and XOR Operands are bit vectors. E has no effect on the operation. $t \leftarrow A; A \leftarrow V \oplus A; \text{return } t$
'0010'	Fetch and OR Operands are bit vectors. E has no effect on the operation. $t \leftarrow A; A \leftarrow V \vee A; \text{return } t$
'0011'	Fetch and AND Operands are bit vectors. E has no effect on the operation. $t \leftarrow A; A \leftarrow V \wedge A; \text{return } t$
'0100'	Fetch and maximum unsigned Operands are unsigned integers. $t \leftarrow A; A \leftarrow \text{Max}(A, V); \text{return } t$ A is unchanged when A is greater than or equal to V.
'0101'	Fetch and maximum signed Operands are signed two's complement integers. $t \leftarrow A; A \leftarrow \text{Max}(A, V); \text{return } t$ A is unchanged when A is greater than or equal to V.
'0110'	Fetch and minimum unsigned Operands are unsigned integers. $t \leftarrow A; A \leftarrow \text{Min}(A, V); \text{return } t$ A is unchanged when A is less than or equal to V.
'0111'	Fetch and minimum signed Operands are signed two's complement integers. $t \leftarrow A; A \leftarrow \text{Min}(A, V); \text{return } t$ A is unchanged when A is less than or equal to V.
'1000'	Fetch and swap Operands are bit vectors. E has no effect on the operation. V is not used. $t \leftarrow A; A \leftarrow W; \text{return } t$
'1001'	Fetch and swap equal Operands are bit vectors. E has no effect on the operation. $t \leftarrow A; \text{When } V = A, \text{ then } A \leftarrow W; \text{return } t$
'1010'	Fetch and swap not equal Operands are bit vectors. E has no effect on the operation. $t \leftarrow A; \text{when } V \neq A, \text{ then } A \leftarrow W; \text{return } t$
'1011' through '1111'	Reserved

Approved

AMO write	amo_w amo_w.n	'0100 1000' '0100 1100'
atomics.w	TLX.vc.3, TLX.dcp.3	4



The AFU is requesting an atomic memory operation specified by the **cmd_flag**. For this request, operands are provided with the command and are found in memory as specified by the EA.

This command is specified with immediate data. Credits for both the VC and DCP shall be obtained before this command is serviced by the TLX.

The AFU TLX shall not service this command unless all data specified by pLength (pL) is available to be sent.

- The dot-n form indicates that the results of the address translation may not be installed into the host's ATC as part of ATC miss handling.

There shall be a single response to this command. The host shall respond with either a **write_response** or a **write_failed** response packet. The **write_failed** response indicates that the operation failed. See the response packet description for additional details.

Operation:

The command's address shall be naturally aligned based on the operand length. The operand length, as specified by the pLength (pL) field, is restricted to 4- and 8-byte operands. That is, the pLength shall be {'010', '011'}. All other values of pLength are reserved.

The number of operands specified by this command is determined by an examination of the **cmd_flag**. Two or three operands may be specified as shown in *Table 2-14* on page 93. The operands are designated as "A," "A2," and "V". The command's address specifies the location of each operand as follows:

- Operand A is located in memory at the address specified by the EA and shall be naturally aligned. When the **cmd_flag** indicates the use of operand A2, the address of operand A is further constrained and shall not target locations at $32n \cdot 2^{\text{bin2dec}(\text{'pL'})}$, where $n = 1, 2, 3 \dots$
- Operand A2 is located in memory at the address specified by the EA plus an offset specified by the width of the operands.

Approved

- Operand V is provided as the command's write data. Operand V shall be aligned within one of the following:
 - a 64-byte data flit based on address bits 5:0 specified by the command
 - a 32-byte data field carried in a control flit based on address bits 4:0 specified by the command
 - an 8-byte data field carried in a control flit based on address 2:0 specified by the command

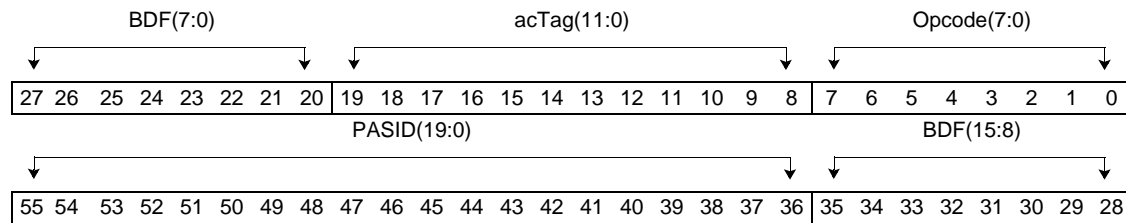
The endianness of the operands is specified by the E bit. The value of E might not affect the result of the operation specified by the cmd_flag. This is noted in the operation description found in *Table 2-14*.

*Table 2-14. The cmd_flag specification for **amo_w***

cmd_flag	Operation name and description
'0000'	Store and Add Operands are unsigned integers. $A \leftarrow A + V$ Overflow conditions are not reported.
'0001'	Store and XOR Operands are bit vectors. E has no effect on the operation. $A \leftarrow V \oplus A$
'0010'	Store and OR Operands are bit vectors. E has no effect on the operation. $A \leftarrow V \vee A$
'0011'	Store and AND Operands are bit vectors. E has no effect on the operation. $A \leftarrow V \wedge A$
'0100'	Store and maximum unsigned. Operands are unsigned integers. $A \leftarrow \text{Max}(A, V)$ A is unmodified when A is greater than or equal to V.
'0101'	Store and maximum signed Operands are signed two's complement integers. $A \leftarrow \text{Max}(A, V)$ A is unmodified when A is greater than or equal to V.
'0110'	Store and minimum unsigned Operands are unsigned integers. $A \leftarrow \text{Min}(A, V)$ A is unmodified when A is less than or equal to V.
'0111'	Store and minimum signed Operands are signed two's complement integers. $A \leftarrow \text{Min}(A, V)$ A is unmodified when A is less than or equal to V.
'1000 through '1011'	Reserved.
'1100'	Store and compare twin Operands are bit vectors. E has no effect on the operation. When $A = A2$, then $(A \leftarrow V, A2 \leftarrow V)$
'1101' through '1111'	Reserved.

Approved

acTag Assignment	assign_actag	'0101 0000'
acTag mgmt	TLX.vc.3	2

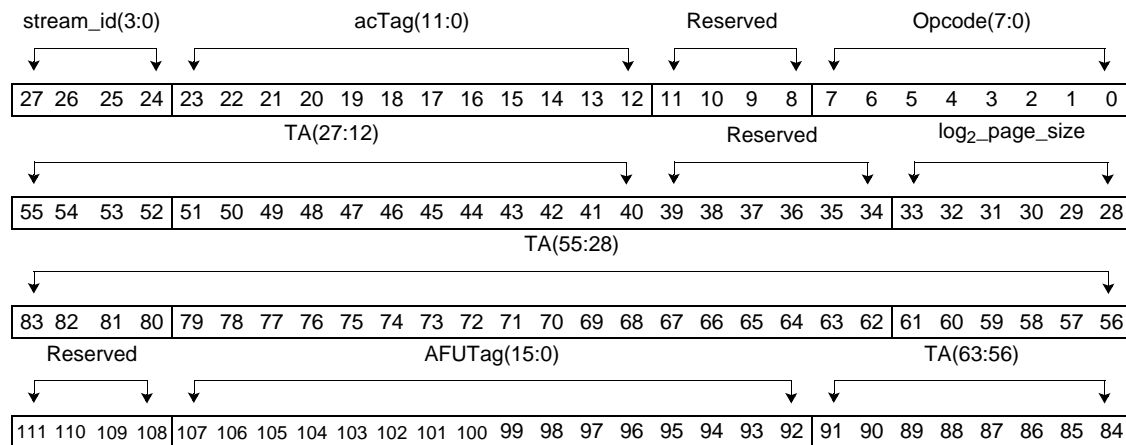


This command is used by the attached OpenCAPI device to assign an acTag value to a BDF and PASID combination. The OpenCAPI device uses this command to manage the contents of the acTag table. See *Section 4 The acTag table* on page 180 for the use of this command, the acTag table, and the management requirements placed on the OpenCAPI device.

This command is serviced when it reaches the head of the VC in the TL. It is not added to a *service queue*.

This command is posted. No response is sent for this command.

Translation release	xlata_release	'0101 0001'
address translation management	TLX.vc.3	4



This command is used to return an address translation to the host. All commands using this translation shall precede the **xlata_release** command in the TLX.vc.3 virtual channel. This command is used by the AFU to manage its ATC as described in *Section 1.8.2 Translated addresses, AFU ATC, and dot-t commands* on page 45.

The acTag must point to the BDF and PASID used when the address translation was obtained. The stream_id is used to ensure that the TLX commands using the translation are in the same service queue as the **xlata_release**.

This command is assigned to TLX.vc.3 which is the same VC used by all TLX read and write commands. This command requires the AFU push all commands using this translated address into the VC before removing the AFU ATC entry. That is, all commands using this translation must be dispatched to the host before the

Approved

AFU ATC entry has been invalidated. To accomplish this the command shall be placed into the TLX.vc.3 virtual channel after all commands using the address translation have been committed to the TLX.vc.3 virtual channel. The host shall ensure that all uses of the address translation have completed before invalidating the AFU's use of the translated address. That is, all non-posted commands, or any clean up that is required by the host implementation with respect to the translated address being used by the AFU, shall complete before the translated address becomes unrecognizable for-use-by-the-AFU by the host.

Developer note

The AFU needs to maintain the ATC entry until it has pushed all commands operations using it into TLX.vc.3. This includes non-posted operations such as **read_me.t**. Updates to the cache are allowed since the responses use **host_tags** and the host has ensured that the correct address translation is used when obtaining the data.

Note that once the cache has been updated and the ATC entry has been invalidated, the cache line is no longer accessible by the AFU processor element since the address translation has been invalidated.

TLX commands using TLX.vc.3 specify a **stream_id**. When the host's *service queue* hash includes the **stream_id**, the commands using the ATC entry may be in different host *service queues*. Placing **xlate_release** in only one *service queue* does not assure that all uses of the address translation have been serviced before the **xlate_release** has been serviced. When the AFU is using multiple streams, the AFU shall either:

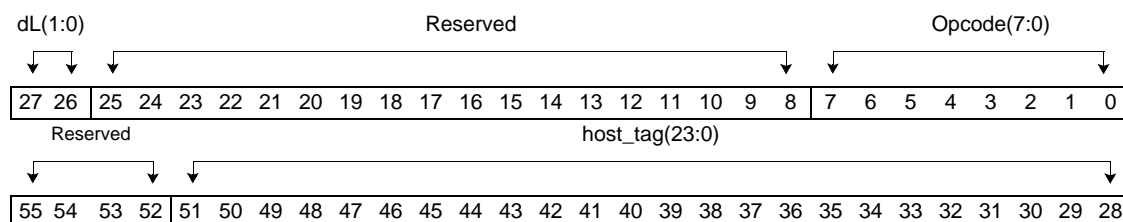
- Issue a **sync(at_stream)** command to each **stream_id** using the translation. Once all **sync** commands have received a response²⁰, the AFU sends **xlate_release** to any *one* of the **stream_id** to complete the **kill_xlate** command.
- Issue a **sync(all_stream)** command. Once the AFU receives a **sync_done** response, the AFU sends **xlate_release** to any *one* of the **stream_id** to complete the **kill_xlate** command.

An AFU that is using a single **stream_id** for all commands using the TLX.vc.3 virtual channel is not required to issue a **sync** command. **xlate_release** shall be sent using the **stream_id** used by the AFU.

Once the AFU has released the address translation, the AFU shall not use the translation for any purpose. Any lines held in an AFU_{C2} L1 cache using this translation shall not be updated or referenced by the AFU_{C2} processing element.

This command is posted. No response is sent for this command.

host_tag update on synonym done	synonym_done	'0101 0100'
Cacheable pushes	TLX.vc.2	2



This command is issued after modification, if any, of the **host_tag** entry specified by a **synonym_detected** command has completed and the **host_tag** entry is unlocked. See *Section 1.5 Host tags* on page 39 and *Section A.10 Host tag locking transactions* on page 235.

²⁰. **sync_done**

Approved

The host has locked the host_tags associated with the specification of the **synonym_detected** sent by the host. The AFU's **synonym_done** command is used by the host to unlock the host_tags.

The command is posted, no response is sent due to this command.

Cast out	castout	'0101 0101'
Cacheable pushes	TLX.vc.2	2



castout is used by the AFU to update the Host's cache directory of the line specified by the host_tag. The cmd_flag field is specified in *Table 2-15*.

Table 2-15. The command flag specification for **castout**

cmd_flag bit	Description
3	Reserved
2	Reserved
1	update_cache_state_flag. Indicates that the cache_state field contains the update value to be applied to the Host directory entry found using the host_tag and the dLength. Only downgrades from the current state are permitted. See <i>Table 1-1 Cache state descriptions</i> on page 32 for legal state downgrades.
0	mru_update_flag. Indicates that the AFU has been using the cache entry specified by the host_tag and dLength. The Host updates any MRU information it may have specified in its directory to reflect that the cache entry specified has been used by the AFU.
cmd_flag encode specification: 0000 Reserved 0001 mru_update_flag; cache_state field is reserved. 0010 Reserved 0011 update_cache_state flag, mru_update_flag. The MRU update action is undefined on a cache state transition to an I state. All other encodes are reserved.	

The combination of the host_tag and the dLength is used to determine the number of host directory entries that are affected by the command. See *host_tag arithmetic on page 20* for details.

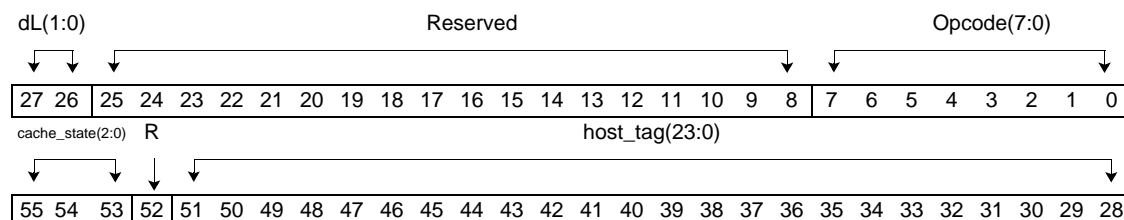
The **castout** command does not move data and is used to update MRU information in the host's cache directory, or update the host proxy cache directory's state to the cache state specified by the cache state field (cache_state). Legal host proxy directory cache state transitions can be found in *Table 7-2 Cache state transition errors* on page 208. See *Section 1.3 AFU_{C2}* on page 30 for cache states and *Table 1-3 L1 EA Cache state change request and notification* on page 35 to determine when a cache state transition shall be reported to the host.

A transition to a cache state of I shall only be specified when the host_tag entry or entries specified by the host_tag field and dLength are invalidated. That is, when synonyms are fully supported, no synonym entries pointed to by the host_tags specified are left in the AFU_{C2} cache when a transition to I is specified.

Approved

castout is a posted operation, no response is sent due to this command.

Cast out with data push	castout.push	'0101 0110'
Cacheable pushes	TLX.vc.2, TLX.dcp.2	2



castout.push is used by the AFU to update the Host's cache directory of the line specified by the `host_tag`.

The combination of the `host_tag` and the `dLength` (`dL`) is used to determine the number of host proxy cache directory entries that are affected by the command. See *host_tag arithmetic on page 20* for details.

The **castout.push** command may modify the host's proxy cache directory and it moves data to the Host's MEM. The command shall update the host proxy cache directory's state to the cache state specified by the cache state field (`cache_state`). Legal host proxy directory cache state transitions can be found in *Table 7-2 Cache state transition errors* on page 208. See *Section 1.3 AFU_{C2}* on page 30 for cache states and *Table 1-3 L1 EA Cache state change request and notification* on page 35 to determine when a cache state transition shall be reported to the host.

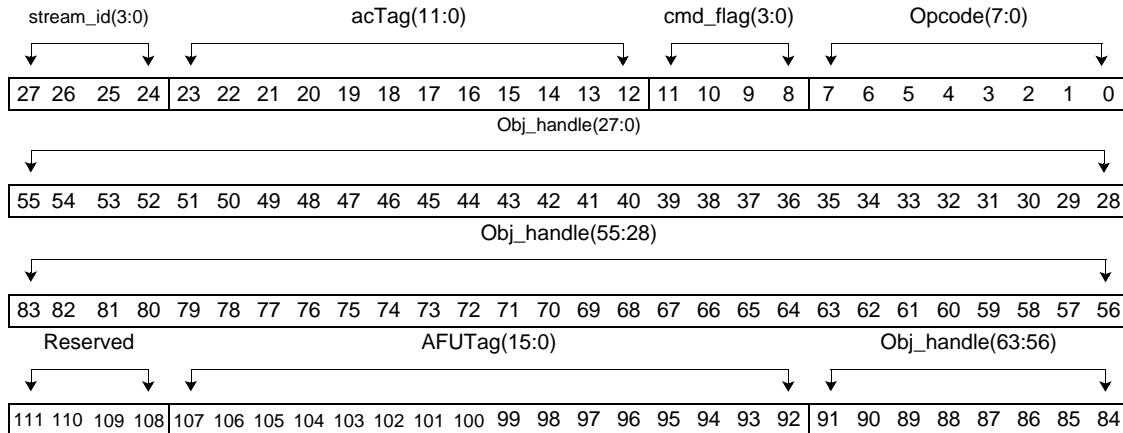
A transition to a cache state of I shall only be specified when the `host_tag` entry or entries specified by the `host_tag` field and `dLength` are invalidated. That is, when synonyms are fully supported, no synonym entries pointed to by the `host_tags` specified are left in the AFU_{C2} cache when a transition to I is specified.

This command is specified with immediate data. The data shall be sent using either 32- or 64-byte data carriers or a combination of both. Credits for both the VC and DCP shall be obtained before this command is serviced by the TLX.

castout.push is a posted operation, no response is sent due to this command.

Approved

Interrupt Request	intrp_req intrp_req.s	'0101 1000' '0101 1001'
message	TLX.vc.3	4



This command is used to request interrupt service on the host. No data is transferred with this request. The specification of the object handle and the cmd_flag is found in the host's platform architecture.

The response to this command is **intrp_resp**.

Engineering Note

The AFUTag is passed back to the TLX in a response packet and has no control function in the TL as described in *Table 2-1 TL and TLX command operands* on page 48. The **intrp_resp**'s response code specification of **intrp_pending** indicates to the TLX that a subsequent **intrp_rdy** command is sent when the host is ready to service the interrupt. The **intrp_rdy** command contains the AFUTag that is specified in the original **intrp_req** command sent. While there are no requirements placed on the AFU to reserve the AFUTag used by the **intrp_req** command until the **intrp_rdy** command is received by the TLX, it is strongly recommended that an AFU implementation do so. It is an AFU implementation choice to use the AFUTag to precisely determine which interrupt to retry.

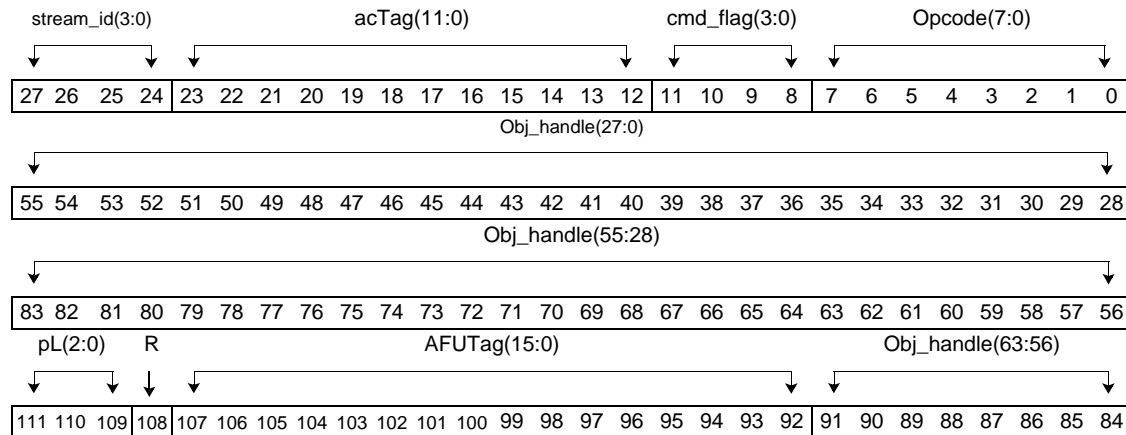
Developer Note

Both the command flag and the object handle fields specified for this command are specified by the host's platform architecture.

The attached OpenCAPI device provides MMIO space where the combination of the object handle and the command flag associated with this command are located. The manufacturer of the OpenCAPI device determines the number of command-flag and object-handle combination entries supported based on what is supported by the host and the function provided by the device.

Approved

Interrupt Request	intrp_req.d intrp_req.d.s	'0101 1010' '0101 1011'
message	TLX.vc.3, TLX.dcp.3	4



This command is used to request interrupt service on the host. Data, with the length specified by the pLength (pL) field, is transferred with this request. The data shall be sent in a 64-byte data flit. The alignment of the data within the data flit is specified by the host's platform architecture. The specification of the object handle and the command flag is found in the host's platform architecture.

The response to this command is **intrp_resp**.

Engineering Note

The AFUTag is passed back to the TLX in a response packet and has no control function in the TL as described in *Table 2-1* on page 48. The **intrp_resp**'s response code specification of **intrp_pending** indicates to the TLX that a subsequent **intrp_rdy** command is sent when the host is ready to service the interrupt. The **intrp_rdy** command contains the AFUTag that is specified in the original **intrp_req.d** command sent. While there are no requirements placed on the AFU to reserve the AFUTag used by the **intrp_req.d** command until the **intrp_rdy** command is received by the TLX, it is strongly recommended that an AFU implementation do so. It is an AFU implementation choice to use the AFUTag to precisely determine which interrupt to retry.

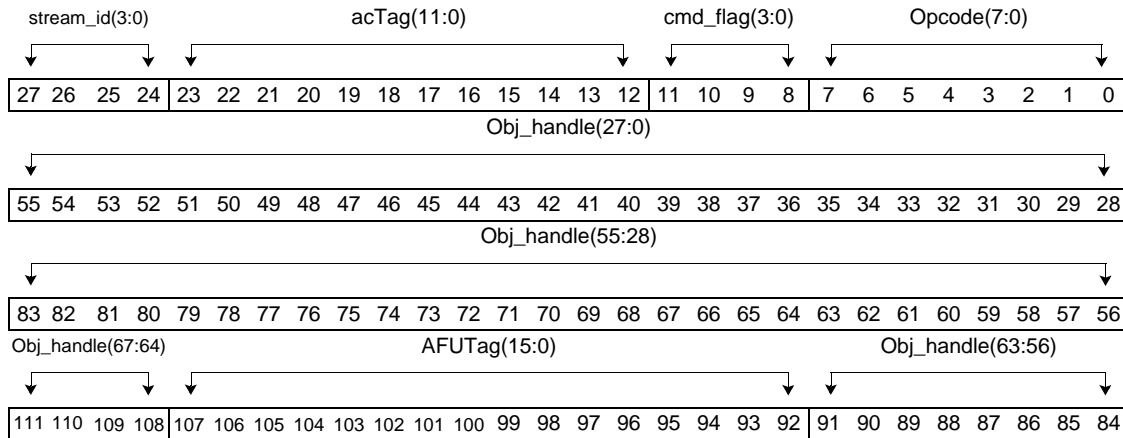
Developer Note

The command flag, data, and the object handle fields specified for this command are specified by the host's platform architecture.

The attached OpenCAPI device provides MMIO space where the combination of the object handle, data, and the command flag associated with this command are located. The manufacturer of the OpenCAPI device determines the number of command-flag and object-handle combination entries supported based on what is supported by the host and the function provided by the device.

Approved

Wake host thread	wake_host_thread wake_host_thread.s	'0101 1100' '0101 1101'
message	TLX.vc.3	4



This command is used to wake a thread on the host. The specification of the object handle and the command flag is found in the host's platform architecture.

This is a non-posted command. Results are returned to the AFU using **wake_host_resp**.

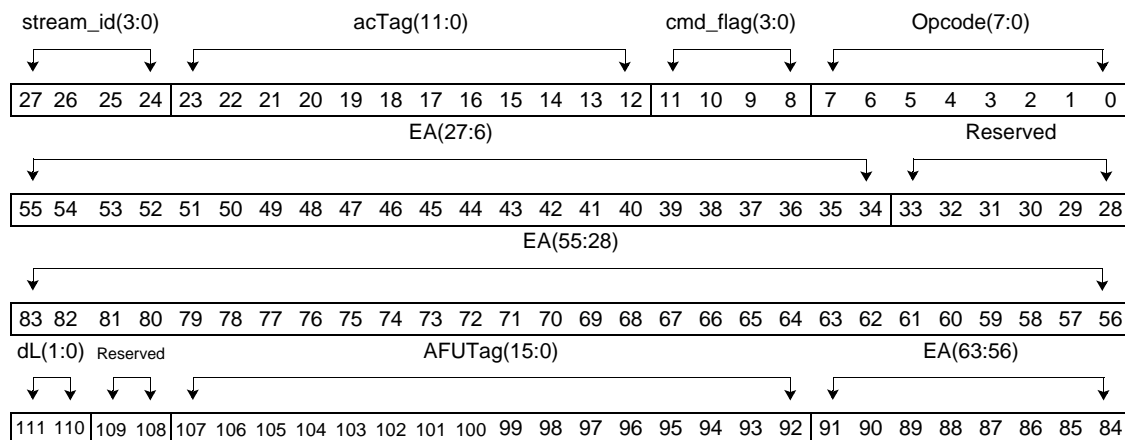
Developer Note

See the Developer note found in the description of **intrp_req** for details on the specification of the object handle and command flag and the requirements this specification places on the OpenCAPI device.

wake_host_resp indicates if the operation was successful, or if an interrupt is required.

Approved

Upgrade State	upgrade_state	'0110 0000'
cacheable read	TLX.vc.3	4



The AFU has determined that the state of the line held is insufficient for a write operation. The AFU is requesting that the cache state held in the AFU L1 cache be upgraded without the need for a data transfer. The address specified by the EA shall be naturally aligned based on the length of the data block as specified by the dLength (dL) field. The cache state upgrade is specified in the cmd_flag field. Write permissions are required for this request to be successful.

When the address translation in the host determines that the line is cache inhibited, the operation results in a response indicating an I state. See *Table 2-21* in the description of **synonym_detected**.

See the description of the cmd_flag in *Table 2-16*.

*Table 2-16. The cmd_flag specification for **upgrade_state*** (Page 1 of 2)

cmd_flag(3:0)	Description
'0000' - '0111'	Reserved
'1000'	I → M. Current cache state is I, request to upgrade to an M state. The data for the block is set to an all 0 state (modified state of the line) in the AFU L1. All other cache holding the line shall transition to an I state.

Approved

Table 2-16. The *cmd_flag* specification for **upgrade_state** (Page 2 of 2)

cmd_flag(3:0)	Description
'1001'	<p>I → E_i. Current cache state is I, request to upgrade to an E_i state. The data for the block is marked invalid in the AFU L1. All cache holding the line shall transition to an I state. The host shall insure that any cache holding the line in a dirty state (M) shall push the line back to MEM.</p> <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <p>Developer Note</p> <p>One use of this upgrade is to allow the AFU to warm up its L1 cache and replace the full cache block.</p> <p>A sparse update might be possible and is dependent on the host proxy and the limitations of the host coherence protocol. For example, a sparse update could be accomplished by first getting the line using a I->E_i upgrade request and then issuing a variation, not currently specified in the architecture, of dma_w.be or dma_pr_w commands. The cache would need to maintain byte marks to track which bytes are held in cache and which are still retained in MEM. The host coherence protocol would have to be written to allow sparse updates in this fashion.</p> <p>The variation on the dma_w.be or dma_pr_w commands would be needed to relax synonym checking by the host proxy as described in the developer note included in the description of dma_w.</p> </div>
'1010' - '1111'	Reserved

The responses to this command are **upgrade_resp**, **synonym_detected** and **read_failed**.

Engineering note

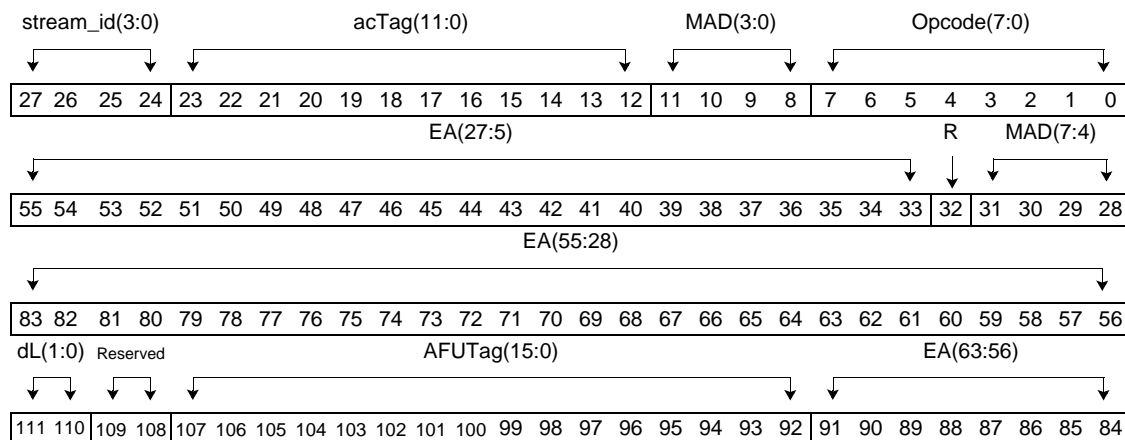
When there is a hit in the host's ATC and write permission is not granted, a **read_failed** response with a Resp_code of rty_req or xlate_pending occurs. In the case of a rty_req, the AFU may retry the operation since the Resp_code may have been due to insufficient resources in the host to initiate an interrupt to obtain write permission. A Resp_code of xlate_pending causes the AFU to wait for the results of the address translation found in a **xlate_done**.

During address translation it might be determined that the address has a memory attribute of cache inhibited. In this case **upgrade_resp** indicates a cache state of I. The AFU is responsible to take any corrective actions to ensure its application operates correctly.

See Figure A-9. *upgrade_state* TLX and TL interaction on page 230.

Approved

Read to store	read_me	'0110 1000'
cacheable read	TLX.vc.3	4



Request by the AFU to obtain a cacheable copy of the line specified in an exclusive (E) or modified (M) state with write permissions. The starting address specified by the EA supports a *critical OW request*. The data is a *naturally aligned data block* with a length specified by the dLength field (dL(1:0)). The dLength specifies the length of the AFU's cache line. See the host's platform architecture for the specification of the MAD field.

The responses to this command are **cl_rd_resp**, **cl_rd_resp.ow**, **synonym_detected**, and **read_failed**.

When write authority is not granted, the host might attempt to re-translate the address to obtain write permissions. If write permission is not authorized for this address and address context combination, the results are seen in **read_failed** responses and **xlate_done** commands.

When the address translation in the host determines that the line is cache inhibited, the operation results in a response indicating an I state. See Table 2-21 in the description of **synonym_detected**.

Engineering note

When there is a hit in the host's ATC and write permission is not granted, a **read_failed** response with a Resp_code of rty_req or xlate_pending occurs. In the case of a rty_req, the AFU may retry the operation since the Resp_code may have been due to insufficient resources in the host to initiate an interrupt to obtain write permission. A Resp_code of xlate_pending causes the AFU to wait for the results of the address translation found in a **xlate_done**.

It is strongly recommended that an AFU wait for the **xlate_done** before retrying the command. However, using a retry back off mechanism is permitted to determine when to retry the command. Such an implementation shall examine **xlate_done** for the results of the address translation and take action based on those results.

During address translation it might be determined that the address has a memory attribute of cache inhibited. In this case the Host bus shall obtain the data and return the data to the AFU indicating a cache state of I.

The state the line is returned is determined by the host's coherency protocol and is specified in the **cl_rd_resp**, **cl_rd_resp.ow**, or **synonym_detected** response packet.

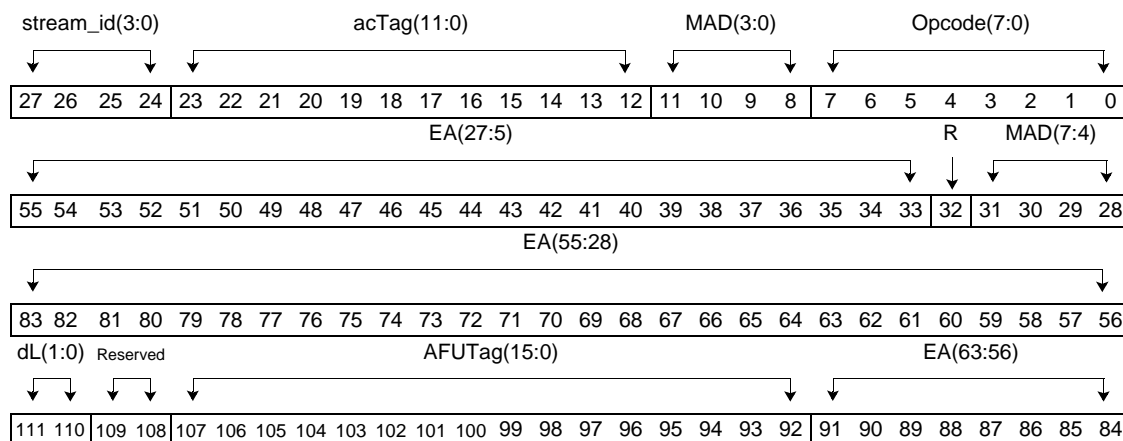
Approved

Developer note

To store into a line that an AFU_{C2} is holding in an S state, the AFU_{C2} could issue a **read_me** to obtain the line with write permissions (M or E state).

The AFU_{C2} is not required to evict the line prior to issuing the **read_me** command. The AFU must be able to handle the **synonym_detected** response that will occur by changing the state of the line as specified by the `cache_state` returned.

Read to load	read_mes	'0110 1001'
cacheable read	TLX.vc.3	4



Request by the AFU to obtain a cacheable copy of the line specified in M, E or S state. The state returned is dependent on the permissions granted by the address context for the specified data which is determined during address translation.

- If write permission is not granted, then the highest cache state returned is S.
- If write permission is granted, then a cache state of M, E, or S may be returned. The state returned is dependent on the host's coherence protocol.

When the address translation in the host determines that the line is cache inhibited, the operation results in a response indicating an I state. See *Table 2-21* in the description of **synonym_detected**.

The starting address specified by the EA supports a *critical OW request*. The data is a *naturally aligned data block* with a length specified by the dLength field (dL(1:0)). The dLength specifies the length of the AFU's cache line. See the host's platform architecture for the specification of the *MAD* field.

The responses to this command are **cl_rd_resp**, **cl_rd_resp.ow**, **synonym_detected**, and **read_failed**.

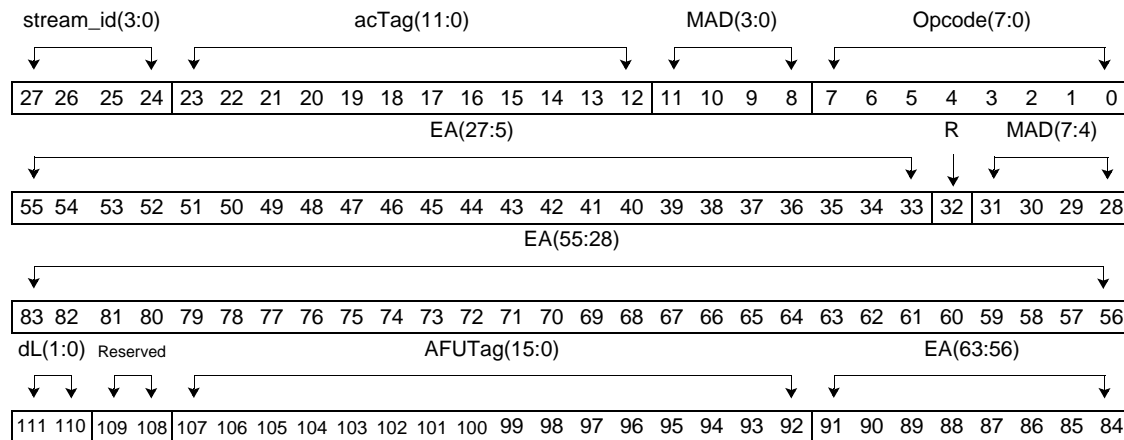
The state the line is returned is determined by the host's coherency protocol and the returned state may not be the state requested. The state returned is specified in the **cl_rd_resp**, **cl_rd_resp.ow**, or **synonym_detected** response packet.

Engineering Note

During address translation it might be determined that the address has a memory attribute of cache inhibited. In this case the Host bus shall obtain the data and return the data to the AFU indicating a cache state of I.

Approved

Read to reference	read_s	'0110 1010'
cacheable read	TLX.vc.3	4



Request by the AFU to obtain a cacheable copy of the line specified in a shared (S) state. The starting address specified by the EA supports a *critical OW request*. The data is a *naturally aligned data block* with a length specified by the dLength field (dL(1:0)). The dLength specifies the length of the AFU's cache line. See the host's platform architecture for the specification of the *MAD* field.

The responses to this command are **cl_rd_resp**, **cl_rd_resp.ow**, **synonym_detected**, and **read_failed**.

When the address translation in the host determines that the line is cache inhibited, the operation results in a response indicating an I state. See *Table 2-21* in the description of **synonym_detected**.

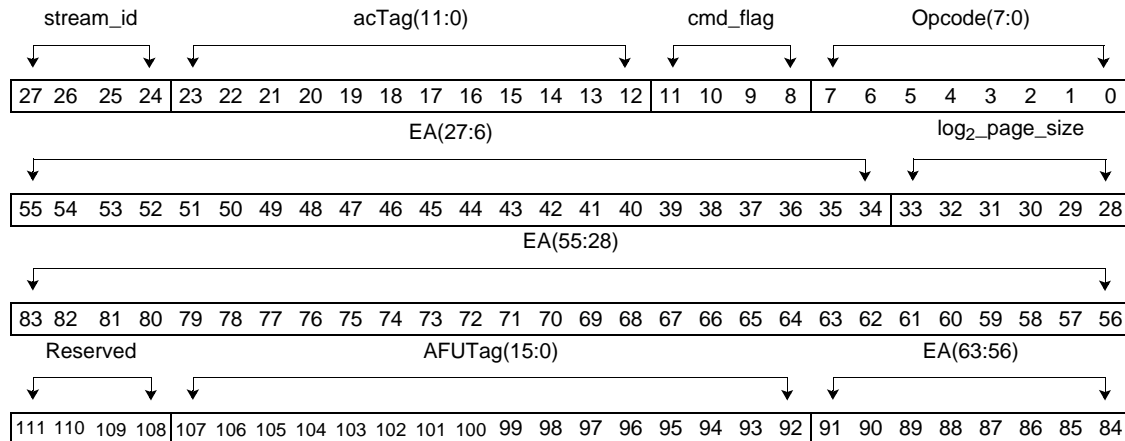
The state the line is returned is determined by the host's coherency protocol and the returned state may not be the state requested. The state returned is specified in the **cl_rd_resp**, **cl_rd_resp.ow**, or **synonym_detected** response packet.

Engineering Note

During address translation it might be determined that the address has a page attribute of cache inhibited. In this case the Host bus shall obtain the data and return the data to the AFU indicating a cache state of I.

Approved

Address translation prefetch	xlate_touch xlate_touch.n	'0111 1000' 0111 1100'
address translation management	TLX.vc.3	4



This command is used to request address translation prefetch for the address (EA) specified. The address can specify any 64-byte aligned address (EA). The **log₂_page_size** field specifies the page size for an age-out ATC entry request and is reserved for an address translation request. See the specification of the command flag, bit 0, in *Table 2-17*.

- The dot-n form indicates that the results of the address translation may not be installed into the host's ATC as part of ATC miss handling.

Table 2-17 provides the specification of the **cmd_flag** field. *Figure 2-1* on page 108 provides the architectural description of the command's operation.

*Table 2-17. The cmd_flag specification for **xlate_touch** (all forms)* (Page 1 of 2)

cmd_flag bit	Description	
3	0	Address translation request returns without a translated address (TA) (no_ta). The address translation cache is updated after a successful address translation
	1	Translated address (TA) request (ta_req). Translation requested with the return of a translated address.
2	0	Light-weight touch (lwt). Address translation stops and returns status if software intervention is required to complete the address translation request. Software intervention shall not be initiated.
	1	Heavy-weight touch (hwt). Address translation invokes software intervention if required to complete the address translation request. Status is returned immediately. The result of the software intervention is reported to the AFU using xlate_done .
1	0	Read-only access requested (ro). Read permission is requested by this address translation request. Write permission may be obtained.
	1	Write access requested (w). Write permission is requested by this address translation request.

Approved

Table 2-17. The *cmd_flag* specification for **xlata_touch** (all forms) (Page 2 of 2)

cmd_flag bit	Description
	0 Address translation request (xlate). log ₂ _page_size is reserved. 1 Age-out ATC entry request (age_out). log ₂ _page_size specifies the page size of the entry to be aged out.
0	<div><p>Engineering note</p><p>xlate_touch can be used to update the LRU mechanism of the host's ATC. cmd_flag(0) can be used to provide hints to the host.</p><p>0 Address translation is invoked. If an entry is found in the ATC, or one is added, that entry is marked MRU</p><p>1 Address translation is invoked. If an entry is found in the ATC, that entry is marked as LRU.</p><p>It is determined by the host implementation if early aging causes immediate invalidation of the matching ATC entries, the entries are marked as LRU, or no action is taken. A host implementation may chose to ignore the LRU hints described above.</p><p>The page size associated with an EA is provided in the touch_resp of a previous xlate_touch. The OpenCAPI device shall retain this information when making an age-out request.</p></div>

cmd_flag encode specification:

0000	xlate, lwt.ro, no_ta	1000	xlate, lwt.ro, ta_req
0001	age out	1001	Reserved
0010	xlate, lwt.w, no_ta	1010	xlate, lwt.w, ta_req
0011	Reserved	1011	Reserved
0100	xlate, hwt.ro, no_ta	1100	xlate, hwt.ro, ta_req
0101	Reserved	1101	Reserved
0110	xlate, hwt.w, no_ta	1110	xlate, hwt.w, ta_req
0111	Reserved	1111	Reserved

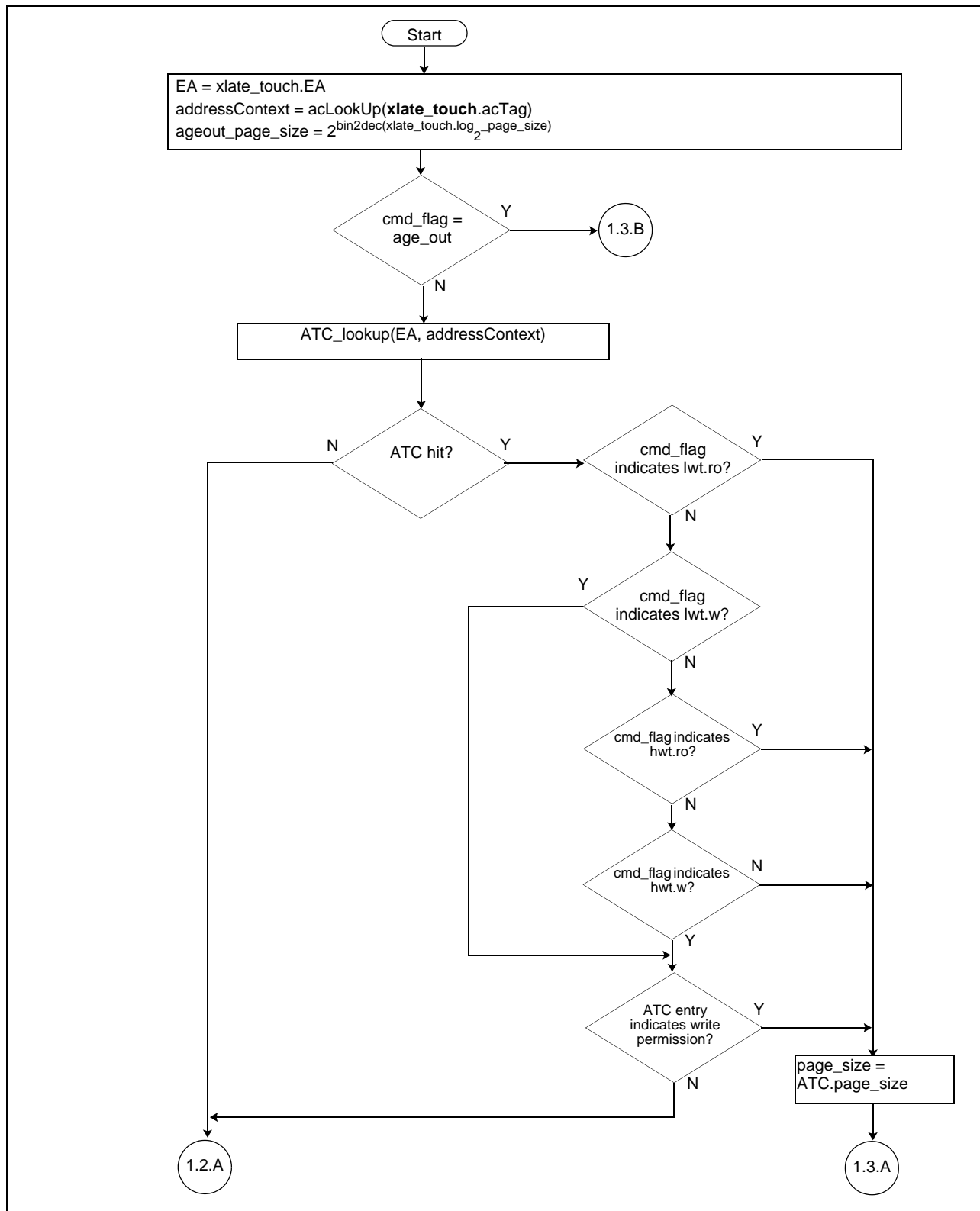
The use of reserved code points results in fatal errors. See *Table 7-1 Error event specification* on page 199.

The use of a dot-n form and age out is an error. See *Age out specified for xlate_touch.n* on page 199 for details.

The use of a dot-n form and ta_req is an error. See *ta_req specified for xlate_touch.n* on page 205 for details.

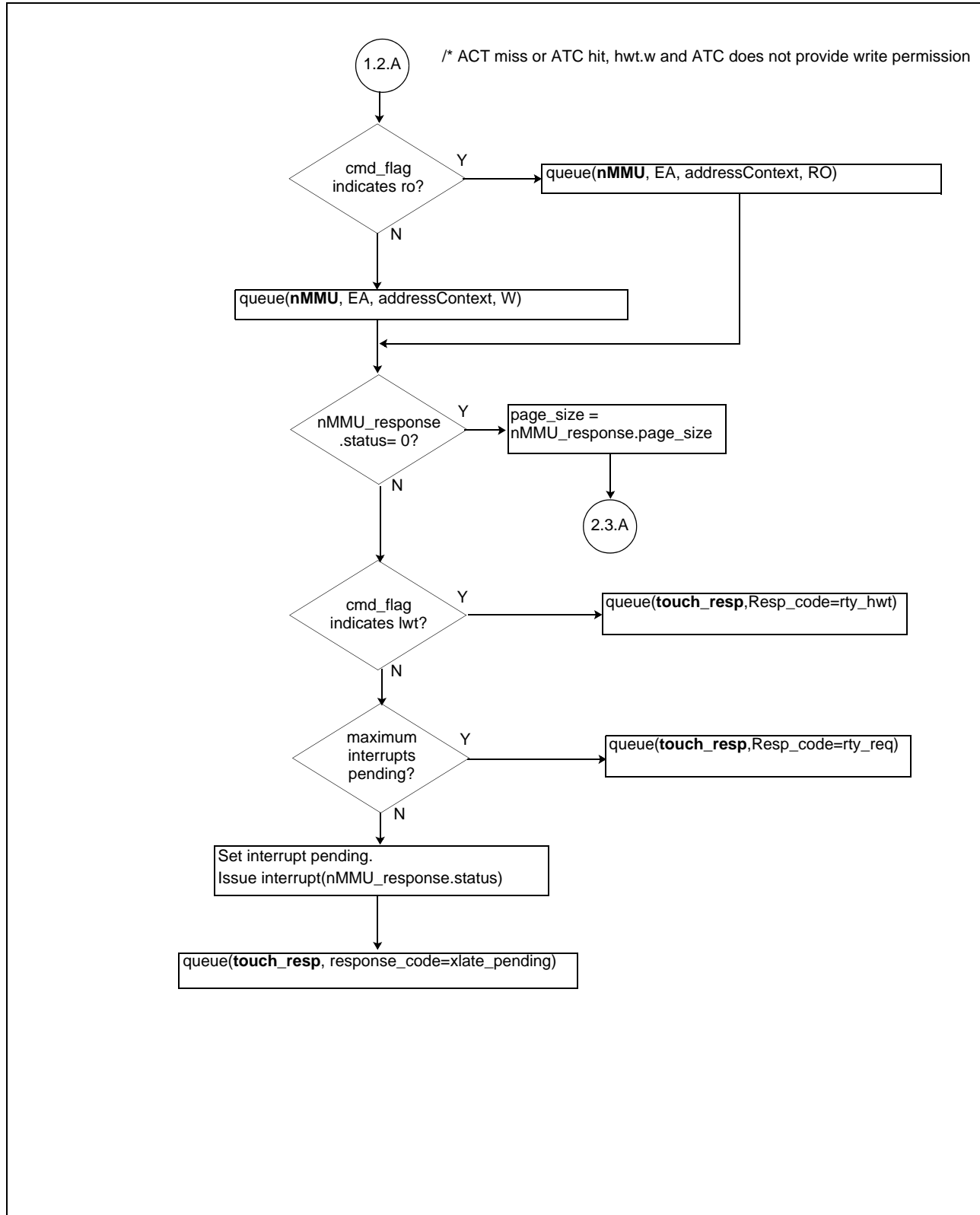
Developer Note

Figure 2-1 on page 108 does not show validation of the addressContext or the impacts of other hardware-driven events that might terminate this operation. See the specification of **touch_resp** for details of the result specification.

Figure 2-1. Address translation sequence: **xlate_touch** (Page 1 of 3)

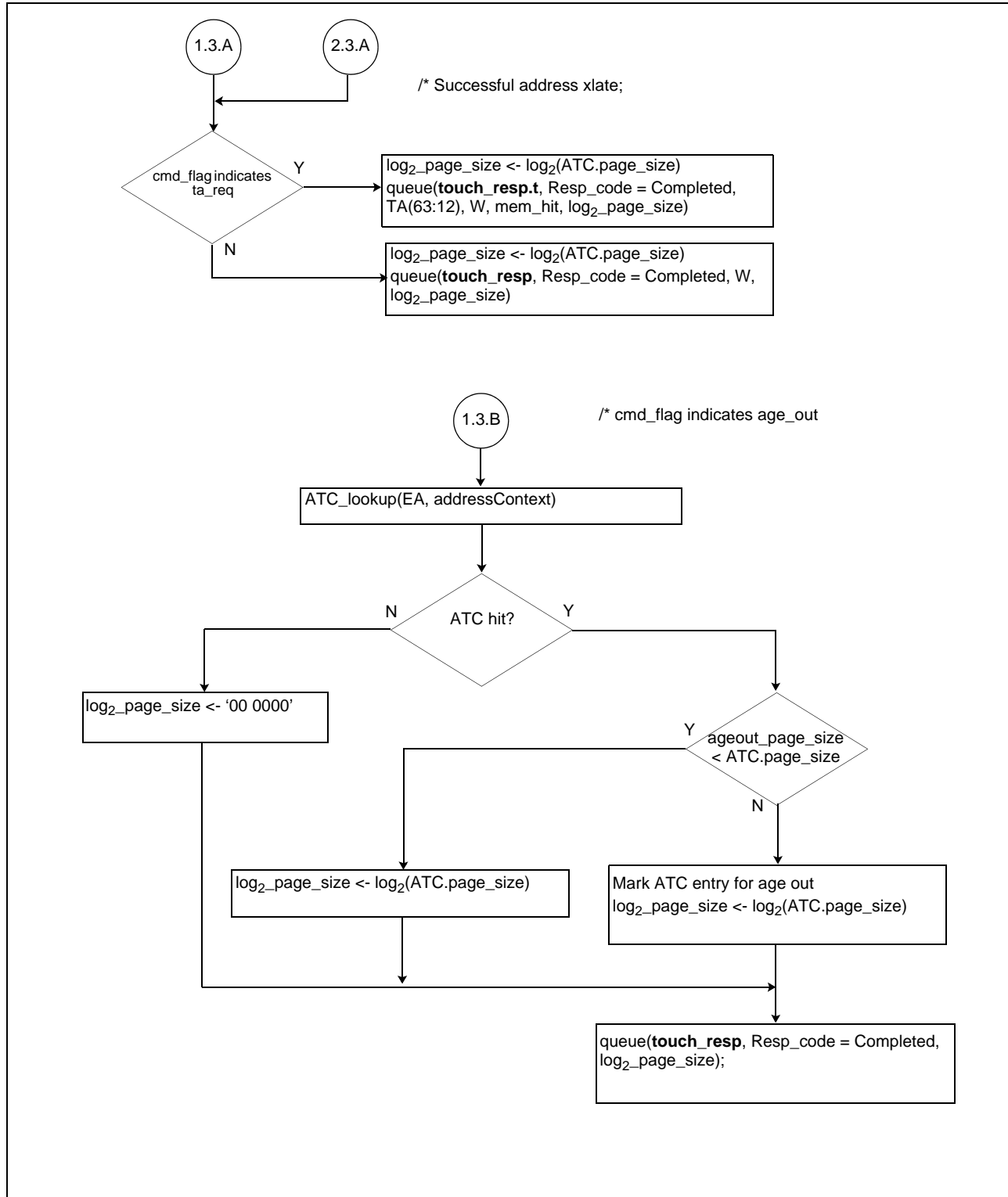
Approved

Figure 2-1. Address translation sequence: **xlate_touch** (Page 2 of 3)



Approved

Figure 2-1. Address translation sequence: *xlata_touch* (Page 3 of 3)



Approved

Results are returned to the AFU using **touch_resp** or **touch_resp.t**.

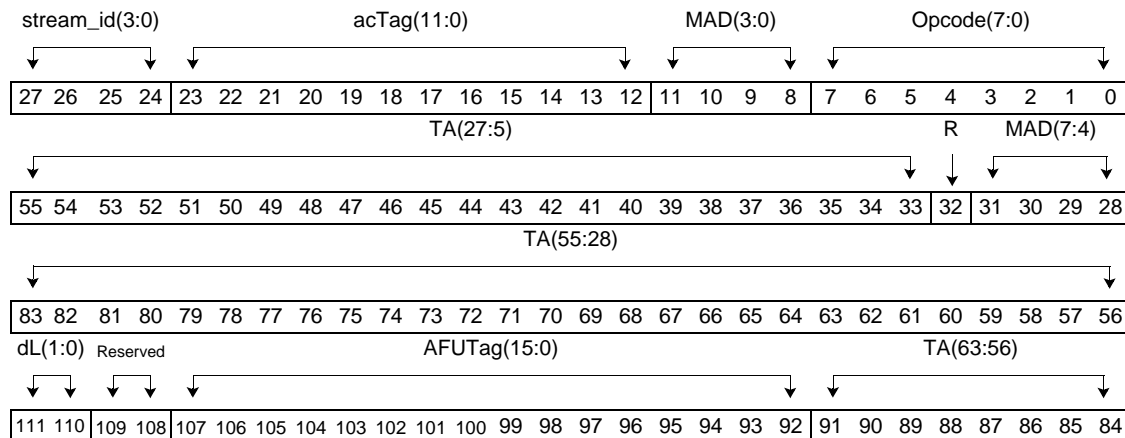
- **touch_resp** is used when a translated address has not been requested and the translation is successful, or when the Resp_code is not indicating complete.
- **touch_resp.t** is used when a translated address is requested and the translation is successful. The TA is being provided to the requester.

Engineering Note

The AFUtag is passed back to the TLX in a response packet and has no control function in the TL as described in *Table 2-1* on page 48. The **touch_resp**'s response code specification of **xlate_pending** indicates to the TLX that a subsequent **xlate_done** command is sent when the host is ready to service the translation request.

- The **xlate_done** command contains the AFUtag that is specified in the original **xlate_touch** command sent. While there are no requirements placed on the AFU to reserve the AFUtag used by the **xlate_touch** command until the **xlate_done** command is received by the TLX, it is strongly recommended that an AFU implementation do so. It is an AFU implementation choice to use the AFUtag to precisely determine which address translation to retry. The alternative is likely to be less efficient.
- **xlate_done** uses TL.vc.0. The implementation shall ensure that the **touch_resp** carrying the response code of **xlate_pending** is added to the VC prior to the **xlate_done**.

Read with no intent to cache	rd_wnltc.t rd_wnltc.t.s	'1001 0000' '1001 0001'
dma_read	TLX.vc.3	4



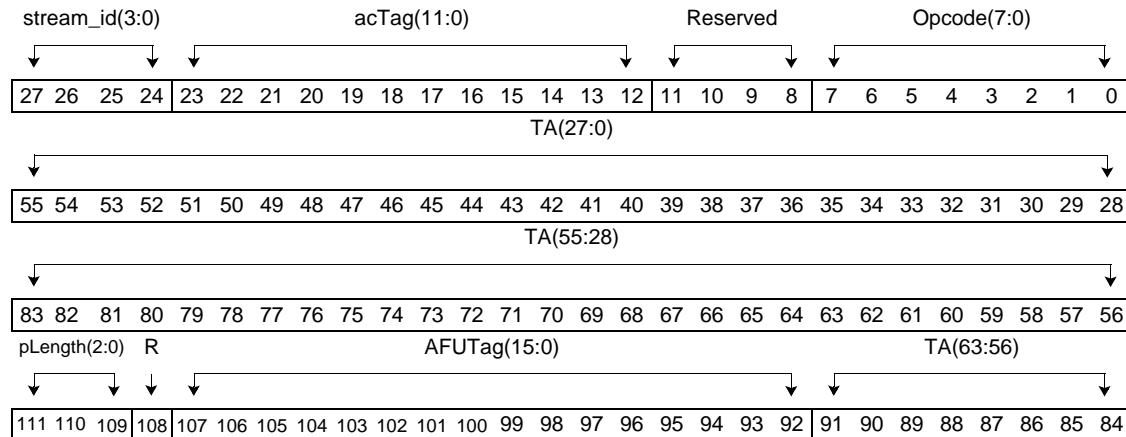
This command is identical to **rd_wnltc** with the exception of the address specification. These commands use the dot-t format which specifies the use of a previously obtained translated address (TA). The starting address specified by the TA supports a *critical OW request* as described in the **rd_wnltc** description.

- The dot-s form indicates that a *presync* is required prior to the execution of the command at the Host.

See the command description of **rd_wnltc** on page 83 for the operation of these commands.

Approved

Partial read with no intent to cache	pr_rd_wnrtc.t pr_rd_wnrtc.t.s	'1001 0010' '1001 0011'
pr_dma_read	TLX.vc.3	4

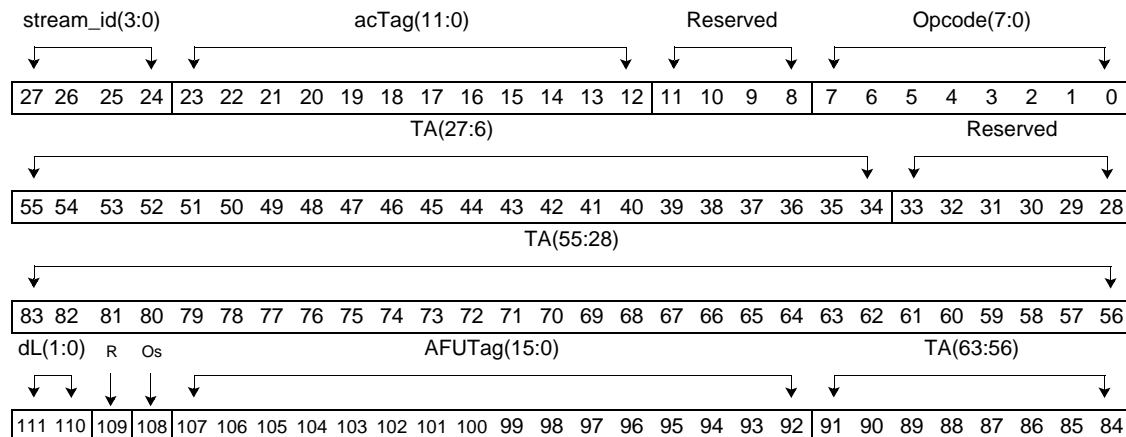


This command is identical to **pr_rd_wnrtc** with the exception of the address specification. These commands use the dot-t format which specifies the use of a previously obtained translated address (TA). The TA shall be naturally aligned based on the pLength (pL) field.

- The dot-s form indicates that a *presync* is required prior to the execution of the command at the Host.

See the command description of *pr_rd_wnrtc* on page 84 for the operation of these commands.

DMA write	dma_w.t.p dma_w.t.p.s	'1010 0010' '1010 0011'
dma_write	TLX.vc.3, TLX.dcp.3	4



This command is identical to **dma_w** with the exception of the address specification. These commands use the dot-t format which specifies the use of a previously obtained translated address (TA). The TA shall be naturally aligned based on the length of the data as specified by the dLength (dL) field.

- The dot-s form indicates that a *presync* is required prior to the execution of the command at the Host.

Approved

- The dot-p form indicates that the command is posted. That is, no response shall be returned for this command.

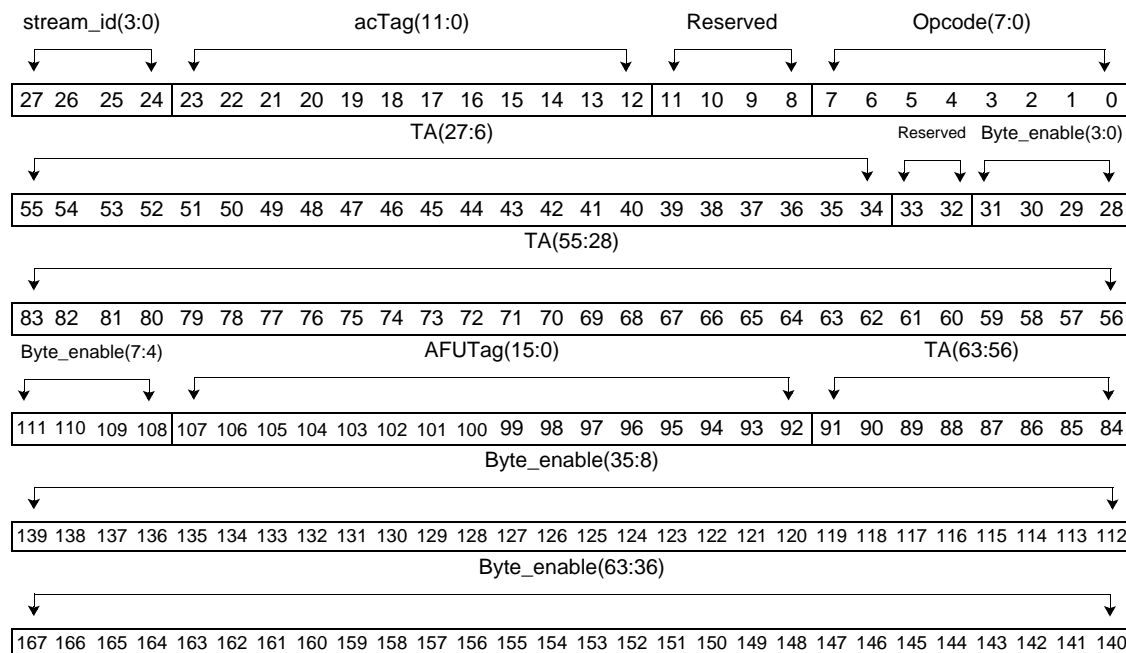
Engineering note

Without a response, a posted command cannot directly indicate to the AFU that an error has occurred. The following errors might be detected for this command:

- At the host, the received data suffered an uncorrectable error, or might have been marked bad in the control flit associated with the data transfer, or might have been damaged in some other fashion while being transferred within the host. In this case, the host might mark the data as bad allowing the subsequent consumer of the data to detect the error.
- The TA specified by the command is not naturally aligned or is not recognized by the host or does not have the necessary write permission. This is a fatal error. See the description of a *Posted command error on page 204*.
- The operation failed due to an unspecified reason. See the description of a *Posted command error on page 204*.

See the command description of *dma_w* on page 85 for the operation of these commands.

Byte enable DMA Write	dma_w.be.t.p dma_w.be.t.p.s	'1010 1010' '1010 1011'
pr_dma_write	TLX.vc.3, TLX.dcp.3	6



This command is identical to **dma_w.be** with the exception of the address specification. These commands use the dot-t format which specifies the use of a previously obtained translated address (TA).

- The dot-s form indicates that a *presync* is required prior to the execution of the command at the Host.
- The dot-p form indicates that the command is posted. That is, no response shall be returned for this command.

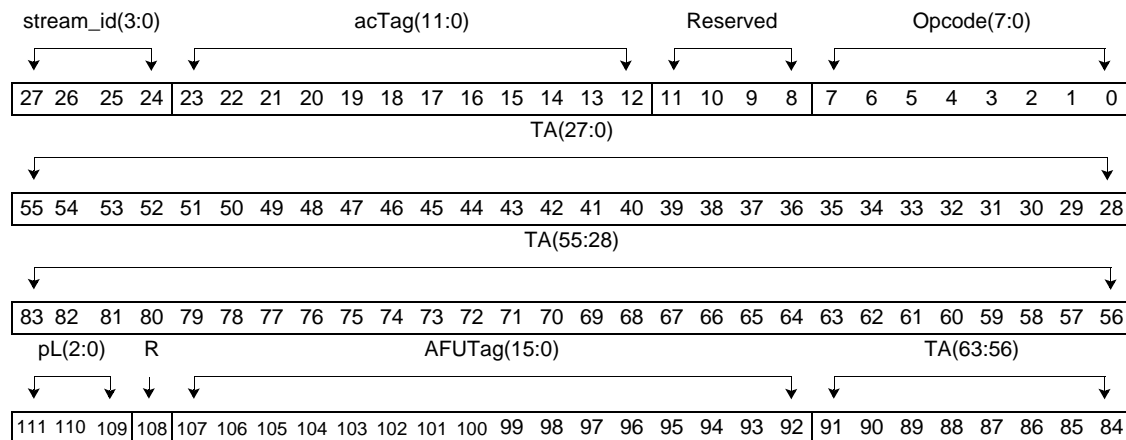
Engineering note

Without a response, a posted command cannot directly indicate to the AFU that an error has occurred. The following errors might be detected for this command:

- At the host, the received data suffered an uncorrectable error, or might have been marked bad in the control flit associated with the data transfer, or might have been damaged in some other fashion while being transferred within the host. In this case, the host might mark the data as bad allowing the subsequent consumer of the data to detect the error.
- The TA specified by the command is not recognized by the host or does not have the necessary write permission. This is a fatal error. See the description of a *Posted command error* on page 204.
- The operation failed due to an unspecified reason. See the description of a *Posted command error* on page 204.

See the command description of *dma_w.be* on page 86 for the operation of these commands

DMA parital write	dma_pr_w.t.p dma_pr_w.t.p.s	'1011 0010' '1011 0011'
pr_dma_write	TLX.vc.3, TLX.dcp.3	4



This command is identical to **dma_pr_w** with the exception of the address specification. These commands use the dot-t format which specifies the use of a previously obtained translated address (TA). The TA shall be naturally aligned based on the length of the data as specified by the pLength (pL) field.

- The dot-s form indicates that a *presync* is required prior to the execution of the command at the Host.
- The dot-p form indicates that the command is posted. That is, no response shall be returned for this command.

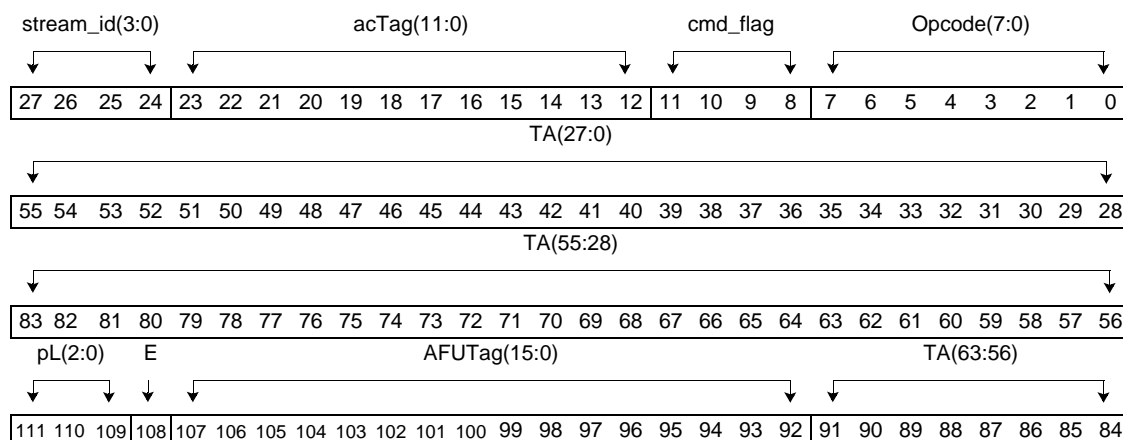
Engineering note

Without a response, a posted command cannot directly indicate to the AFU that an error has occurred. The following errors might be detected for this command:

- At the host, the received data suffered an uncorrectable error, or might have been marked bad in the control flit associated with the data transfer, or might have been damaged in some other fashion while being transferred within the host. In this case, the host might mark the data as bad allowing the subsequent consumer of the data to detect the error.
- The TA specified by the command is not naturally aligned, or is not recognized by the host, or does not have the necessary write permission. This is a fatal error. See the description of a *Posted command error on page 204*.
- The operation failed due to an unspecified reason. See the description of a *Posted command error on page 204*.

See the command description of *dma_pr_w* on page 87 for the operation of these commands

AMO read	amo_rd.t amo_rd.t.s	'1011 1000' '1011 1001'
atomics.r	TLX.vc.3	4



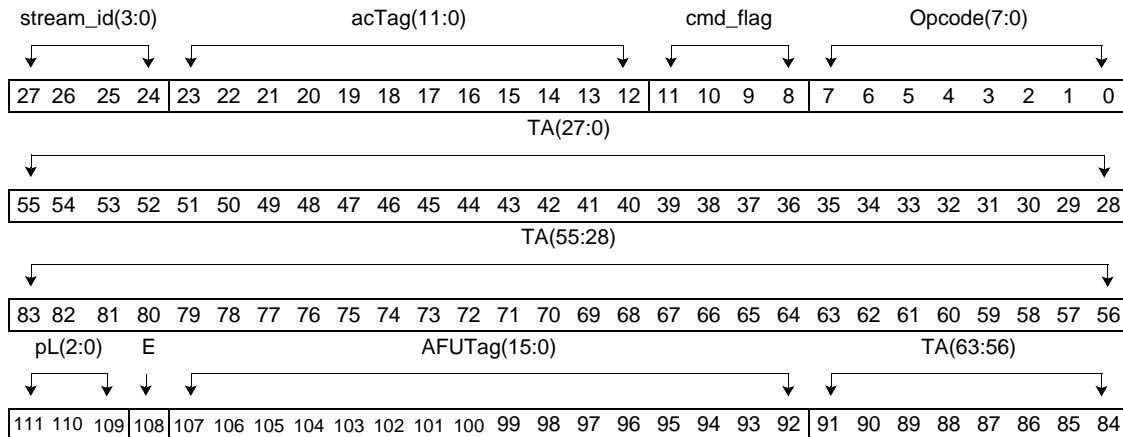
This command is identical to **amo_rd** with the exception of the address specification. These commands use the dot-t format which specifies the use of a previously obtained translated address (TA). The TA shall be naturally aligned as specified in the **Operation** section of the **amo_rd** command description.

- The dot-s form indicates that a *presync* is required prior to the execution of the command at the Host.
- The dot-p form indicates that the command is posted. That is, no response shall be returned for this command.

See the command description of *amo_rd* on page 88 for the operation of these commands

Approved

AMO read write	amo_rw.t amo_rw.t.s	'1100 0000' '1100 0001'
atomics.rw	TLX.vc.3, TLX.dcp.3	4



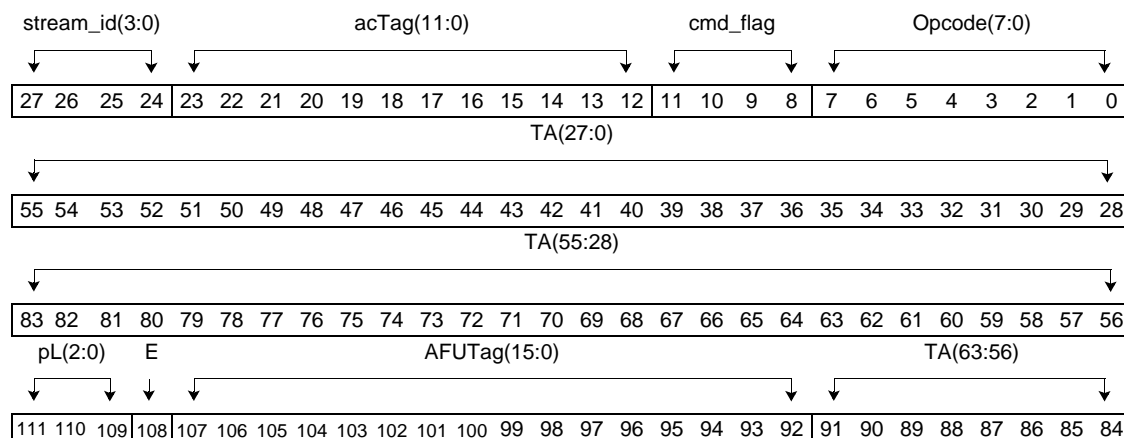
This command is identical to **amo_rw** with the exception of the address specification. These commands use the dot-t format which specifies the use of a previously obtained translated address (TA). The TA shall be naturally aligned. Operands are located in data carriers as described in the Operation section of the **amo_rw** command description.

- The dot-s form indicates that a *presync* is required prior to the execution of the command at the Host.
- The dot-p form indicates that the command is posted. That is, no response shall be returned for this command.

See the command description of *amo_rw* on page 89 for the operation of these commands

Approved

AMO write	amo_w.t.p amo_w.t.p.s	'1100 1010' '1100 1011'
atomics.w	TLX.vc.3, TLX.dcp.3	4



This command is identical to **amo_w** with the exception of the address specification. These commands use the dot-t format which specifies the use of a previously obtained translated address (TA). The TA shall be naturally aligned based on the operand length specified by the pLength (pL) field.

- The dot-s form indicates that a *presync* is required prior to the execution of the command at the Host.
- The dot-p form indicates that the command is posted. That is, no response shall be returned for this command.

Engineering note

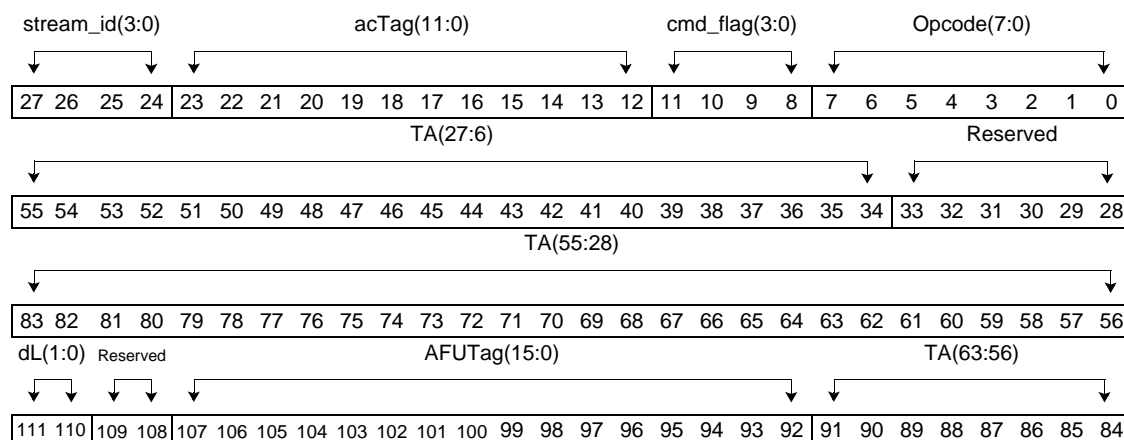
Without a response, a posted command cannot directly indicate to the AFU that an error has occurred. The following errors might be detected for this command:

- At the host, the received data suffered an uncorrectable error, or might have been marked bad in the control flit associated with the data transfer, or might have been damaged in some other fashion while being transferred within the host. In this case, the host might mark the data as bad allowing the subsequent consumer of the data to detect the error.
- The TA specified by the command is not naturally aligned, or is not recognized by the host, or does not have the necessary write permission. This is a fatal error. See the description of a *Posted command error on page 204*.
- The operation failed due to an unspecified reason. See the description of a *Posted command error on page 204*.

See the command description of *amo_w* on page 92 for the operation of these commands

Approved

Upgrade State	upgrade_state.t	'1110 0000'
cacheable read	TLX.vc.3	4



This command is identical to **upgrade_state** with the exception of the address specification. This command use the dot-t format which specifies the use of a previously obtained translated address (TA). The address specified by the TA shall be naturally aligned based on the length of the data block as specified by the dLength field.

If write authority is not granted, the operation shall fail with a **read_failed** response and a Resp_code of TA_adr_error.

See the command description of *upgrade_state* on page 101 for the operation of this command.

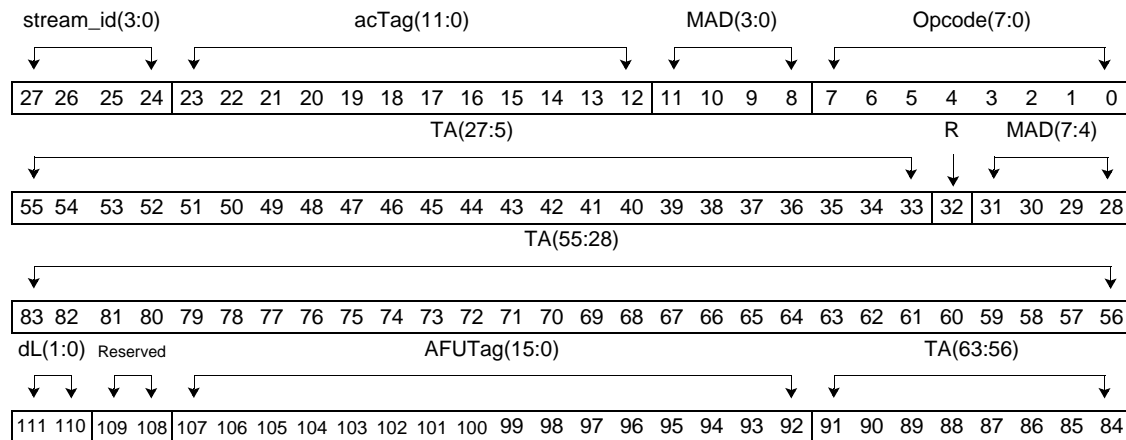
Engineering note

When a {TA, address context} is specified where write permission is not granted, a **read_failed** response with a Resp_code of TA_adr_error is returned. Since the {TA, address context} was not obtained with sufficient authority, the AFU might retry the operation after obtaining a new TA using **xlata_touch** and indicate that write permission is requested.

During address translation it might be determined that the address has a memory attribute of cache inhibited. In this case the Host bus shall obtain the data and return the data to the AFU indicating a cache state of I.

Approved

Read to store	read_me.t	'1110 1000'
cacheable read	TLX.vc.3	4



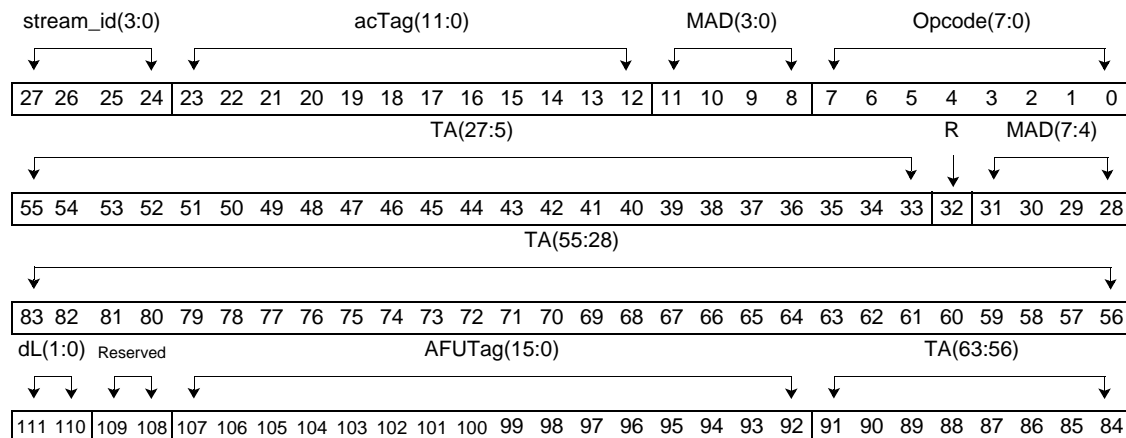
This command is identical to **read_me** with the exception of address specification. These commands use the dot-t format which specifies the use of a previously obtained translated address (TA).

The starting address specified by the TA supports a *critical OW request*. The data is a *naturally aligned data block* with a length specified by the dLength field (dL(1:0)). The dLength specifies the length of the AFU's cache line.

If write authority is not granted, the operation shall fail with a **read_failed** response and a Resp_code of TA_adr_error.

See the command description of *read_me* on page 103 for the operation of this command.

Read to load	read_mes.t	'1110 1001'
cacheable read	TLX.vc.3	4



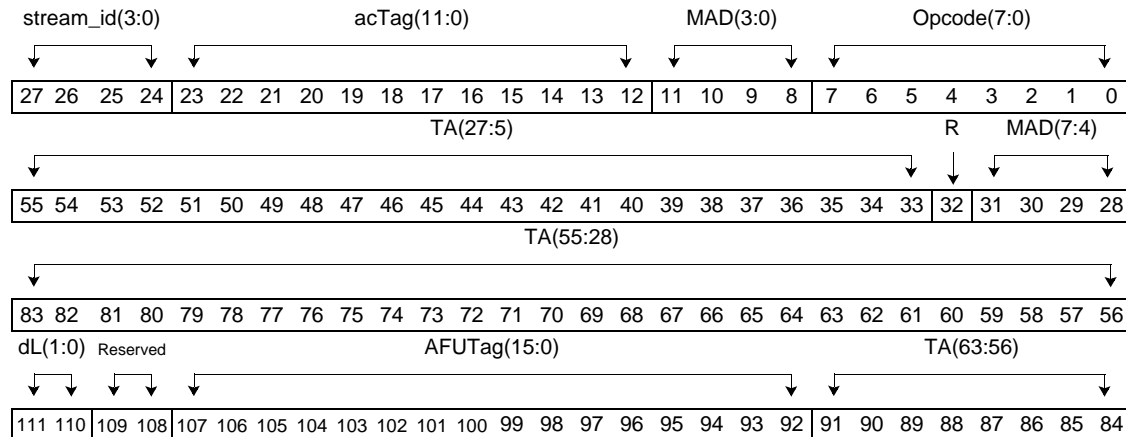
This command is identical to **read_mes** with the exception of address specification. These commands use the dot-t format which specifies the use of a previously obtained translated address (TA).

Approved

The starting address specified by the TA supports a *critical OW request*. The data is a *naturally aligned data block* with a length specified by the dLength field (dL(1:0)). The dLength specifies the length of the AFU's cache line.

See the command description of *read_mes* on page 104 for the operation of this command.

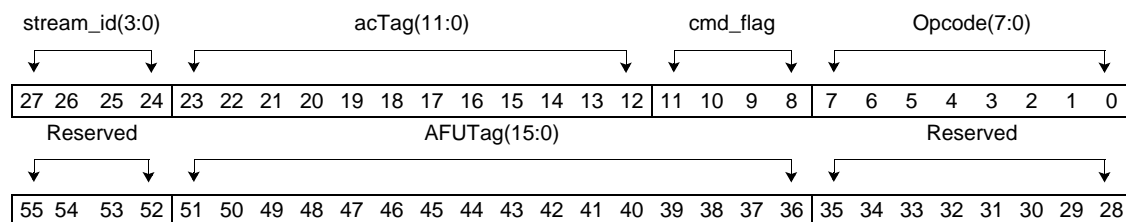
Read shared	read_s.t	'1110 1010'
cacheable read	TLX.vc.3	4



This command is identical to **read_s** with the exception of address specification. These commands use the dot-t format which specifies the use of a previously obtained translated address (TA).

See the command description of *read_s* on page 105 for the operation of this command.

Sync barrier	sync	'1110 1111'
barrier	TLX.vc.3	2



This command is used as a synchronization barrier applied at the head of either a TLX.vc.3 service queue or at the VC (TLX.vc.3) queue. The command is not removed from the head of the queue until all prior commands issued from the queue have completed. See *Section 3.3 TL Virtual channel and service queues* on page 168 for a description of head of queue blocking and *Section 3.4 TL Presync queues* on page 171 for a description of the prior command tracking function.

The acTag is not used for address translation or authorization. The acTag is used to determine the BDF and PASID that are used when assigning the command to a TLX.vc.3 service queue²¹. The acTag shall be valid and correctly formed, otherwise a fatal acTag error as described in *Section 7.1 Error events* on page 198 is reported.

Approved

Table 2-18 provides a specification of the cmd_flag field.

Table 2-18. The cmd_flag specification for **sync**

cmd_flag	Description
'0000'	sync(at_stream) The synchronization barrier occurs at the head of the TLX.vc.3 <i>service queue</i> .
'1000'	sync(all_stream) The synchronization barrier occurs at the head of the VC (TLX.vc.3) queue. The acTag is not used for this command to determine the TLX.vc.3 service queue since the command blocks at the head of the TLX.vc.3 queue and does not enter a TLX.vc.3 service queue. The command shall specify a valid entry. <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <p>Developer Note</p> <p>It is noted that the acTag field could have been specified as invalid if the cmd_flag is set to sync(all_stream). It was not specified in this way to simplify the implementation. That is, it simplified the implementation by not requiring a test of the cmd_flag to determine if the acTag should be checked.</p> </div>
all other values	Reserved

The response to this command is **sync_done**.

21. The **sync** command is assigned to a service queue when the cmd_flag is set to **sync(at_stream)**. See the description of the handling of the command when the cmd_flag is set to **sync(all_stream)** in Section 3.3.1 Host TLX command handling on page 168.

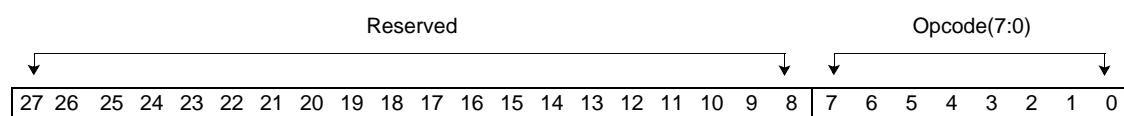
Approved

2.4 TL CAPP response packets

TL responses are sent from the host to the AFU. An alphabetical list of the TL responses follows; each response is hyperlinked to its specification. In this section, the TL response specifications are in opcode order.

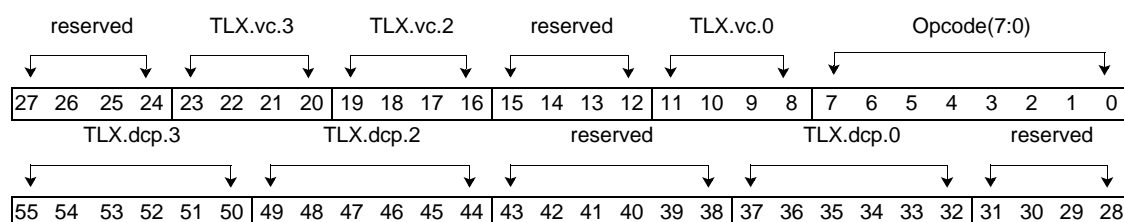
cl_rd_resp	cl_rd_resp.ow	intrp_resp
	nop	read_failed
read_response	read_response.ow	read_response.xw
sync_done	synonym_detected	touch_resp
upgrade_resp	wake_host_resp	write_response
		write_failed
		return_tlx_credits
		touch_resp.t

No operation	nop	'0000 0000'
NA	NA	1



This response has no operands and performs no action. It is discarded at the TLX.

Return TLX credits	return_tlx_credits	'0000 0001'
credit return	NA	2



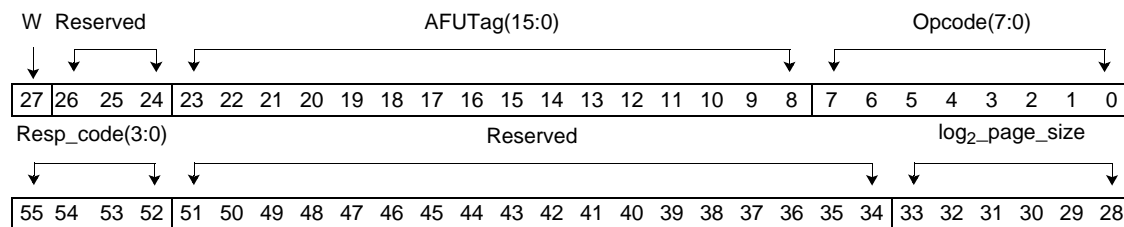
This response packet is used by the TL to return VC and DCP credits to the TLX. There is no VC associated with this response, and credits are not required to service this response. Each TLX.* field contains the number of credits being returned.

This response packet shall be placed only in slots 1 to 0 of any control flit using a template which specifies those slots as a 2-slot or larger location.

TLX.vc.{0, 2, 3} and TLX.dcp.{0, 2, 3} credits are returned. TLX credits are for resources owned by the TL that the TLX consumes. The TL controls the total number of credits for each of the VC and DCP it provisions the TLX with.

Approved

touch response	touch_resp	'0000 0010'
address translation management	TL.vc.0	2



This is a response to an **xlata_touch** command with no address translation return requested (no_ta), a **xlata_touch** with a request for a TA (ta_req) and the address translation fails. log₂_page_size and write permission (W) are valid only when the Resp_code = Completed, and when xlata_touch specifies address translation (xlata), otherwise the fields are reserved. The Resp_code field is specified in *Table 2-19*.

Table 2-19. The Resp_code specification for touch_resp

Resp_code encode	Description
'0000'	Completed. Address translation completed successfully.
'0001'	Retry using the heavy-weight touch specification (rty_hwt). The translation could not be completed using the light-weight touch (lwt) specified by the xlata_touch command.
'0010'	Retry request (rty_req). Indicates that the address translation could not be completed at this time. An address translation attempt may be made a later time. This is a long <i>back-off event</i> .
'0011'	Light-weight retry request (lw_rty_req). Use of this code point might be due to the lack of hardware resources in the host to service the command, or a transient error condition that the hardware is able to clear out on its own. The retry back off timer is specified in the devices configuration space and should be set to a value that allows the hardware recovery mechanism to complete before the device attempts to retry the operation. The operation may be retried by the device. This is a short back-off event.
'0100'	Translation pending (xlata_pending). Indicates that the address translation could not be completed. The ATC did not contain the translation, and software was invoked. An asynchronous xlata_done TL command shall be sent when software actions have completed. It is strongly recommended that the device wait for xlata_done to be received before retrying the operation. However, using a retry back off mechanism is permitted to determine when to retry the command. Such an implementation shall examine xlata_done for the results of the address translation and take action based on those results. <ul style="list-style-type: none"> xlata_done uses TL.vc.0. The implementation shall ensure that the touch_resp carrying the response code of xlata_pending is added to the VC prior to the xlata_done.
'0101' - '1011'	Reserved.
'1100'	Reserved.
'1101'	Reserved.
'1110'	Failed. The operation has failed and cannot be recovered. <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>Engineering Note</p> <p>It is strongly recommended that an implementation provide error collection facilities to indicate the reason for the Resp_code = Failed. The specification of the error collection facility should be documented in the host's platform architecture.</p> </div>
'1111'	Reserved.

Note: The errors specified by Resp_code do not include the fatal error conditions described in *Table 7-1* on page 199.

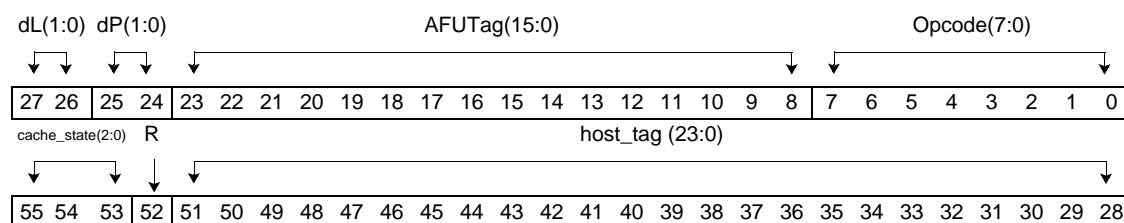
Approved

touch_resp responds to the TLX commands found in *Table 2-20*. For each command only the Resp_codes indicated with a Y may be used. Resp_code with N shall not be used.

Table 2-20. touch_resp Resp_code use by TLX command

TLX command	Completed (0)	rty_hwt (1)	rty_req (2)	lw_rty_req (3)	xlate_pending (4)	Failed (14)
xlate_touch	Y	Y	Y	Y	Y	Y
xlate_touch.n	Y	Y	Y	Y	Y	Y

Synonym detected	synonym_detected	'0000 0011'
Read data return	TL.vc.0	2



In response to a cacheable read²² or **upgrade_state** command from the AFU, the Host is reporting that the data associated with the command has been previously provided to the requester. This is a *synonym* case. The requester can access the data using the **host_tag** provided. When the command is a cacheable read and the AFU determines that the **host_tag** is no longer valid, the AFU may retry the operation²³. The AFU locks the **host_tags** specified by the **host_tag** field and the **dLength(dL)** field and *host_tag arithmetic* with the **synonym_detected** response. Additional actions are taken by the AFU to complete the TLX command that was originally issued. **synonym_done** shall be issued as part of the actions taken.

The **dLength** field indicates the amount of data referenced by this response. Multiple read response packets may be received for a single cacheable read or **upgrade_state** command. When the **dLength** field in the response does not match the full amount of data requested by the command, the **dPart** field is used to indicate the offset within the *naturally aligned data block* specified by the address in the command. For multiple responses to a single command, the **AFUtag** is unchanged. That is, only the **dLength** and **dPart** may vary when multiple responses are returned for a single command. For multiple responses to a single command, there is no order requirement placed by the architecture. That is, the TLX may see the values of **dPart** returned in any order.

- When multiple responses are received for a cacheable read command, a combination of **cl_rd_resp**, **cl_rd_resp.ow**, **synonym_detected**, and **read_failed** responses may be received.
- When multiple responses are received for a **upgrade_state** command, a combination of **upgrade_resp**, **synonym_detected**, and **read_failed** responses may be received.

Taken together, all responses shall contain a combination of implied length, **dLength** and **dPart** to cover the command's **dLength** specification.

22. **read_me**, **read_mes** or **read_s** and the dot-t variants of these commands are cacheable read commands.

23. This case might occur when the AFU is casting out the line referenced by the **host_tag**, making the **host_tag** invalid, while a read operation using a different address context and EA has responded with **synonym_detected** because the host has not yet seen the **castout** or **castout.push** from the AFU.

Approved

The AFU is not required to fully support synonyms. The minimum support an AFU shall provide on receipt of a **synonym_detected** response is waiting until all responses for the current operation have been received and scheduling a **castout** or **castout.push** of the data block specified by the host tag followed by a **synonym_done**. Once **synonym_done** has been issued, the AFU may re-issue the original request.

A **synonym_done** shall be issued for each **synonym_detected** received from the host²⁴.

Engineering note

Implementations that choose to provide the minimum support for synonyms should consider the case where a line is held in a read-only state, for example S, and a write to the line is requested by the AFU_{C2}.

The minimum synonym support would work, that is, the line would be evicted using a castout command due to the **synonym_detected** response and the operation, for example a **read_me**, would be retried. An implementation could instead detect this situation and evict the line before issuing the **read_me**. It should be noted that the **castout** and the **read_me** are not in the same VC, therefore, ordering at the host is not assured and the AFU_{C2} may still see a **synonym_detected** response. The AFU would see it's host_tag entry as invalid, so the information from the **synonym_detected** is discarded and the **read_me** would have to be reissued.

This operation could be shortened if the implementation supported upgrading an S state to an E state. The **read_me** getting a **synonym_detected** with an E state completes the operation since the S state can be upgraded to E and the data is already held in the AFU_{C2} data cache.

The cache_state indicates the state the cache line has been granted to the AFU. When the AFU finds the host_tag state is valid, the AFU shall take the line to the state specified by the cache_state field. When a cache state of I or E_I is indicated by this field, the data shall be marked invalid for the {EA, address context} specified in the original command. See *Table 2-21 synonym_detected formation and actions* on page 127 for details.

Data is not returned with this response. The AFU uses the host_tag to locate the data previously sent to the AFU. An AFU is unaware of a synonym until it receives a **synonym_detected** response. The following cases might result:

- The AFU might find the host_tag invalid. This might occur when the AFU has casted out the host_tag due to a **force_evict** TL command. This is not an error condition. To complete the operation, the AFU retries the original command.
- The AFU might find that the host_tag is valid and the data held in the AFU data cache is invalid. This is not an error condition. To complete the transaction, the AFU shall cast out the host_tag to an I state and then retry the transaction.

Table 2-21 specifies the **synonym_detected** response to all TLX commands that might receive a **synonym_detected** TL response. For completeness, the case where **cl_rd_resp** and **upgrade_resp** is also shown. The abbreviation of "sd" is used in the table to indicate **synonym_detected**. The final state of the host proxy cache is shown for the cases where write authority has been granted during address translation, where read only authority has been granted, and in the case where address translation indicates that the address specified by the command is not cacheable.

24. For example, consider when a device sends a **read_me** with a dLength of 256 bytes and the host cache line is 128 bytes. If both 128-byte segments of the AFUC2 line hit the host proxy cache, there would be two **synonym_detected** responses.

Approved

As noted in the header, the contents of the columns shows the value of the cache_state in the response and the final L2 cache state²⁵. Actions taken when Write authority is required by the command and the line is found to be cache inhibited are specified in the comments and footnotes found in *Table 2-21*. The order in which these actions are taken by the host determines which indication is provided to the device. Regardless of the order taken, the state of the L2 shall not be modified.

25. The final cache state of the host proxy cache is bounded by parenthesis. An X indicates the operation has failed and the response does not contain a cache state. See the footnote specified in the table for details.

Approved

Table 2-21. **synonym_detected** formation and actions Cache state field values and final host proxy states due to starting states and address translation (Page 1 of 7)

Command	Starting L2 state	Response	Address translation indicated access writes: Entry shows: Response cache_state (final L2 cache state)			Comments
			Write Authority	RO Authority	cache inhibited	
read_me read_me.t	I	cl_rd_resp	M/E (M/E) ³	X (I) ¹	I (I)	
	S	sd	E (E) ³	X (S) ¹	I (S) ²	(E (E)): When data is dirty with respect to the POC, the host shall clean the data passed to the AFU _{C2} .
	E	sd	E (E) ³	X (E) ¹	I (E) ²	The clean/dirty state of the cache line is known only to the AFU _{C2} which may have modified the line without informing the host.
	E _I	sd	E (E _I) ³	X (E _I) ¹	I (E _I) ²	The AFU is being directed to go to a state with valid data, with a response where no data is supplied. The AFU may have created data for the line. That is, the AFU _{C2} may be holding the line in an E state with the data marked as dirty, or may be in an M state. If the line is held in an E _I state by the AFU _{C2} EA L1 cache, it shall evict the line from the AFU _{C2} . That is, the host tag must be freed in order for the host to be able to send data in response to the command. The AFU _{C2} may then retry the operation until it obtains the data requested.
	M	sd	E/M (M) ³	X (M) ¹	I (M) ²	The AFU _{C2} holds the data in a dirty state and shall issue a castout.push when invalidating the host tag's use. When write authority is granted, the host may respond with a cache_state field value of either E or M. The AFU _{C2} shall track the clean/dirty state of the cache block associated with all valid host tags.
<ol style="list-style-type: none"> For read_me, a read_failed response is returned with a Resp_code of xlate_pending or rty_req while the host is attempting to obtain Write authority. If only RO authority is granted, the host informs the device using xlate_done and a Resp_code of adr_error. For read_me.t, a read_failed is returned with a Resp_code of TA_adr_error. Synonyms might not be detected by the host implementation when the {EA, address_context} address translation results in a storage attribute of cache inhibited. A host implementation might not examine the L2 and shall leave the L2 state unchanged. When the host does not examine the L2, the resulting response shall be cl_rd_resp with a cache state of I once the host has completed its non-cacheable read operation. For cache inhibited cases, see the cache inhibited column. For upgrade_state, a read_failed response is returned with a Resp_code of xlate_pending or rty_req while the host is attempting to obtain Write authority. If only RO authority is granted, the host informs the device using xlate_done and a Resp_code of adr_error. For upgrade_state.t, a read_failed is returned with a Resp_code of TA_adr_error. Finding the line in the L1 when the host's address translation results in cache inhibited appears to be an error with either the L1, the host's translation mechanism, or a combination of the two. When the host's address translation results in cache inhibited, synonyms might not be detected by the host implementation. Regardless of the host implementation's examination of the L2, the host implementation shall leave the L2 state unchanged. When the host does not examine the L2, the resulting response shall be upgrade_resp with a cache_state of I. 						

Approved

Table 2-21. **synonym_detected** formation and actions Cache state field values and final host proxy states due to starting states and address translation (Page 2 of 7)

Command	Starting L2 state	Response	Address translation indicated access writes: Entry shows: Response cache_state (final L2 cache state)			Comments
			Write Authority	RO Authority	cache inhibited	
read_mes read_mes.t	I	cl_rd_resp	M/E/S (M/E/S) ³	S (S) ³	I (I)	M/E/S (M/E/S): The cache state held by the host proxy cache at the conclusion of the host's coherence protocol actions to obtain the line is the state reported to the AFU _{C2} .
	S	sd	E/S (E/S) ³	S (S) ³	I (S) ²	E/S (E/S): The cache state held by the host proxy cache at the conclusion of the host's coherence protocol actions to obtain the line is the state reported to the AFU _{C2} . When a response of E (E) is returned, and when the data held by the AFU _{C2} is dirty with respect to the POC, the host shall clean the data prior to the response.
	E	sd	E (E) ³	S (E) ³	I (E) ²	The clean/dirty state of the cache line is known only to the AFU _{C2} which may have modified the line without informing the host.
	E _I	sd	E (E _I) ³	S (E _I) ³	I (E _I) ²	The AFU is being directed to go to a state with valid data, with a response where no data is supplied. The AFU may have created data for the line. That is, the AFU _{C2} may be holding the line in an E state with the data marked as dirty, or may be in an M state. If the line is held in an E _I state by the AFU _{C2} EA L1 cache, it shall evict the line from the AFU _{C2} . That is, the host tag must be freed in order for the host to be able to send data in response to the command. The AFU _{C2} may then retry the operation until it obtains the data requested.
	M	sd	E/M (M) ³	S (M) ³	I (M) ²	The AFU _{C2} holds the data in a dirty state and shall issue a castout.push when invalidating the host tag's use. When write authority is granted, the host may respond with a cache_state field value of either E or M. The AFU _{C2} shall track the clean/dirty state of the cache block associated with all valid host tags.
<ol style="list-style-type: none"> 1. For read_me, a read_failed response is returned with a Resp_code of xlate_pending or rty_req while the host is attempting to obtain Write authority. If only RO authority is granted, the host informs the device using xlate_done and a Resp_code of adr_error. For read_me.t, a read_failed is returned with a Resp_code of TA_adr_error. 2. Synonyms might not be detected by the host implementation when the {EA, address_context} address translation results in a storage attribute of cache inhibited. A host implementation might not examine the L2 and shall leave the L2 state unchanged. When the host does not examine the L2, the resulting response shall be cl_rd_resp with a cache state of I once the host has completed its non-cacheable read operation. 3. For cache inhibited cases, see the cache inhibited column. 4. For upgrade_state, a read_failed response is returned with a Resp_code of xlate_pending or rty_req while the host is attempting to obtain Write authority. If only RO authority is granted, the host informs the device using xlate_done and a Resp_code of adr_error. For upgrade_state.t, a read_failed is returned with a Resp_code of TA_adr_error. 5. Finding the line in the L1 when the host's address translation results in cache inhibited appears to be an error with either the L1, the host's translation mechanism, or a combination of the two. When the host's address translation results in cache inhibited, synonyms might not be detected by the host implementation. Regardless of the host implementation's examination of the L2, the host implementation shall leave the L2 state unchanged. When the host does not examine the L2, the resulting response shall be upgrade_resp with a cache_state of I. 						

Approved

Table 2-21. **synonym_detected** formation and actions Cache state field values and final host proxy states due to starting states and address translation (Page 3 of 7)

Command	Starting L2 state	Response	Address translation indicated access writes: Entry shows: Response cache_state (final L2 cache state)			Comments
			Write Authority	RO Authority	cache inhibited	
read_s read_s.t	I	cl_rd_resp	S (S) ³	S (S) ³	I (I)	
	S	sd	S (S) ³	S (S) ³	I (S) ²	
	E	sd	S (E) ³	S (E) ³	I (E) ²	
	E _i	sd	S (E _i) ³	S (E _i) ³	I (E _i) ²	The AFU is being directed to go to a state with valid data, with a response where no data is supplied. The AFU may have created data for the line. That is, the AFU _{C2} may be holding the line in an E state with the data marked as dirty, or may be in an M state. If the line is held in an E _i state by the AFU _{C2} L1 cache, it shall evict the line from the AFU _{C2} . That is, the host tag must be freed in order for the host to be able to send data in response to the command. The AFU _{C2} may then retry the operation until it obtains the data requested.
	M	sd	S (M) ³	S (M) ³	I (M) ²	The AFU _{C2} holds the data in a dirty state and shall issue a castout.push when invalidating the host tag's use. The AFU _{C2} shall track the clean/dirty state of the cache block associated with all valid host tags.
<ol style="list-style-type: none"> For read_me, a read_failed response is returned with a Resp_code of xlate_pending or rty_req while the host is attempting to obtain Write authority. If only RO authority is granted, the host informs the device using xlate_done and a Resp_code of adr_error. For read_me.t, a read_failed is returned with a Resp_code of TA_adr_error. Synonyms might not be detected by the host implementation when the {EA, address_context} address translation results in a storage attribute of cache inhibited. A host implementation might not examine the L2 and shall leave the L2 state unchanged. When the host does not examine the L2, the resulting response shall be cl_rd_resp with a cache state of I once the host has completed its non-cacheable read operation. For cache inhibited cases, see the cache inhibited column. For upgrade_state, a read_failed response is returned with a Resp_code of xlate_pending or rty_req while the host is attempting to obtain Write authority. If only RO authority is granted, the host informs the device using xlate_done and a Resp_code of adr_error. For upgrade_state.t, a read_failed is returned with a Resp_code of TA_adr_error. Finding the line in the L1 when the host's address translation results in cache inhibited appears to be an error with either the L1, the host's translation mechanism, or a combination of the two. When the host's address translation results in cache inhibited, synonyms might not be detected by the host implementation. Regardless of the host implementation's examination of the L2, the host implementation shall leave the L2 state unchanged. When the host does not examine the L2, the resulting response shall be upgrade_resp with a cache_state of I. 						

Approved

Table 2-21. **synonym_detected** formation and actions Cache state field values and final host proxy states due to starting states and address translation (Page 4 of 7)

Command	Starting L2 state	Response	Address translation indicated access writes: Entry shows: Response cache_state (final L2 cache state)			Comments
			Write Authority	RO Authority	cache inhibited	
upgrade_state upgrade_state.t (I → M) <i>(continue on next page)</i>	I	upgrade_resp	M (M) ³	X (I) ⁴	I (I)	M (M): Data cache set to all 0. The host shall not update the POC to an all 0 state since the host does not know if the operation has completed in the AFU. On completion of the operation, the AFU _{C2} holds the data in a dirty state and shall issue a castout.push when invalidating the host tag's use. The AFU _{C2} shall track the clean/dirty state of the cache block associated with all valid host tags.
	S	sd	M (E) ³	X (S) ⁴	I (S) ⁵	M (E): Data cache set to all 0. The host shall not update the POC to an all 0 state since the host does not know if the operation has completed in the AFU. On completion of the operation, the AFU _{C2} holds the data in a dirty state and shall issue a castout.push when invalidating the host tag's use. The AFU _{C2} shall track the clean/dirty state of the cache block associated with all valid host tags.
	E	sd	M (E) ³	X (E) ⁴	I (E) ⁵	M (E): Data cache set to all 0. The host shall not update the POC to an all 0 state since the host does not know if the operation has completed in the AFU. On completion of the operation, the AFU _{C2} holds the data in a dirty state and shall issue a castout.push when invalidating the host tag's use. The AFU _{C2} shall track the clean/dirty state of the cache block associated with all valid host tags.
	E _I	sd	M (E) ³	X (E _I) ⁴	I (E _I) ⁵	M (E): Data cache set to all 0. Even though the data cache state started as invalid, the final data state is known based on the specification of the operation. The host shall not update the POC to an all 0 state since the host does not know if the operation has completed in the AFU. On completion of the operation, the AFU _{C2} holds the data in a dirty state and shall issue a castout.push when invalidating the host tag's use. The AFU _{C2} shall track the clean/dirty state of the cache block associated with all valid host tags.
<ol style="list-style-type: none"> For read_me, a read_failed response is returned with a Resp_code of xlate_pending or rty_req while the host is attempting to obtain Write authority. If only RO authority is granted, the host informs the device using xlate_done and a Resp_code of adr_error. For read_me.t, a read_failed is returned with a Resp_code of TA_adr_error. Synonyms might not be detected by the host implementation when the {EA, address_context} address translation results in a storage attribute of cache inhibited. A host implementation might not examine the L2 and shall leave the L2 state unchanged. When the host does not examine the L2, the resulting response shall be cl_rd_resp with a cache state of I once the host has completed its non-cacheable read operation. For cache inhibited cases, see the cache inhibited column. For upgrade_state, a read_failed response is returned with a Resp_code of xlate_pending or rty_req while the host is attempting to obtain Write authority. If only RO authority is granted, the host informs the device using xlate_done and a Resp_code of adr_error. For upgrade_state.t, a read_failed is returned with a Resp_code of TA_adr_error. Finding the line in the L1 when the host's address translation results in cache inhibited appears to be an error with either the L1, the host's translation mechanism, or a combination of the two. When the host's address translation results in cache inhibited, synonyms might not be detected by the host implementation. Regardless of the host implementation's examination of the L2, the host implementation shall leave the L2 state unchanged. When the host does not examine the L2, the resulting response shall be upgrade_resp with a cache_state of I. 						

Approved

Table 2-21. **synonym_detected** formation and actions Cache state field values and final host proxy states due to starting states and address translation (Page 5 of 7)

Command	Starting L2 state	Response	Address translation indicated access writes: Entry shows: Response cache_state (final L2 cache state)			Comments
			Write Authority	RO Authority	cache inhibited	
(continued from previous page) upgrade_state upgrade_state.t (I → M)	M	sd	M (M) ³	X (M) ⁴	I (M) ⁵	M (M): Data cache set to all 0. The host shall not update the POC to an all 0 state since the host does not know if the operation has completed in the AFU. On completion of the operation, the AFU _{C2} holds the data in a dirty state and shall issue a castout.push when invalidating the host tag's use. The AFU _{C2} shall track the clean/dirty state of the cache block associated with all valid host tags.
upgrade_state upgrade_state.t (I → E _I) (continue on next page)	I	upgrade_resp	E _I (E _I) ³	X (I) ⁴	I (I)	E _I (E _I): The host shall ensure that the data held in the POC has been updated to the current coherent data state.
	S	sd	E _I (E _I) ³	X (S) ⁴	I (S) ⁵	E _I (E _I): Data cache is unchanged. Using an unspecified implementation method, pointers to the valid data shall be retained by all existing synonym entries in the AFU _{C2} L1 cache. Using an unspecified implementation method, synonym entries in the AFU _{C2} EA L1 cache holding the line in an S state can continue to access the data. Synonym entries in the AFU _{C2} L1 cache holding the line in an E _I state shall view the data cache for the cache line to be invalid. Alternatively an implementation shall invalidate all other synonym entries in the AFU _{C2} EA L1 cache that are held in an S state and mark the data cache as invalid. The host shall ensure that the data held in the POC has been updated to the current coherent data state. The AFU _{C2} shall track the clean/dirty state of the cache block associated with all valid host tags.
<ol style="list-style-type: none"> 1. For read_me, a read_failed response is returned with a Resp_code of xlate_pending or rty_req while the host is attempting to obtain Write authority. If only RO authority is granted, the host informs the device using xlate_done and a Resp_code of adr_error. For read_me.t, a read_failed is returned with a Resp_code of TA_adr_error. 2. Synonyms might not be detected by the host implementation when the {EA, address_context} address translation results in a storage attribute of cache inhibited. A host implementation might not examine the L2 and shall leave the L2 state unchanged. When the host does not examine the L2, the resulting response shall be cl_rd_resp with a cache state of I once the host has completed its non-cacheable read operation. 3. For cache inhibited cases, see the cache inhibited column. 4. For upgrade_state, a read_failed response is returned with a Resp_code of xlate_pending or rty_req while the host is attempting to obtain Write authority. If only RO authority is granted, the host informs the device using xlate_done and a Resp_code of adr_error. For upgrade_state.t, a read_failed is returned with a Resp_code of TA_adr_error. 5. Finding the line in the L1 when the host's address translation results in cache inhibited appears to be an error with either the L1, the host's translation mechanism, or a combination of the two. When the host's address translation results in cache inhibited, synonyms might not be detected by the host implementation. Regardless of the host implementation's examination of the L2, the host implementation shall leave the L2 state unchanged. When the host does not examine the L2, the resulting response shall be upgrade_resp with a cache_state of I. 						

Approved

Table 2-21. **synonym_detected** formation and actions Cache state field values and final host proxy states due to starting states and address translation (Page 6 of 7)

Command	Starting L2 state	Response	Address translation indicated access writes: Entry shows: Response cache_state (final L2 cache state)			Comments
			Write Authority	RO Authority	cache inhibited	
(continued from previous page) upgrade_state upgrade_state.t (I → E ₁) (continue on next page)	E	sd	E ₁ (E) ³	X (E) ⁴	I (E) ⁵	<p>E₁ (E): Data cache is unchanged and might be marked dirty by the AFU_{C2}. Using an unspecified implementation method, pointers to the valid data shall be retained by all existing synonym entries in the AFU_{C2} L1 cache.</p> <p>Using an unspecified implementation method, synonym entries in the AFU_{C2} EA L1 cache holding the line in any valid-data-cache-state can continue to access the data. Synonym entries in the AFU_{C2} EA L1 cache holding the line in an E₁ state shall view the data cache for the cache line to be invalid.</p> <p>Alternatively an implementation shall invalidate all other synonym entries in the AFU_{C2} EA L1 cache that are held in any valid-data-cache-state and mark the data cache as invalid. When the cache block is marked as dirty, the AFU_{C2} shall clean the dirty state of the line, by issuing a castout.push with a final state indicating E₁. That is, the host tag is not invalidated by this action.</p> <p>The AFU_{C2} shall track the clean/dirty state of the cache block associated with all valid host tags.</p>
<ol style="list-style-type: none"> 1. For read_me, a read_failed response is returned with a Resp_code of xlate_pending or rty_req while the host is attempting to obtain Write authority. If only RO authority is granted, the host informs the device using xlate_done and a Resp_code of adr_error. For read_me.t, a read_failed is returned with a Resp_code of TA_adr_error. 2. Synonyms might not be detected by the host implementation when the {EA, address_context} address translation results in a storage attribute of cache inhibited. A host implementation might not examine the L2 and shall leave the L2 state unchanged. When the host does not examine the L2, the resulting response shall be cl_rd_resp with a cache state of I once the host has completed its non-cacheable read operation. 3. For cache inhibited cases, see the cache inhibited column. 4. For upgrade_state, a read_failed response is returned with a Resp_code of xlate_pending or rty_req while the host is attempting to obtain Write authority. If only RO authority is granted, the host informs the device using xlate_done and a Resp_code of adr_error. For upgrade_state.t, a read_failed is returned with a Resp_code of TA_adr_error. 5. Finding the line in the L1 when the host's address translation results in cache inhibited appears to be an error with either the L1, the host's translation mechanism, or a combination of the two. When the host's address translation results in cache inhibited, synonyms might not be detected by the host implementation. Regardless of the host implementation's examination of the L2, the host implementation shall leave the L2 state unchanged. When the host does not examine the L2, the resulting response shall be upgrade_resp with a cache_state of I. 						

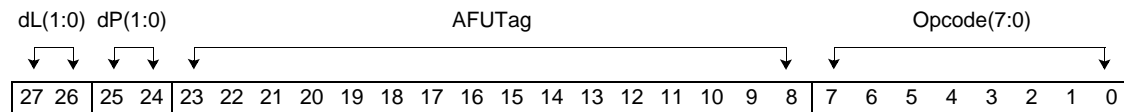
Approved

Table 2-21. **synonym_detected** formation and actions Cache state field values and final host proxy states due to starting states and address translation (Page 7 of 7)

Command	Starting L2 state	Response	Address translation indicated access writes: Entry shows: Response cache_state (final L2 cache state)			Comments
			Write Authority	RO Authority	cache inhibited	
(continued from previous page) upgrade_state upgrade_state.t (I → E _I)	E _I	sd	E _I (E _I) ³	X (E _I) ⁴	I (E _I) ⁵	E _I (E _I): Data cache is unchanged and might be marked dirty by the AFU _{C2} . Using an unspecified implementation method, pointers to the valid data shall be retained by all existing synonym entries in the AFU _{C2} L1 cache. Using an unspecified implementation method, synonym entries in the AFU _{C2} L1 cache holding the line in any valid-data-cache-state can continue to access the data. Synonym entries in the AFU _{C2} L1 cache holding the line in an E _I state shall view the data cache for the cache line to be invalid. Alternatively an implementation shall invalidate all other synonym entries in the AFU _{C2} L1 cache that are held in any valid-data-cache-state and mark the data cache as invalid. When the cache block is marked as dirty, the AFU _{C2} shall clean the dirty state of the line, by issuing a castout.push with a final state indicating E _I . That is, the host tag is not invalidated by this action. The AFU _{C2} shall track the clean/dirty state of the cache block associated with all valid host tags.
	M	sd	E _I (M) ³	X (M) ⁴	I (M) ⁵	E _I (M): Data cache is unchanged and is dirty. Using an unspecified implementation method, pointers to the valid data shall be retained by all existing synonym entries in the AFU _{C2} L1 cache. Using an unspecified implementation method, synonym entries in the AFU _{C2} L1 cache holding the line in any valid-data-cache-state can continue to access the data. Synonym entries in the AFU _{C2} L1 cache holding the line in an E _I state shall view the data cache for the cache line to be invalid. Alternatively an implementation shall invalidate all other synonym entries in the AFU _{C2} L1 cache that are held in any valid-data-cache-state and mark the data cache as invalid. Since the cache block is marked as dirty, the AFU _{C2} shall clean the dirty state of the line, by issuing a castout.push with a final state indicating E _I . That is, the host tag is not invalidated by this action. The AFU _{C2} shall track the clean/dirty state of the cache block associated with all valid host tags.
<ol style="list-style-type: none"> For read_me, a read_failed response is returned with a Resp_code of xlate_pending or rty_req while the host is attempting to obtain Write authority. If only RO authority is granted, the host informs the device using xlate_done and a Resp_code of adr_error. For read_me.t, a read_failed is returned with a Resp_code of TA_adr_error. Synonyms might not be detected by the host implementation when the {EA, address_context} address translation results in a storage attribute of cache inhibited. A host implementation might not examine the L2 and shall leave the L2 state unchanged. When the host does not examine the L2, the resulting response shall be cl_rd_resp with a cache state of I once the host has completed its non-cacheable read operation. For cache inhibited cases, see the cache inhibited column. For upgrade_state, a read_failed response is returned with a Resp_code of xlate_pending or rty_req while the host is attempting to obtain Write authority. If only RO authority is granted, the host informs the device using xlate_done and a Resp_code of adr_error. For upgrade_state.t, a read_failed is returned with a Resp_code of TA_adr_error. Finding the line in the L1 when the host's address translation results in cache inhibited appears to be an error with either the L1, the host's translation mechanism, or a combination of the two. When the host's address translation results in cache inhibited, synonyms might not be detected by the host implementation. Regardless of the host implementation's examination of the L2, the host implementation shall leave the L2 state unchanged. When the host does not examine the L2, the resulting response shall be upgrade_resp with a cache_state of I. 						

Approved

Read response	read_response	'0000 0100'
Read data return	TL.vc.0, TL.dcp.0	1



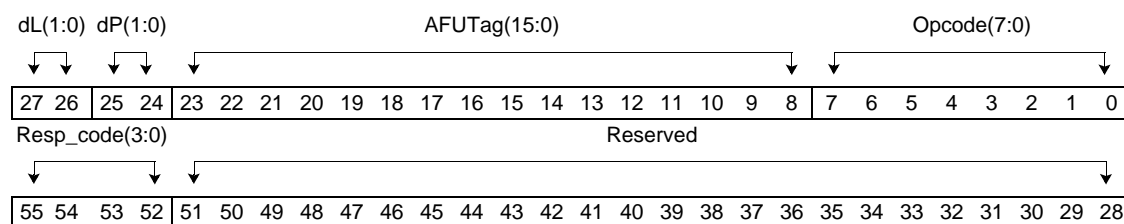
In response to a non-cacheable read command initiated by the AFU, the host is returning data. The data may be returned in a data flit, or may be returned in multiple 32-byte data fields carried in control flits. The AFU can determine which command to associate the data with by using the AFUtag provided with the command and returned with the response.

The dLength (dL) field indicates the amount of data contained in this response. Multiple read response packets may be received for a single read command. When the dLength field in the response does not match the full amount of data requested by the command, the dPart (dP) field is used to indicate the offset within the *naturally aligned data block* specified by the address in the read command. For multiple responses to a single command, the AFUtag is unchanged. That is, the dLength may vary and the dPart shall vary when multiple responses are returned for a single command. For multiple responses to a single command, there is no order requirement placed by the architecture. That is, the TLX may see the values of dPart returned in any order. When multiple responses are received for a read command, a combination of **read_response**, **read_response.ow**, and **read_failed** responses may be received. Taken together, all responses shall contain a combination of dPart and implied length or dLength to cover the command's dLength specification.

The dLength and dPart fields shall be specified as 64 bytes and offset at 0 for **pr_rd_wntc** and any of the commands classified as **mem_atomics** that return data. A single response covers the entire operation. Data is aligned within the data flit based on the command's address bits 5:0.

This response is specified with immediate data. Credits for both the VC and DCP shall be obtained before this response is serviced by the TL.

Read failed response	read_failed	'0000 0101'
Read data return	TL.vc.0	2



In response to a read command initiated by the AFU, the host is indicating that the read failed. The AFU determines which command to associate the failure with by using the AFUtag provided with the command and returned with the response.

The dLength and dPart fields specify how much of the read operation is being reported. Multiple read response packets may be received for a single read command. When the dLength field in the response does not match the full amount of data requested by the command, the dPart field is used to indicate the offset within the *naturally aligned data block* specified by the address in the read command. For multiple responses

Approved

to a single command, the AFUtag is unchanged. That is, the dLength may vary and the dPart shall vary when multiple responses are returned for a single command. For multiple responses to a single command, there is no order requirement placed by the architecture. That is, the TLX may see the values of dPart returned in any order.

- When multiple responses are received for **rd_wnltc**, a combination of **read_response**, **read_response.ow**, **synonym_detected**, and **read_failed** responses may be received. Taken together, all responses shall contain a combination of dPart and implied length or dLength to cover the command's dLength specification.
- When multiple responses are received for **read_me**, **read_mes**, or **read_s** and their dot-t variants, a combination of **cl_rd_resp**, **cl_rd_resp.ow**, **synonym_detected**, and **read_failed** responses may be received. Taken together, all responses shall contain a combination of dPart and implied length or dLength to cover the command's dLength specification.
- When multiple responses are received for **upgrade_state** or **upgrade_state.t**, a combination of **upgrade_resp**, **synonym_detected**, and **read_failed** responses may be received. Taken together, all responses shall contain a combination of dPart and implied length or dLength to cover the command's dLength specification.

The dLength and dPart fields shall be specified as 64 bytes and offset at 0 for **pr_rd_wnltc**, **amo_rd**, **amo_rw**, and all dot variants of these commands. A single response shall be returned for these commands.

The Resp_code field indicates the type of failure being reported. The Resp_code field is specified in *Table 2-22*.

*Table 2-22. The Resp_code specification for **read_failed** (Page 1 of 2)*

Resp_code encode	Description
'0000' - '0001'	Reserved.
'0010'	Retry request (rty_req). Use of this code point might be due to an event in the host that may require software intervention, or may indicate that address translation could not be completed for the command at this time, or may be due to a hardware recovery mechanism that exceeds the programmability of the devices long back off event timer. The operation may be retried by the device. This is a long back-off event.
'0011'	Light-weight retry request (lw_rty_req). Use of this code point might be due to the lack of hardware resources in the host to service the command, or a transient error condition that the hardware is able to clear out on its own. The retry back off timer is specified in the devices configuration space and should be set to a value that allows the hardware recovery mechanism to complete before the device attempts to retry the operation. The operation may be retried by the device. This is a short back-off event.
'0100'	Translation pending (xlate_pending). Indicates that the address translation could not be completed. The ATC did not contain the translation, and software was invoked. An asynchronous xlate_done TL command shall be sent when software actions have completed. It is strongly recommended that an AFU wait for the xlate_done before retrying the command. However, using a retry back off timer is permitted to determine when to retry the command. Such an implementation shall examine xlate_done for the results of the address translation and take action based on those results. <ul style="list-style-type: none"> • xlate_done uses TL.vc.0. The implementation shall ensure that the read_failed carrying the response code of xlate_pending is added to the VC prior to the xlate_done.
'0101'	Reserved
'0110'	Reserved.
'0111'	Reserved.
Note: The errors specified by Resp_code do not include the fatal error conditions described in <i>Table 7-1</i> on page 199.	

Approved

Table 2-22. The Resp_code specification for **read_failed** (Page 2 of 2)

Resp_code encode	Description
'1000'	<p>Data error (dError). The host's protocol stack operation has completed. The data obtained by the host has been corrupted and is not correctable. This may be a recoverable error by retrying the operation. See the device documentation and <i>Section 2.1.1</i> for additional information.</p> <p>Engineering note — A dError condition may also be reported using cl_rd_resp, cl_rd_resp.ow, read_response, read_response.ow, or read_response.xw, as appropriate for the TLX command, and shall indicate that the data is bad using the bad data indication in the control flit as specified for the data carrier used.</p>
'1001'	Unsupported operand length. The operation specifies an operand length that is not supported by the device. A retry of the operation shall not be successful.
'1010'	Reserved.
'1011'	Bad address specification. The address specified, an EA, TA or PA, by the command is not naturally aligned on a boundary specified by the operand length. Additional restrictions for address specification are specified in the operation descriptions of the TLX command amo_rd on page 88. A retry of the operation shall not be successful.
'1100'	For dot-t commands only. The {TA, address context} is not recognized. A retry of the operation shall not be successful.
'1101'	Reserved.
'1110'	<p>Failed. The operation has failed and cannot be recovered. This code point indicates that the state of the host due to the error occurrence does not allow a successful retry of the operation.</p> <p>Engineering Note — It is strongly recommended that an implementation provide error collection facilities to indicate the reason for the Resp_code = Failed. The specification of the error collection facility should be documented in the host's platform architecture.</p>
'1111'	TA_adr_error. For dot-t commands only. The {TA, address context} specified by the command does not have write permission as required by the command (upgrade_state.t and amo_rw.t). A retry of the operation shall not be successful.

Note: The errors specified by Resp_code do not include the fatal error conditions described in *Table 7-1* on page 199.

read_failed responds to the TLX commands found in *Table 2-23*. For each command only the Resp_codes indicated with a Y may be used. Resp_code indicated with an N shall not be used.

Table 2-23. **read_failed** Resp_code use by TLX command (Page 1 of 2)

TLX command	rtv_req (2)	lw_rtv_req (3)	xlate_pending (4)	dError (8)	Unsupported operand length (9)	Bad address specification (11)	TA not recognized (12)	Failed (14)	TA_adr_error (15)
rd_wntc	Y	Y	Y	Y	N	N	N	Y	N
pr_rd_wntc	Y	Y	Y	Y	N	Y	N	Y	N
rd_wntc.n	Y	Y	Y	Y	N	N	N	Y	N
pr_rd_wntc	Y	Y	Y	Y	N	Y	N	Y	N

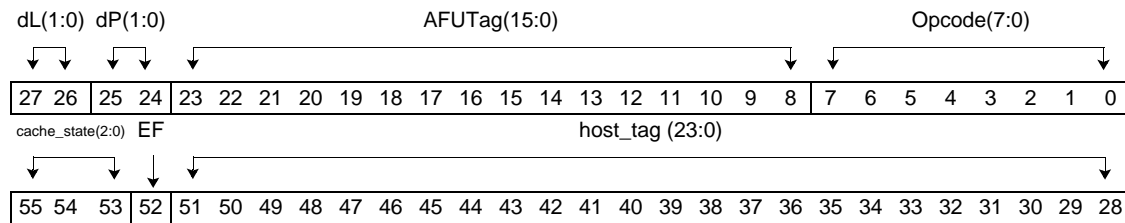
Approved

Table 2-23. **read_failed** Resp_code use by TLX command (Page 2 of 2)

TLX command	rty_req (2)	lw_rty_req (3)	xlate_pending (4)	dError (8)	Unsupported operand length (9)	Bad address specification (11)	TA not recognized (12)	Failed (14)	TA_adr_error (15)
amo_rd	Y	Y	Y	Y	N	Y	N	Y	N
amo_rd.n	Y	Y	Y	Y	N	Y	N	Y	N
amo_rw	Y	Y	Y	Y	N	Y	N	Y	N
amo_rw.n	Y	Y	Y	Y	N	Y	N	Y	N
mem_pa_flush	N	Y	N	N	N	Y	N	Y	N
upgrade_state	Y	Y	Y	N	N	Y	N	Y	N
read_me	Y	Y	Y	Y	N	N	N	Y	N
read_mes	Y	Y	Y	Y	N	N	N	Y	N
read_s	Y	Y	Y	Y	N	N	N	Y	N
rd_wntc.t	N	Y	N	Y	N	N	Y	Y	N
rd_wntc.t.s	N	Y	N	Y	N	N	Y	Y	N
pr_rd_wntc.t	N	Y	N	Y	N	Y	Y	Y	N
pr_rd_wntc.t.s	N	Y	N	Y	N	Y	Y	Y	N
amo_rd.t	N	Y	N	Y	Y	Y	Y	Y	N
amo_rd.t.s	N	Y	N	Y	Y	Y	Y	Y	N
amo_rw.t	N	Y	N	Y	Y	Y	Y	Y	Y
amo_rw.t.s	N	Y	N	Y	Y	Y	Y	Y	Y
upgrade_state.t	N	Y	N	N	N	Y	Y	Y	Y
read_me.t	N	Y	N	Y	N	Y	Y	Y	Y
read_mes.t	N	Y	N	Y	N	Y	Y	Y	N
read_s.t	N	Y	N	Y	N	Y	Y	Y	N

Approved

cacheable read response	cl_rd_resp	'0000 0110'
Read data return	TL.vc.0, TL.dcp.0	2



In response to a cacheable read from the AFU, the Host is returning data and the state of the line (*cache_state*). The data may be returned in a data flit, or may be returned in multiple 32-byte data carriers. The AFU can determine which command to associate the data with by using the AFUTag provided with the command and is returned with the response.

The *dLength* field indicates the amount of data contained in this response. Multiple read response packets may be received for a single read command. When the *dLength* field in the response does not match the full amount of data requested by the command, the *dPart* field is used to indicate the offset within the *naturally aligned data block* specified by the address in the read command. For multiple responses to a single command, the AFUTag, and *cache_state* is unchanged. That is, the *dLength*, *dPart*, *EF* and *host_tag* may vary when multiple responses are returned for a single command. For multiple responses to a single command, there is no order requirement placed by the architecture. That is, the TLX may see the values of *dPart* returned in any order. When multiple responses are received for a cacheable read command, a combination of **cl_rd_resp**, **cl_rd_resp.ow**, **synonym_detected**, and **read_failed** responses may be received. Taken together, all responses shall contain a combination of implied length, *dLength* and *dPart* to cover the command's *dLength* specification.

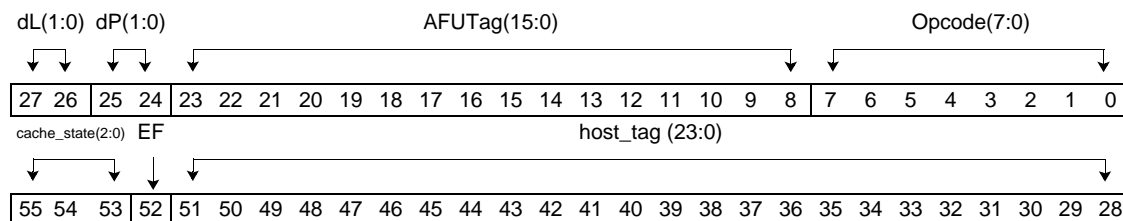
The *cache_state* indicates the state the cache line has been granted to the AFU. When a cache state of *I* is indicated by this field, the *host_tag* field is reserved and the data shall not be retained in the AFU's cache.

The evict and fill directive (*EF*) indicates if the *host_tag* is being re-assigned. When asserted, the entry in the L1 cache indicated by the current state of the *host_tag database host_tag entry* is to be evicted from the L1 cache before installing the new data and cache state specified by this response. See *Section A.10 Host tag locking transactions* on page 235 for transaction examples.

This response is specified with immediate data; Credits for both the VC and DCP shall be obtained before this response is serviced by the TL. The association between the immediate data and the *host_tag* field is found in *Section 5.1.3.1* beginning on page 186.

Approved

Upgrade response	upgrade_resp	'0000 0111'
Read data return	TL.vc.0	2



In response to an **upgrade_state** or **upgrade_state.t** command, the Host provides the current state of the line (*cache_state*) after an upgrade request. No data is returned.

The cache state specification is limited to the I, M, or E_I encodes as specified in *cache_state* on page 49.

A cache state of E_I or M indicates that address translation determined that write permissions are granted has a page attribute of not-cache-inhibited.

A cache state of I indicates that address translation has determined that write permissions are granted and has a page attribute of cache-inhibited.

Engineering note

When **upgrade_resp** returns with a cache state of I, the AFU shall treat this response as an error condition and shall take any corrective actions necessary, including killing the running application. The method used by the AFU to report or correct this condition is beyond the scope of this architecture.

Unlike a **read_me** read request, for example, there is no data returned to the AFU that could be examined and operated on. The cache state of I does not allow an update to the AFU_{C2} cache.

The *dLength* field indicates the number of 64 byte segments of the requested upgrade specified by this response. Multiple upgrade response packets may be received for a single upgrade request command. *dPart* indicates the offset from the starting address specified in the upgrade request command.

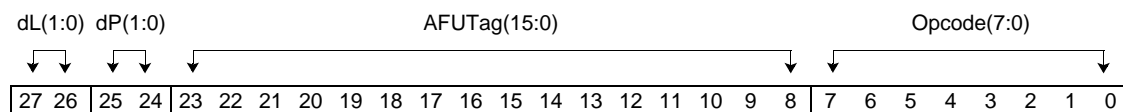
The evict and fill directive (EF) indicates if the *host_tag* is being re-assigned. When asserted, the entry in the L1 cache indicated by the current state of the *host_tag database host_tag entry* is to be evicted from the L1 cache before installing the new data and cache state specified by this response. See *Section A.10 Host tag locking transactions* on page 235 for transaction examples.

The *dLength* and *dPart* fields specify how much of the **upgrade_state** operation is being reported. Multiple **upgrade_resp** response packets may be received for a single **upgrade_state** command. When the *dLength* field in the response does not match the full amount of data requested by the **upgrade_state** command, the *dPart* field is used to indicate the offset within the *naturally aligned data block* specified by the address in the **upgrade_state** command. For multiple responses to a single command, the AFUTag is unchanged. That is, the *dLength* may vary and the *dPart* shall vary when multiple responses are returned for a single command. For multiple responses to a single command, there is no order requirement placed by the architecture. That is, the TLX may see the values of *dPart* returned in any order.

Approved

When multiple responses are received for **upgrade_state** or **upgrade_state.t**, a combination of **upgrade_resp**, **synonym_detected**, and **read_failed** responses may be received. Taken together, all responses shall contain a combination of dPart and implied length or dLength to cover the command's dLength specification.

Write response	write_response	'0000 1000'
write response	TL.vc.0	1



This packet is used in response to a non-posted write command (that is, **dma_w**, **dma_w.be**, **dma_pr_w**, **amo_w**) operation that has succeeded. The AFU determines which command to associate with this response by using the AFUTag provided with the command and returned with the response. Data specified by this response is global visible. That is, a subsequent read shall see the new data.

For **dma_w**, the dLength (dL) and dPart (dP) fields specify how much of the write operation is being reported. A single response may cover the entire operation. For a single response, the dLength must match the dLength specified by the command, and dPart must indicate a starting offset of 0. Multiple write response packets may be received for a single write command. When the dLength field in the response does not match the full amount of data requested by the command, the dPart field is used to indicate the offset from the starting address specified in the write command. For multiple responses to a single command, the AFUTag is unchanged. That is, the dLength may vary and the dPart shall vary when multiple responses are returned for a single command. For multiple responses to a single command, there is no order requirement placed by the architecture. That is, the TLX may see the values of dPart returned in any order. When multiple responses are received for a write command, a combination of **write_response** and **write_failed** responses may be received. Taken together, all responses shall contain a combination of dLength and dPart to cover the command's dLength specification.

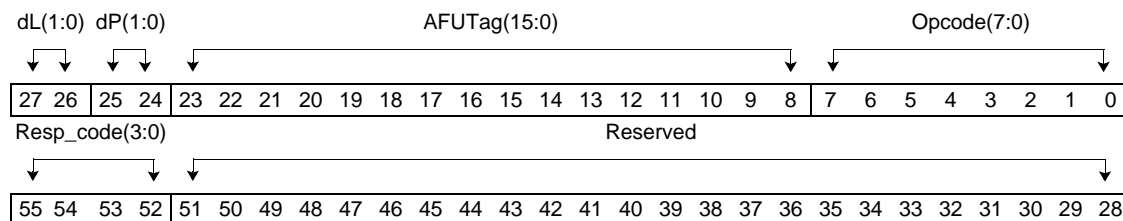
For **dma_pr_w** and **amo_w** commands, only one response is expected; dLength and dPart shall be specified as 64 bytes and offset at 0.

Engineering note

A **write_response** is used in response to a **dma_w** regardless of how the data was transported to the TLX. The response is on a multiple of 64-byte blocks even when the data might have been moved using 32-byte data fields in control flits. The AFU shall gather the completion of the 32-byte transfers and report the results as if the transfers occurred in 64-byte address-aligned data flits.

Approved

Write failed response	write_failed	'0000 1001'
write response	TL.vc.0	2



In response to a write command initiated by the AFU, the host is indicating that the write failed. The AFU determines which command to associate the failure with by using the AFUTag provided with the command and returned with the response.

The dLength and dPart fields specify how much of the write operation is being reported. A single response may cover the entire operation. For a single response, the dLength must match the dLength specified by the command, and dPart must indicate a starting offset of 0. Multiple write response packets may be received for a single write command. When the dLength field in the response does not match the full amount of data requested by the command, the dPart field is used to indicate the offset from the starting address specified in the write command. For multiple responses to a single command, the AFUTag is unchanged. That is, the dLength may vary and the dPart shall vary when multiple responses are returned for a single command. For multiple responses to a single command, there is no order requirement placed by the architecture. That is, the TLX may see the values of dPart returned in any order. When multiple responses are received for a write command, a combination of **write_response** and **write_failed** responses may be received. Taken together, all responses shall contain a combination of dLength and dPart to cover the command's dLength specification.

This response shall be returned when the operation fails and the write command is non-posted.

For **dma_w.be**, **dma_pr_w**, **amo_w**, and all dot variants of these commands, only one response shall be returned; dLength and dPart shall be specified as 64 bytes and offset at 0.

The Resp_code field indicates the type of failure being reported. The Resp_code field is specified in *Table 2-24*.

Table 2-24. The Resp_code specification of **write_failed** (Page 1 of 2)

Resp_code encode	Description
'0000' - '0001'	Reserved.
'0010'	Retry request (rty_req). Use of this code point might be due to an event in the host that may require software intervention, or may indicate that address translation could not be completed at this time, or may be due to a hardware recovery mechanism that exceeds the programmability of the device's long back off timer specified in the device's configuration space. The operation may be retried by the device. This is a long back-off event.
'0011'	Light-weight retry request (lw_rty_req). Due to hardware congestion or other non-software event, the host is unable to process the command at this time. This is a short back-off event. Use of this code point might be due to the lack of hardware resources in the host to service the command, or a transient error condition that the hardware is able to clear out on its own. The retry back off time is specified in the device's configuration space and should be set to a value that allows the hardware recovery mechanism to complete before the device attempts to retry the operation. The operation may be retried by the device.
'0100'	Translation pending (xlate_pending). Indicates that the address translation could not be completed. The ATC did not contain the translation, and software was invoked. An asynchronous xlate_done TL command shall be sent when software actions have completed. It is strongly recommended that an AFU wait for the xlate_done before retrying the command. However, using a retry back off timer is permitted to determine when to retry the command. Such an implementation shall examine xlate_done for the results of the address translation and take action based on those results. <ul style="list-style-type: none"> xlate_done uses TL.vc.0. The implementation shall ensure that the write_failed carrying the response code of xlate_pending is added to the VC prior to the xlate_done.
'0101' - '0110'	Reserved.
'0111'	Reserved.
'1000'	Data error (dError). The host's protocol stack operation completed. The received data was UE data, or might have been marked bad in the control flit associated with the data transfer, or might have been damaged in the host. Changes, if any, to the memory location specified by the response are globally visible. The memory location shall contain SUE data. <div> <p>Engineering note</p> <p>If an implementation is unable to modify the memory location specified by the command to contain SUE data, the implementation shall not report a dError. The implementation shall report a Failed.</p> </div> <div> <p>Engineering note</p> <p>A dError condition may also be reported by the consumer of the data. That is, the reporting of the dError condition may be delayed until the data is consumed by a read operation. This requires that the actions taken when the error condition is detected either shall cause the memory location to contain SUE data or shall use an alternate method to report the data is invalid prior to or when it is consumed. When either of these methods are used, the host may response with write_response instead of a write_failed.</p> </div>
'1001'	Unsupported operand length. The operation specifies an operand length that is not supported by the device. A retry of the operation shall not be successful.
'1010'	Reserved.
'1011'	Bad address specification. The address specified is not naturally aligned on a boundary specified by the operand length. A retry of the operation shall not be successful.
'1100'	Reserved. <div> <p>Developer Note</p> <p>{TA, address_context} error is not reported for this response since all dot-t write commands are posted. Instead this condition is treated as a fatal error. See <i>Posted command error on page 204</i>.</p> </div>

Note: The errors specified by Resp_code do not include the fatal error conditions described in Table 7-1 on page 199.

Approved

Table 2-24. The Resp_code specification of **write_failed** (Page 2 of 2)

Resp_code encode	Description
'1101'	Reserved.
'1110'	<p>Failed. The operation has failed and cannot be recovered. This code point indicates that the state of the host due to the error occurrence does not allow a successful retry of the operation. This includes the following:</p> <ul style="list-style-type: none"> A dError event was detected and the implementation is unable to modify the memory location specified by the command to contain SUE data. Changes, if any to the memory location specified by the response are globally visible. The memory location may be unmodified, or may contain undefined data. Any other failure detected by the host that is not included in any of the specified response codes. The failure may cause the modification of the memory location specified by the command. Changes, if any to the memory location specified by the response are globally visible. The memory location may be unmodified, may contain undefined data, or may contain SUE data. <p>Engineering Note</p> <p>It is strongly recommended that an implementation provide error collection facilities to indicate the reason for the Resp_code = Failed. The specification of the error collection facility should be documented in the host's platform architecture.</p>
'1111'	<p>Reserved.</p> <p>Developer Note</p> <p>{TA, address_context} specified by this command does not have the necessary write permission. This error is not reported using this code point since all dot-t write commands are posted. Instead this condition is treated as a fatal error See <i>Posted command error on page 204</i>.</p>
<p>Note: The errors specified by Resp_code do not include the fatal error conditions described in <i>Table 7-1</i> on page 199.</p>	

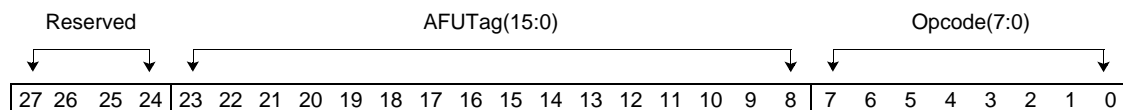
write_failed responds to the TLX commands found in *Table 2-25*. For each command only the Resp_codes indicated with a Y may be used. Resp_code indicated with an N shall not be used.

Table 2-25. **write_failed** Resp_code use by TLX command

TLX command	rty_req (2)	lw_rty_req (3)	xlate_pending (4)	dError (8)	Unsupported operand length (9)	Bad address specification (11)	Failed (14)
dma_w	Y	Y	Y	Y	N	Y	Y
dma_w.n	Y	Y	Y	Y	N	Y	Y
dma_w.be	Y	Y	Y	Y	N	N	Y
dma_w.be.n	Y	Y	Y	Y	N	N	Y
dma_pr_w	Y	Y	Y	Y	N	Y	Y
dma_pr_w.n	Y	Y	Y	Y	N	Y	Y
amo_w	Y	Y	Y	Y	Y	Y	Y
amo_w.n	Y	Y	Y	Y	Y	Y	Y

Approved

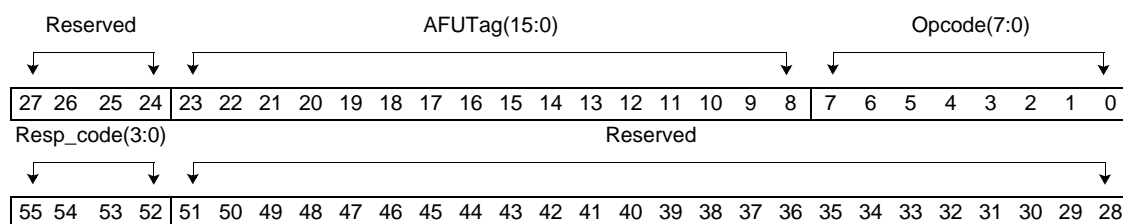
Sync barrier done	sync_done	'0000 1011'
barrier	TL.vc.0	1



In response to a **sync** command. All commands prior to the **sync** command have completed in the TLX.vc.3 service queue. This response is sent once the **sync** command has been removed from the head of the service queue and is placed after all the responses of prior commands of the service queue.

See *TL Virtual channel and service queues* on page 168 and *TL Presync queues* on page 171.

Interrupt response	intrp_resp	'0000 1100'
message response	TL.vc.0	2



This packet is used in response to **intrp_req**, **intrp_req.s**, **intrp_req.d**, and **intrp_req.d.s** commands.

The response code indicates that the interrupt was successfully initiated or provides error status. The Resp_code field is specified in *Table 2-26*.

Table 2-26. The Resp_code specification for intrp_resp

Resp_code encode	Description (Page 1 of 2)
'0000'	Interrupt request accepted.
'0001'	Reserved.
'0010'	Retry request (rty_req). Use of this code point might be due to an event in the host that may require software intervention, or may indicate that address translation could not be completed at this time, or may be due to a hardware recovery mechanism that exceeds the programmability of the device's long back off timer specified in the device's configuration space. The operation may be retried by the device. This is a long back-off event.
'0011'	Light-weight retry request (lw_rty_req). Use of this code point might be due to the lack of hardware resources in the host to service the command, or a transient error condition that the hardware is able to clear out on its own. The retry back off timer is specified in the devices configuration space and should be set to a value that allows the hardware recovery mechanism to complete before the device attempts to retry the operation. The operation may be retried by the device. This is a short back-off event.

Note: The errors specified by Resp_code do not include the fatal error conditions described in *Table 7-1* on page 199.

Approved

Table 2-26. The *Resp_code* specification for *intrp_resp*

Resp_code encode	Description (Page 2 of 2)
'0100'	Interrupt resources pending (<i>intrp_pending</i>). Indicates that the operation could not be completed at this time requiring additional software intervention. Software intervention has been successfully invoked. An asynchronous <i>intrp_rdy</i> TL command shall be sent when software actions have completed and the operation can be retried. It is strongly recommended that an AFU wait for the <i>intrp_rdy</i> before retrying the command. However, using a retry back off timer is permitted to determine when to retry the command. Such an implementation shall examine <i>intrp_rdy</i> for the results and take action based on those results. <ul style="list-style-type: none"> <i>intrp_rdy</i> uses TL.vc.0. The implementation shall ensure that the <i>intrp_resp</i> carrying the response code of <i>intrp_pending</i> is added to the VC prior to the <i>intrp_rdy</i>.
'0101' - '0110'	Reserved.
'0111'	Reserved.
'1000'	Data error (dError). Used only in response to <i>intrp_req.d</i> . The received data was corrupted and not correctable, or might have been marked bad in the control flit associated with the data transfer, or might have been damaged in the host. The operation is aborted.
'1001'	Unsupported operand length. Used only in response to <i>intrp_req.d</i> . The operation specifies an operand length that is not supported by the device. A retry of the operation shall not be successful.
'1010'	Reserved.
'1011'	Bad object handle specification. The object handle is specified by the platform architecture. A retry of the operation shall not be successful.
'1100'	Reserved.
'1101'	Reserved.
'1110'	Failed. The operation has failed and cannot be recovered. This code point indicates that the state of the host due to the error occurrence does not allow a successful retry of the operation. <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <p style="text-align: center;">Engineering Note</p> <p>It is strongly recommended that an implementation provide error collection facilities to indicate the reason for the <i>Resp_code</i> = Failed. The specification of the error collection facility should be documented in the host's platform architecture.</p> </div>
'1111'	Reserved.

Note: The errors specified by *Resp_code* do not include the fatal error conditions described in *Table 7-1* on page 199.

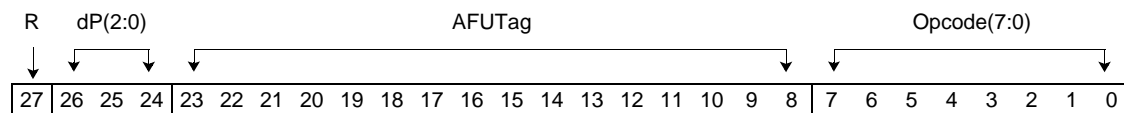
intrp_resp responds to the TLX commands found in *Table 2-27*. For each command only the *Resp_codes* indicated with a Y may be used. *Resp_code* indicated with an N shall not be used.

Table 2-27. *intrp_resp* *Resp_code* use by TLX command

TLX command	rty_req (2)	lw_rty_req (3)	xlate_pending (4)	dError (8)	Unsupported operand length (9)	Bad address specification (11)	Failed (14)
<i>intrp_req</i>	Y	Y	Y	N	N	Y	Y
<i>intrp_req.d</i>	Y	Y	Y	Y	Y	Y	Y

Approved

Read response	read_response.ow	'0000 1101'
Read data return	TL.vc.0, TL.dcp.0	1



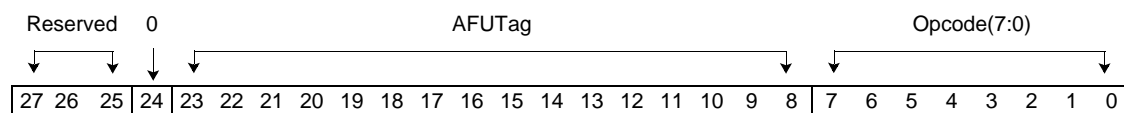
In response to a non-cacheable read command initiated by the AFU, the host is returning data using a 32-byte data field carried in control flits. The AFU can determine which command to associate the data with by using the AFUTag provided with the command and returned with the response.

The response implies a data length of 32 bytes. Multiple read response packets may be received for a single read command. The dPart (dP) indicates the offset within the *naturally aligned data block* specified by the address in the read command. For multiple responses to a single command, the AFUTag is unchanged. That is, the dPart shall vary when multiple responses are returned for a single command. For multiple responses to a single command, there is no order requirement placed by the architecture. That is, the TLX may see the values of dPart returned in any order. When multiple responses are received for a read command, a combination of **read_response**, **read_response.ow**, and **read_failed** responses may be received. Taken together, all responses shall contain a combination of dPart and implied lengths or dLength to cover the command's dLength specification.

The dPart field shall be specified as offset at 0 for **pr_rd_wntc** and any of the commands classified as **mem_atomics** that return data. A single response covers the entire operation. Data is aligned within the 32-byte data carrier based on the command's address bits 4:0.

This response is specified with immediate data. Credits for both the VC and DCP shall be obtained before this response is serviced by the TL.

Read response	read_response.xw	'0000 1110'
Read data return	TL.vc.0, TL.dcp.0	1



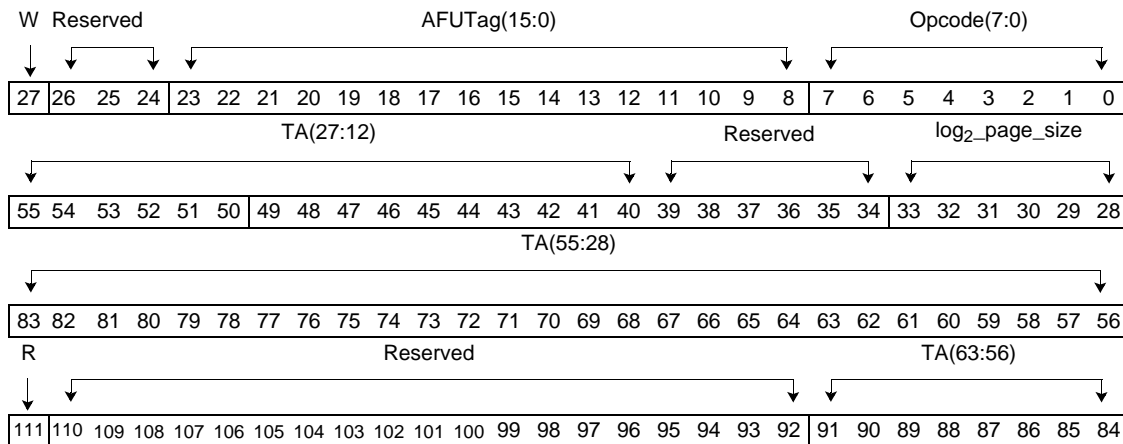
In response to a non-cacheable read command initiated by the AFU, the host is returning data using an 8-byte data field carried in a control flit. The AFU can determine which command to associate the data with by using the AFUTag provided with the command and returned with the response.

The response implies a data length of 8 bytes. The full amount of the data requested by the command is returned.

This response is specified with immediate data. Credits for both the VC and DCP shall be obtained before this response is serviced by the TL.

Approved

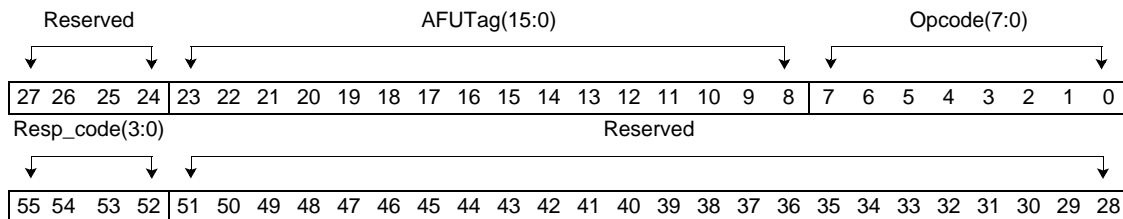
touch response with TA	touch_resp.t	'0000 1111'
address translation management	TL.vc.0	4



This is one of two possible responses to an **xlata_touch** command. This response provides a page size aligned translated address (TA), *log₂_page_size*, and write permission (*W*)

touch_resp is used when a translated address has not been requested, or when the Resp_code is not indicating complete.

Wake Host Thread Response	wake_host_resp	'0001 0000'
message response	TL.vc.0	2



This packet is used in response to a **wake_host_thread** or **wake_host_thread.s** command. The operation in the host was either successful in waking the thread specified by the command or it was not. The Resp_code field and the reporting priority when multiple errors are detected is specified in *Table 2-28*.

Approved

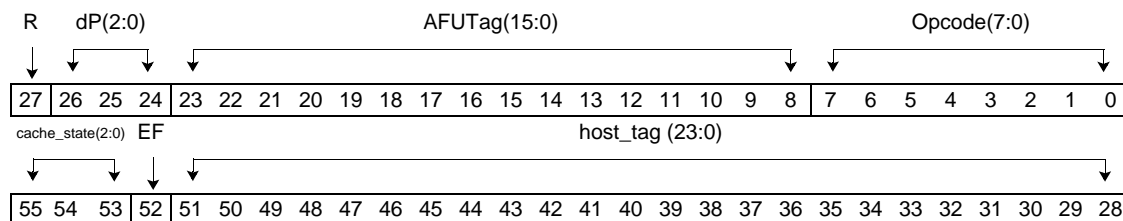
Table 2-28. The Resp_code specification for *wake_host_resp*

Resp_code encode	Description
'0000'	Thread found. Thread woken.
'0001'	Reserved.
'0010'	Retry request (rty_req). Indicates that the operation could not be completed at this time. The operation may be retried at a later time. This is a long <i>back-off event</i> .
'0011'	Light-weight retry request (lw_rty_req). Use of this code point might be due to the lack of hardware resources in the host to service the command, or a transient error condition that the hardware is able to clear out on its own. The retry back off timer is specified in the devices configuration space and should be set to a value that allows the hardware recovery mechanism to complete before the device attempts to retry the operation. The operation may be retried by the device. This is a short back-off event.
'0100'	Interrupt resources pending. (intrp_pending). Indicates that the operation could not be completed at this time requiring additional software intervention. Software intervention has been successfully invoked. An asynchronous intrp_rdy TL command will be sent when software actions have completed and the operation can be retried. It is strongly recommended that an AFU wait for the intrp_rdy before retrying the command. However, using a retry back off timer is permitted to determine when to retry the command. Such an implementation shall examine intrp_rdy for the results and take action based on those results. <ul style="list-style-type: none"> intrp_rdy uses TL.vc.0. The implementation shall ensure that the wake_host_resp carrying the response code of intrp_pending is added to the VC prior to the intrp_rdy.
'0101'	Thread not found. An interrupt is required to service the operation.
'0110'	Reserved.
'0111'	Reserved.
'1000' - '1010'	Reserved.
'1011'	Bad object handle specification. The object handle is specified by the platform architecture. A retry of the operation shall not be successful.
'1100'	Reserved.
'1101'	Reserved.
'1110'	Failed. The operation has failed and cannot be recovered. This code point indicates that the state of the host due to the error occurrence does not allow a successful retry of the operation. <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <p style="text-align: center;">Engineering Note</p> <p>It is strongly recommended that an implementation provide error collection facilities to indicate the reason for the Resp_code = Failed. The specification of the error collection facility should be documented in the host's platform architecture.</p> </div>
'1111'	Reserved.

Note: The errors specified by Resp_code do not include the fatal error conditions described in Table 7-1 on page 199.

Approved

cacheable read response	cl_rd_resp.ow	'0001 0110'
Read data return	TL.vc.0, TL.dcp.0	2



In response to a cacheable read from the AFU, the Host is returning data using a 32-byte data field carried in control flits and the state of the line (cache_state). The AFU can determine which command to associate the data with by using the AFUTag provided with the command and is returned with the response.

The response implies a data length of 32-bytes. Multiple read response packets may be received for a single read command. The dPart (dP(2:0)) indicates the offset within the *naturally aligned data block* specified by the address in the read command. For multiple responses to a single command, the AFUTag and cache_state is unchanged. That is, the dPart, host_tag and EF fields may vary when multiple responses are returned for a single command. For multiple responses to a single command, there is no order requirement placed by the architecture. That is, the TLX may see the values of dPart returned in any order. When multiple responses are received for a cacheable read command, a combination of **cl_rd_resp**, **cl_rd_resp.ow**, **synonym_detected**, and **read_failed** responses may be received. Taken together, all responses shall contain a combination of implied lengths, dLength and dPart to cover the command's dLength specification.

The cache_state indicates the state the cache line has been granted to the AFU. When a cache state of I is indicated by this field, the data shall not be retained in the AFU's cache.

The evict and fill directive (EF) indicates if the host_tag is being re-assigned. When asserted, the entry in the L1 cache indicated by the current state of the *host_tag database host_tag entry* is to be evicted from the L1 cache before installing the new data and cache state specified by this response. See *Section A.10 Host tag locking transactions* on page 235 for transaction examples.

This response is specified with immediate data; Credits for both the VC and DCP shall be obtained before this response is serviced by the TL. The association between the immediate data and the host_tag field is found in *Section 5.1.3.1* beginning on page 186.

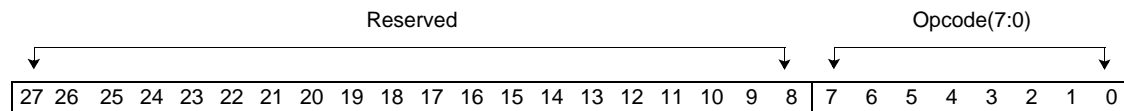
Approved

2.5 TLX AP response packets

TLX responses are sent from the AFU to the host. An alphabetical list of the TLX responses follows; each response is hyperlinked to its specification. In this section, the TLX response specifications are in opcode order.

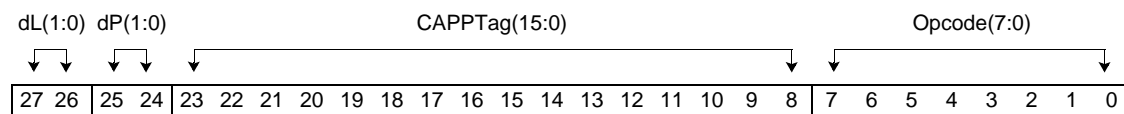
atc_disabled **atc_enabled** **cache_disabled** **cache_enabled**
kill_xlate_done **mem_cntl_done** **mem_rd_fail**
mem_rd_response **mem_rd_response.ow** **mem_rd_response.xw** **mem_wr_fail**
mem_wr_response **nop** **return_tl_credits**

No operation	nop	'0000 0000'
NA	NA	1



This response has no operands and performs no action. It is discarded at the TL.

Memory read response	mem_rd_response	'0000 0001'
mem_response	TLX.vc.0, TLX.dcp.0	1



In response to a memory read command initiated by the host, the AFU is returning data. The data may be returned in a data flit, or may be returned in multiple 32-byte data carriers. The host can determine which command to associate the data with by using the CAPPTag provided with the command and returned with the response.

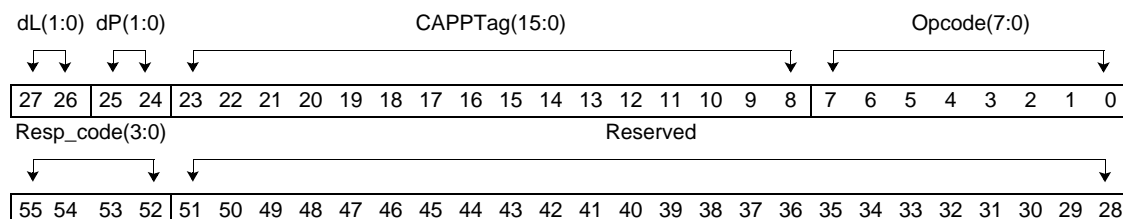
The dLength (dL) field indicates the amount of data contained in this response. Multiple response packets may be received for a single memory read command. When the dLength field in the response does not match the full amount of data specified by the command, the dPart (dP) field is used to indicate the offset within the *naturally aligned data block* specified by the address in the memory read command. For multiple responses to a single command, the CAPPTag is unchanged. That is, only the dLength and dPart fields may vary when multiple responses are returned for a single command. For multiple responses to a single command, there is no order requirement placed by the architecture. That is, the TL may see the values of dPart returned in any order. When multiple responses are received for a memory read command, a combination of **mem_rd_response**, **mem_rd_response.ow**, and **mem_rd_fail** responses may be received. Taken together, all responses shall contain a combination of dPart and implied length or dLength to cover the command's dLength specification.

For **pr_rd_mem**, **amo_rd**, **amo_rw**, and **config_read**, the dLength and dPart fields shall be specified as 64 bytes and offset at 0. A single response shall be returned.

Approved

This response is specified with immediate data. Credits for both the VC and DCP shall be obtained before this response is serviced by the TLX.

Memory read failure	mem_rd_fail	'0000 0010'
mem_response	TLX.vc.0	2



In response to a memory read command initiated by the host, the AFU is indicating that the read failed. The host determines which command to associate with the failure by using the CAPPTag provided with the command and returned with the response.

For **rd_mem**, the dLength (dL) and dPart (dP) fields specify how much of the read operation is being reported. A single response may cover the entire operation. For a single response, the dLength field must match the dLength specified by the command, and the dPart field must indicate a starting offset of 0. Multiple response packets may be received for a single memory read command. When the dLength field in the response does not match the full amount of data specified by the command, the dPart field is used to indicate the offset within the *naturally aligned data block* specified by the address in the memory read command. For multiple responses to a single command, the CAPPTag is unchanged. That is, the dLength may vary and the dPart shall vary when multiple responses are returned for a single command. For multiple responses to a single command, there is no order requirement placed by the architecture. That is, the TL may see the values of dPart returned in any order. When multiple responses are received for a memory read command, a combination of **mem_rd_response**, **mem_rd_response.ow**, and **mem_rd_fail** responses may be received. Taken together, all responses shall contain a combination of dLength and dPart to cover the command's dLength specification.

For the **pr_rd_mem** command, **amo_rd**, **amo_rw**, and **config_read**, the dLength and dPart fields shall be specified as 64 bytes and offset at 0. A single response shall be returned. Violating this rule results in a *Bad response received* error event.

The Resp_code field indicates the type of failure being reported. The Resp_code field is specified in Table 2-29.

Approved

Table 2-29. The Resp_code specification for **mem_rd_fail**

Resp_code encode	Description
'0000' - '0001'	Reserved.
'0010'	Retry request (rty_req). The read operation could not be serviced at this time. This is a long back off event. Use of this code point might be due to an event in the device that may require software intervention, or may be due to a hardware recovery mechanism that exceeds the programmability of the host's short back off event timer. The operation may be retried by the host.
'0011'	Light-weight retry request (lw_rty_req). The read operation could not be serviced at this time. This is a short back off event. Use of this code point might be due to the lack of hardware resources in the device to service the command, or a transient error condition that the hardware is able to clear out on its own. The retry back off timer is specified in the host and should be set to a value that allows the hardware recovery mechanism to complete before the host attempts to retry the operation.
'0100' - '0111'	Reserved
'1000'	Data error (dError). The memory access completed. The data obtained by the AFU has been corrupted and is not correctable. Data is not sent to the host. This may be a recoverable error by retrying the operation. See the device documentation for additional information. <div style="border: 1px solid black; padding: 5px;">Engineering note A dError condition may also be reported using mem_rd_response, mem_rd_response.ow, or mem_rd_response.xw, as appropriate for the TL command, and shall indicate that the data is bad using the bad data indication in the control flit as specified for the data carrier used.</div>
'1001'	Unsupported operand length. The operation specifies an operand length that is not supported by the device. A retry of the operation shall not be successful.
'1010'	Reserved.
'1011'	Bad address specification. The address specified is not naturally aligned on a boundary specified by the operand length. A retry of the operation shall not be successful.
'1100' - '1101'	Reserved.
'1110'	Failed. The operation has failed and cannot be retried. This code point indicates that the state of the device due to the error occurrence does not allow a successful retry of the operation. This includes the following: <ul style="list-style-type: none"> The device and function number specified in the address of a config_read is not recognized by the AFU. config_read is issued with T=1. Any other failure detected by the AFU that is not included in any of the specified response codes. <div style="border: 1px solid black; padding: 5px;">Engineering Note It is strongly recommended that an implementation provide error collection facilities to indicate the reason for the Resp_code = Failed. The specification of the error collection facility should be documented in the device documentation.</div>
'1111'	Reserved.

Note: The errors specified by Resp_code do not include the fatal error conditions described in Table 7-1 on page 199.

mem_rd_fail responds to the TL commands found in Table 2-30. For each command only the Resp_codes indicated with a Y may be used. Resp_code indicated with an N shall not be used.

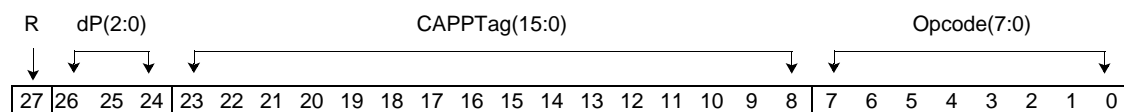
Approved

Table 2-30. *mem_rd_fail* Resp_code use by TL command

TL command	rty_req (2)	lw_rty_req (3)	dError (8)	Unsupported operand length (9)	Bad address specification (11)	Failed (14)
rd_mem	Y	Y	Y	Y ¹	N	Y
pr_rd_mem	Y	Y	Y	Y ¹	Y	Y
amo_rd	Y	Y	Y	Y ²	Y	Y
amo_rw	Y	Y	Y	Y ²	Y	Y
config_read	Y	Y	Y	Y	Y	Y

1. May occur during MMIO space read access only.
2. May occur only when the pLength field did not specify 4 or 8 bytes using the encode defined. This type of error shall not be reported as a *Reserved field value used* error.

Memory read response	mem_rd_response.ow	'0000 0011'
mem_response	TLX.vc.0, TLX.dcp.0	1



In response to a memory read command initiated by the host, the AFU is returning data using 32-byte data carriers. The host can determine which command to associate the data with by using the CAPPTag provided with the command and returned with the response.

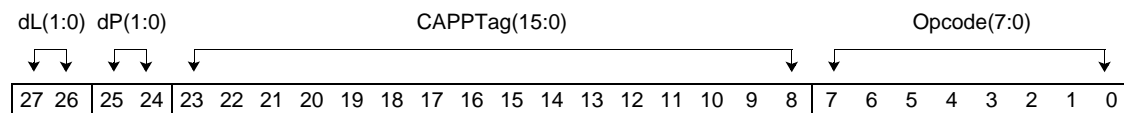
This response implies a data length of 32 bytes. Multiple response packets may be received for a single memory read command. The dPart (dP) indicates the offset within the *naturally aligned data block* specified by the address in the memory read command. For multiple responses to a single command, the CAPPTag is unchanged. That is, dPart shall vary when multiple responses are returned for a single command. For multiple responses to a single command, there is no order requirement placed by the architecture. That is, the TL may see the values of dPart returned in any order. When multiple responses are received for a memory read command, a combination of **mem_rd_response**, **mem_rd_response.ow**, and **mem_rd_fail** responses may be received. Taken together, all responses shall contain a combination of dPart and implied lengths or dLength to cover the command's dLength specification.

For **pr_rd_mem** and any of the commands classified as mem_atomics that return data, the dPart field shall be specified as an offset at 0. A single response covers the entire operation.

This response is specified with immediate data. Credits for both the VC and DCP shall be obtained before this response is serviced by the TLX.

Approved

Memory write response	mem_wr_response	'0000 0100'
mem_response	TLX.vc.0	1

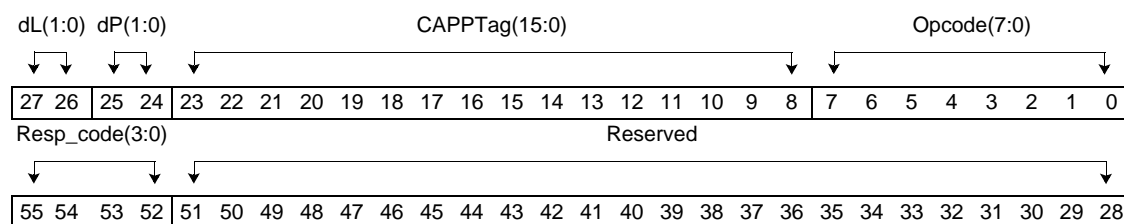


This packet is used in response to a non-posted command that writes to memory. This response is used to indicate the successful completion of all or a portion of the operation. Data specified by this response is global visible. That is, a subsequent read shall see the new data.

For **write_mem** or **pad_mem**, the dLength (dL) and dPart (dP) fields specify how much of the write operation is being reported. A single response may cover the entire operation. For a single response, the dLength must match the dLength specified by the command, and dPart must indicate a starting offset of 0. Multiple response packets may be received for a single memory write command. When the dLength field in the response does not match the full amount of data specified by the command, the dPart field is used to indicate the offset from the starting address specified in the memory write command. For multiple responses to a single command, the CAPPTag is unchanged. That is, the dLength may vary and the dPart shall vary when multiple responses are returned for a single command. For multiple responses to a single command, there is no order requirement placed by the architecture. That is, the TL may see the values of dPart returned in any order. When multiple responses are received for a memory write command, a combination of **mem_wr_response** and **mem_wr_fail** responses may be received. Taken together, all responses shall contain a combination of dLength and dPart to cover the command's dLength specification.

For the **pr_wr_mem**, **amo_w**, **write_mem.be**, and **config_write** commands a single response shall be returned. The dLength and dPart fields shall be specified as 64 bytes and offset at 0.

Memory write failed	mem_wr_fail	'0000 0101'
mem_response	TLX.vc.0	2



In response to a **write_mem** or **pad_mem** command initiated by the host, the AFU is indicating that the write failed. The host determines which command to associate with the failure by using the CAPPTag provided with the command and returned with the response.

The dLength (dL) and dPart (dP) fields specify how much of the write operation is being reported. A single response may cover the entire operation. For a single response, the dLength must match the dLength specified by the command, and dPart must indicate a starting offset of 0. Multiple response packets may be received for a single memory write command. When the dLength field in the response does not match the full amount of data specified by the command, the dPart field is used to indicate the offset from the starting address specified in the memory write command. For multiple responses to a single command, the CAPPTag is unchanged. That is, the dLength may vary and the dPart shall vary when multiple responses are returned

Approved

for a single command. For multiple responses to a single command, there is no order requirement placed by the architecture. That is, the TL may see the values of dPart returned in any order. When multiple responses are received for a memory write command, a combination of **mem_wr_response** and **mem_wr_fail** responses may be received. Taken together, all responses shall contain a combination of dLength and dPart to cover the command's dLength specification.

For **amo_w**, **config_write**, **pr_wr_mem** and **write_mem.be**, only one response shall be returned; dLength and dPart shall be specified as 64 bytes and offset at 0.

The Resp_code field indicates the type of failure being reported. The Resp_code field is specified in *Table 2-31*.

*Table 2-31. The Resp_code specification for **mem_wr_fail** (Page 1 of 2)*

Resp_code encode	Description
'0000' - '0001'	Reserved.
'0010'	Retry request (rty_req). The write operation could not be serviced at this time. This is a long back off event. Use of this code point might be due to an event in the device that may require software intervention, or may be due to a hardware recovery mechanism that exceeds the programmability of the host's short back off event timer. The operation may be retried by the host.
'0011'	Light-weight retry request (lw_rty_req). The write operation could not be serviced at this time. This is a short back off event. Use of this code point might be due to the lack of hardware resources in the device to service the command, or a transient error condition that the hardware is able to clear out on its own. The retry back off timer is specified in the host and should be set to a value that allows the hardware recovery mechanism to complete before the host attempts to retry the operation.
'0100' - '0111'	Reserved
'1000'	<p>Data error (dError). The AFU's operation has completed. The data sent by the TL was corrupted prior to the completion of the operation. Changes, if any to the memory location specified by the response are globally visible.</p> <ul style="list-style-type: none"> Memory locations specified by the command's PA and length that correspond to system memory space shall contain SUE data. Memory locations specified by the command's PA and length that correspond to either MMIO or configuration space may be unmodified, may contain undefined data, or may contain SUE data. <p>The corruption of the data might have occurred anywhere in the AFU's hardware or might have been detected by a bad data indication.</p> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>Engineering note</p> <p>If an implementation is unable to modify the memory location specified by the command's PA that correspond to system memory space to contain SUE data, the implementation shall not report a dError. The implementation shall report a Failed.</p> </div> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>Engineering note</p> <p>A dError condition may also be reported by the consumer of the data. That is, the reporting of the dError condition may be delayed until the data is consumed by a read operation. This requires that the actions taken when the error condition is detected either shall cause the memory location to contain SUE data or shall use an alternate method to report the data is invalid prior to or when it is consumed. When either of these methods are used, the AFU may response with mem_wr_response, instead of a mem_wr_fail.</p> </div>
'1001'	Unsupported operand length. The operation specifies an operand length that is not supported by the device. A retry of the operation shall not be successful.
'1010'	Reserved.
'1011'	Bad address specification. The address specified is not naturally aligned on a boundary specified by the operand length. A retry of the operation shall not be successful.
Note: The errors specified by Resp_code do not include the fatal error conditions described in <i>Table 7-1</i> on page 199.	

Approved

Table 2-31. The *Resp_code* specification for **mem_wr_fail** (Page 2 of 2)

Resp_code encode	Description
'1100 - '1101'	Reserved.
'1110'	<p>Failed. The operation has failed and cannot be retried. This code point indicates that the state of the device due to the error occurrence does not allow a successful retry of the operation. This includes the following:</p> <ul style="list-style-type: none"> The device and function number specified in the address of a config_write is not recognized by the AFU. config_write specified with T=1. A dError event was detected and the implementation is unable to modify the memory locations specified by the command's PA and length that correspond to system memory space to contain SUE data. Changes, if any to the memory location specified by the response are globally visible. The memory location may be unmodified, or may contain undefined data. Any other failure detected by the AFU that is not included in any of the specified response codes. The failure may cause the modification of the memory location specified by the command. Changes, if any to the memory location specified by the response are globally visible. The memory location may be unmodified, may contain undefined data, or may contain SUE data. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p style="text-align: center;">Engineering Note</p> <p>It is strongly recommended that an implementation provide error collection facilities to indicate the reason for the <i>Resp_code</i> = Failed. The specification of the error collection facility should be documented in the device documentation.</p> </div>
'1111'	Reserved.

Note: The errors specified by *Resp_code* do not include the fatal error conditions described in *Table 7-1* on page 199.

mem_wr_fail responds to the TL commands found in *Table 2-32*. For each command only the *Resp_codes* indicated with a Y may be used. *Resp_code* indicated with an N shall not be used.

Table 2-32. **mem_wr_fail** *Resp_code* use by TL command

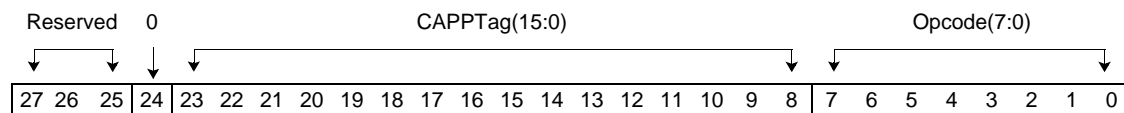
TL command	rty_req (2)	lw_rty_req (3)	dError (8)	Unsupported operand length (9)	Bad address specification (11)	Failed (14)
amo_w	Y	Y	Y	Y	Y	Y
pad_mem	Y	Y	N	Y ²	Y	Y
write_mem	Y	Y	Y	Y ²	Y	Y
write_mem.be	Y	Y	Y	N	Y	Y ²
pr_wr_mem	Y	Y	Y	Y ¹	Y	Y
config_write	Y	Y	Y	Y	Y	Y

1. Unsupported operand length may occur only when target memory is defined as MMIO space and the command's specified pLength is not supported at the address specified.

2. May occur during MMIO address space write operation only.

Approved

Memory read response	mem_rd_response.xw	'0000 0111'
mem_response	TLX.vc.0, TLX.dcp.0	1

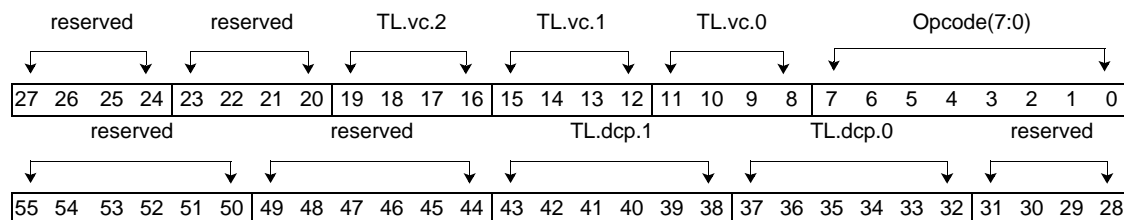


In response to a memory read command initiated by the host, the AFU is returning data using 8-byte data fields found in some control flits. The host can determine which command to associate the data with by using the CAPPTag provided with the command and returned with the response.

This response implies a data length of 8 bytes. A single response packet shall be received for a memory read operation when results are returned using this response packet. The full amount of data specified by the command is returned.

This response is specified with immediate data. Credits for both the VC and DCP shall be obtained before this response is serviced by the TLX.

Return TL credits	return_tl_credits	'0000 1000'
credit return	NA	2



This response packet is used by the TLX to return VC and DCP credits to the TL. There is no VC associated with this response, and credits are not required to service this response. Each TL.* field contains the number of credits being returned.

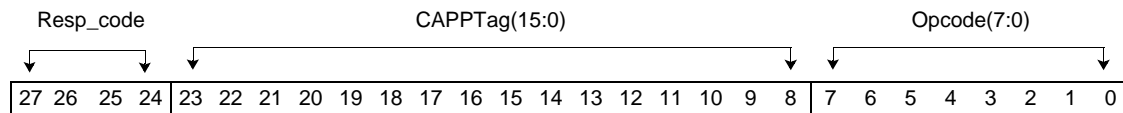
This response packet shall be placed only in slots 1 to 0 of any control flit using a template which specifies those slots as a 2-slot or larger location. The following exceptions apply:

- Control flits using template x'07' may place a **return_tl_credits** response into slots 11 to 10.
- Control flits using template x'09' may place a **return_tl_credits** response into slots 11 to 10.
- Control flits using template x'0B' may place a **return_tl_credits** response into slots 13 to 12.

TL.vc.{0, 1, 2} and TL.dcp.{0, 1} credits are returned. TL credits are for resources owned by the TLX that the TL consumes. The TLX controls the total number of credits for each of the VC and DCP it provisions the TL with.

Approved

Memory control operation done	mem_cntl_done	'0000 1011'
message	TLX.vc.0	1



In response to a **mem_cntl** command. The operation specified by the **mem_cntl** has completed as specified by the device manufacturer's specification.

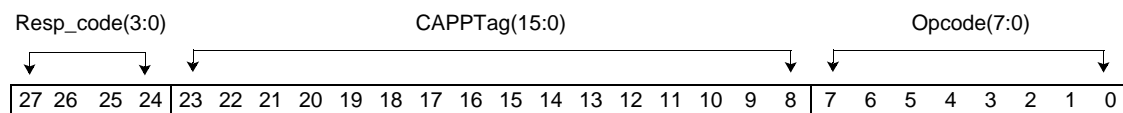
The Resp_code field specifies the type of error being reported. The Resp_code field is specified in Table 2-33.

Table 2-33. The Resp_code specification for **mem_cntl_done**

Resp_code encode	Description
'0000'	Complete. The operation has completed successfully.
'0010' - '1101'	Reserved
'1100' - '1101'	Reserved.
'1110'	Failed. The operation has failed and cannot be retried. This code point indicates that the state of the device due to the error occurrence does not allow a successful retry of the operation. <ul style="list-style-type: none"> The specification of the cmd_flag or object_handle fields in the mem_cntl command are invalid. The operation has failed and cannot be retried for any reason. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>Engineering Note</p> <p>It is strongly recommended that an implementation provide error collection facilities to indicate the reason for the Resp_code = Failed. The specification of the error collection facility should be documented in the device documentation.</p> </div>
'1111'	Reserved.

Note: The errors specified by Resp_code do not include the fatal error conditions described in Table 7-1 on page 199.

Memory check out response	kill_xlate_done	'0000 1100'
address translation management	TLX.vc.3	1



Used in response to **kill_xlate**. All commands using the address translations specified by the **kill_xlate** command shall precede the **kill_xlate_done** response in the TLX.vc.3 virtual channel. That is, this response requires that the AFU to push all commands using the translated addresses specified by the **kill_xlate** command into the VC before removing the AFU entry or entries. All commands using the specified address translations shall be dispatched to the host before the AFU ATC entry or entries have been invalidated. To accomplish this, the response shall be placed into the TLX.vc.3 virtual channel after all commands using the address translations have been committed to the TLX.vc.3 virtual channel. This response indicates to the host that the address translation is no longer in use by the AFU and has been removed from its ATC. The

Approved

host shall ensure that all uses of the address translations have completed before invalidating the AFU's use of the translated address. That is, all non-posted commands, or any clean up that is required by the host implementation with respect to the translated addresses being used by the AFU, shall completed before the addresses become unrecognizable for use-by-the-AFU by the host²⁶.

Developer note

The AFU needs to maintain the ATC entry until it has pushed all commands using it into TLX.vc.3. This includes non-posted operations such as **read_me.t**. Updates to the cache are allowed since the responses use `host_tags` and the host has ensured that the correct address translation is used when obtaining the data.

Note that once the cache has been updated and the ATC entry has been invalidated, the cache line is no longer accessible by the AFU processor element since the address translation has been invalidated.

Special queuing and servicing of **kill_xlate_done** is specified in *Section 3.4.1 TL queuing and service of kill_xlate_done* on page 173.

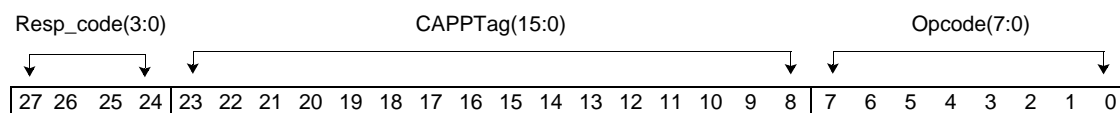
The `Resp_code` field is specified in *Table 2-34*.

*Table 2-34. The `Resp_code` specification for **kill_xlate_done***

Resp_code encode	Description
'0000'	Completed.
'0001' - '1101'	Reserved.
'1110'	Failed. The operation has failed and cannot be retried. This code point indicates that the state of the device due to the error occurrence does not allow a successful retry of the operation. <div> Engineering Note It is strongly recommended that an implementation provide error collection facilities to indicate the reason for the <code>Resp_code</code> = Failed. The specification of the error collection facility should be documented in the device documentation. </div>
'1111'	Reserved.

Note: The errors specified by `Resp_code` do not include the fatal error conditions described in *Table 7-1* on page 199.

AFU _{C2} cache disabled	cache_disabled	'0000 1101'
address translation management	TLX.vc.0	1



Used in response to a **disable_cache** command. The AFU_{C2} has disabled processing element access to its cache, based on the address context specified, causing the processing element to stall waiting on responses from its cache. Cache miss actions based on the address context specified normally taken by the AFU_{C2} shall not occur until **enable_cache** is processed.

²⁶. That is, the entries in the Host's ATC are invalidated.

Approved

TLX commands associated with cache and ATC transactions may be seen after this response has been seen. These commands can either be serviced normally by the host or can be responded with a `rty_req` `Resp_code`.

Developer note

TLX Commands associated with cache transactions are **read_me**, **read_mes**, **read_s** and **upgrade_state** and their dot-t variants: These are all TLX.vc.3 commands, and since there is no order between VC, the **cache_disabled** response might pass these commands.

Failing these commands results in a TL **read_failed** response with the `Resp_code` = `rty_req`.

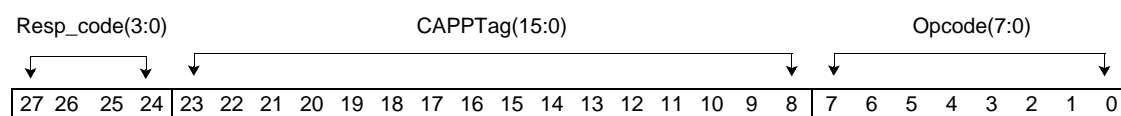
The `Resp_code` field is specified in *Table 2-35*.

*Table 2-35. The `Resp_code` specification for **cache_disabled***

Resp_code encode	Description
'0000'	Completed.
'0001' - '1101'	Reserved.
'1110'	Failed. The operation has failed and cannot be retried. This code point indicates that the state of the device due to the error occurrence does not allow a successful retry of the operation. <div> <p>Engineering Note</p> <p>It is strongly recommended that an implementation provide error collection facilities to indicate the reason for the <code>Resp_code</code> = Failed. The specification of the error collection facility should be documented in the device documentation.</p> </div>
'1111'	Reserved.

Note: The errors specified by `Resp_code` do not include the fatal error conditions described in *Table 7-1* on page 199.

AFU _{C2} cache enabled	cache_enabled	'0000 1110'
address translation management	TLX.vc.0	1



Used in response to **enable_cache** command. The AFU_{C2} has enabled processing element access to its cache based on the address context specified.

The `Resp_code` field is specified in *Table 2-36*.

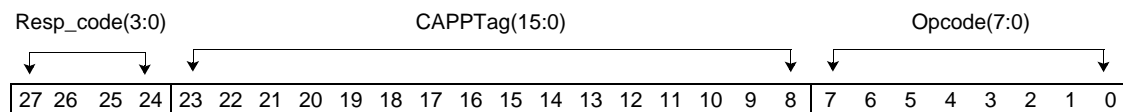
Approved

Table 2-36. The Resp_code specification for **cache_enabled**

Resp_code encode	Description
'0000'	Completed.
'0001' - '1101'	Reserved.
'1110'	Failed. The operation has failed and cannot be retried. This code point indicates that the state of the device due to the error occurrence does not allow a successful retry of the operation. <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>Engineering Note</p> <p>It is strongly recommended that an implementation provide error collection facilities to indicate the reason for the Resp_code = Failed. The specification of the error collection facility should be documented in the device documentation.</p> </div>
'1111'	Reserved.

Note: The errors specified by Resp_code do not include the fatal error conditions described in Table 7-1 on page 199.

AFU _{C2} cache disabled	atc_disabled	'1000 0000'
address translation management	TLX.vc.0	1



Used in response to a **disable_atc** command. The AFU_{C2} has disabled processing element access to its ATC, based on the address context specified, causing the processing element to stall waiting on responses from its ATC. ATC miss actions based on the address context specified normally taken by the AFU shall not occur until **enable_atc** is processed.

This response requires the AFU to push all TLX.vc.3 commands using any translated address matching the specification provided by the **disable_atc** command into the VC before responding to the **disable_atc** command. That is, all commands using translations that are being disabled shall be dispatched to the host before responding to the **disable_atc** command.

atc_disabled is assigned to TLX.vc.0 and is not in the TLX.vc.3 virtual channel, so order between the TLC.vc.3 commands and the **atc_disabled** response cannot be assured using VC ordering rules. The AFU shall issue a **sync(all_stream)** command once all the prior TLX.vc.3 commands have been pushed into the TLX.vc.3 virtual channel. Once the AFU receives a **sync_done** response for the **sync(all_stream)**, the AFU sends **atc_disabled** to complete the operation.

Once the AFU has disabled the address translations matching the specification provided by the **disable_atc** command, the AFU shall not use these translations for any purpose. Any lines held in an AFU_{C2} L1 cache using these translations shall not be updated or referenced by the AFU_{C2} processing element.

The Resp_code field is specified in Table 2-37.

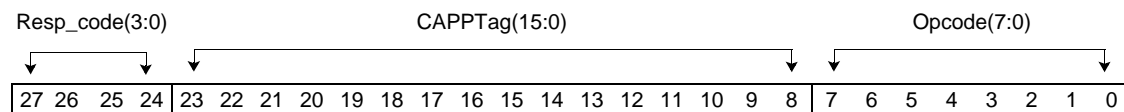
Approved

Table 2-37. The Resp_code specification for **atc_disabled**

Resp_code encode	Description
'0000'	Completed.
'0001' - '1101'	Reserved.
'1110'	Failed. The operation has failed and cannot be retried. This code point indicates that the state of the device due to the error occurrence does not allow a successful retry of the operation. <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p style="text-align: center;">Engineering Note</p> <p>It is strongly recommended that an implementation provide error collection facilities to indicate the reason for the Resp_code = Failed. The specification of the error collection facility should be documented in the device documentation.</p> </div>
'1111'	Reserved.

Note: The errors specified by Resp_code do not include the fatal error conditions described in Table 7-1 on page 199.

AFU _{C2} cache enabled	atc_enabled	'1000 0001'
address translation management	TLX.vc.0	1



Used in response to **enable_atc** command. The AFU has enabled processing element access to its ATC based on the address context specified.

The Resp_code field is specified in Table 2-38.

Table 2-38. The Resp_code specification for **atc_enabled**

Resp_code encode	Description
'0000'	Completed.
'0001' - '1101'	Reserved.
'1110'	Failed. The operation has failed and cannot be retried. This code point indicates that the state of the device due to the error occurrence does not allow a successful retry of the operation. <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p style="text-align: center;">Engineering Note</p> <p>It is strongly recommended that an implementation provide error collection facilities to indicate the reason for the Resp_code = Failed. The specification of the error collection facility should be documented in the device documentation.</p> </div>
'1111'	Reserved.

Note: The errors specified by Resp_code do not include the fatal error conditions described in Table 7-1 on page 199.

3. Virtual channel and data credit pool specification

Commands and responses are assigned to virtual channels (VCs) to allow ordering and the specification of servicing service queues and virtual queues. Credits to use these VCs are managed by the destination (where the resources are consumed) and are released to the source (where the command or response originates) when the resource is available. Each VC has its own credit pool, which may be varied by the destination. VCs in each direction are numbered; the value assigned is used only for differentiation between VC. Each VC credit permits the sending of one command or response.

The following VC descriptions use the specific command or response or the classification of the command or response. See the command and response descriptions in *Section 2 TL and TLX command and response specifications* on page 48 for command and response VC classifications.

VCs are specified between the TL and TLX and between the TLX and the TL. Ordering is maintained within a VC and is assured between these endpoints only. VCs cannot block each other; blocking within a VC may occur due to ordering requirements (head of line blocking). With the exception of *Command ordering* on page 39, any queuing or ordering occurring in the upper protocol layers (host bus and AFU) is not assured to be retained. Synchronization points are managed at the interfaces between the TL and host bus protocol stack and between the TLX and AFU protocol stack.

Data credit pool (DCP) credits are required when a command or response has immediate data; that is, data that is associated with the sending of the command or response. For example, a write command has immediate data while a read command does not. Commands and responses with immediate data shall obtain the necessary credits for both the VC and DCP assigned in an atomic fashion.

For example, sending 128 bytes requires atomically obtaining two DCP credits when using 64-byte data flits. See *Section 5.1.3 Data transport, order, and alignment* on page 184 for details on the relationship between DCP credits and *data carrier* use. If the credits are not available, the command or response cannot be serviced. That is, the command or response shall not be placed into a DL frame for transmission. See the description of *DCP* on page 19.

The order of data sent to the destination is the same as the order of the data's corresponding command or response that is sent to its destination. This allows the destination to use the arrival order of commands and responses with immediate data to associate the immediate data with its command or response.

Credits are released to the consumer of the credits by the owner of the resources that the credits represent. That is, TLX credits represent TL resources that are consumed by the TLX. TL credits represent TLX resources that are consumed by the TL. The responses used to return credits are **return_tl credits** and **return_tlx credits**. A compliant implementation shall:

- Provide a 16-bit counter to track credits available for use for each DCP and VC specified in the following sections.
- Provision a minimum of one and less than 64K credits for each VC specified in the following sections.
- Provision a minimum of four and less than 64K credits for each DCP specified in the following sections.

Developer Note

- The host is required to release credits only for the VC/DCP that will be used by the AFU. Releasing credits for VC/DCPs that are not going to be used might not be optimal because the released credits correspond to resources in the host that could have been used for actual command/data/response traffic from the AFU.
- The AFU is required to release credits only for the VC/DCP that will be used by the host. Releasing credits for VC/DCPs that are not going to be used might not be optimal because the released credits correspond to resources in the AFU that could have been used for actual command/data/response traffic from the host.

The credits required can be determined by knowing the AFU capabilities as described in *Section 1.2 Host operation modes* on page 28, the commands and responses used, and the associated VC and DCP.

Table 3-3 on page 166 specifies the assignment of VC and DCP to commands and responses.

Developer Note

The architecture permits an implementation to release a minimum-total of four DCP credits. Releasing a minimum-total of a single DCP credit was considered and discarded because this limits the data to a single 64-byte transfer. This was considered too restrictive and might not easily enable implementations to optimize for the data block size normally used by the implementation.

3.1 Virtual channel

Ordering shall be maintained between elements within a VC. Blocking between VCs shall not be permitted.

3.1.1 TLX command and response VC (TLX.vc)

The VC is directed from the AP to the host. Three VCs are specified {0, 2, 3}. VC credits are consumed by the TLX and are returned by the TL using **return_tlx_credits**.

Engineering Note

TLX VC credits represent resources in the TL used for processing TLX commands and responses. Each credit released by the TL represents a *unique* resource and shall not be shared with any other VC. Doing so would result in breaking the accounting rules implied by this specification. For example, if the TL used the same resource for two different VCs, the actual credit available for VCs using the shared resource would also be diminished, and the TLX would be unaware of this change.

The ability of the TL to return TLX VC credits shall not be dependent on any action taken by the TLX, including the return of TL credits.

3.1.2 TL command and response VC (TL.vc)

The VC is directed from the host to the AP. Three VCs are specified {0..2}. VC credits are consumed by the TL and are returned by the TLX using **return_tl_credits**.

Engineering Note

TL VC credits represent resources in the TLX used for processing TL commands and responses. Each credit released by the TLX represents a *unique* resource and shall not be shared with any other VC. Doing so would result in breaking the accounting rules implied by this specification. For example, if the TLX used the same resource for two different VCs, the actual credit available for VCs using the shared resource would also be diminished, and the TL would be unaware of this change.

The ability of the TLX to return TL VC credits shall not be dependent on any action taken by the TL, including the return of TLX credits.

3.1.3 VC credit count specification

Table 3-1 specifies the maximum number of VC credits supported by an OpenCAPI-compliant device. A device is not required to release or support the maximum count. However, the consumer of the credit shall provide a counter that supports the architected maximum count.

Table 3-1. VC maximum credit count specification

VC	Maximum credit count
TLX.vc.0	64K-1
TLX.vc.2	64K-1
TLX.vc.3	64K-1
TL.vc.0	64K-1
TL.vc.1	64K-1
TL.vc.2	64K-1

3.2 Data credit pool

3.2.1 TLX data DCP (TLX.dcp)

The data credit pool is used when moving immediate data from the AP to the host. Three DCPs are specified {0, 2, 3}. DCP credits are consumed by the TLX and returned by the TL using **return_tlx_credits**.

Engineering Note

TLX DCP credits represent resources in the TL used for accepting data from the TLX. Each credit released by the TL represents a *unique* data resource and shall not be shared with any other DCP. Doing so would result in breaking the accounting rules implied by this specification. For example, if the TL used the same resource for two different DCPs, the actual credit available for DCPs using the shared resource would also be diminished, and the TLX would be unaware of this change.

See Section 5.1.3 *Data transport, order, and alignment* on page 184 for the association between a DCP credit and the immediate data associated with the command or response.

Resources for holding any meta-data associated with the data resource are accounted for by the DCP credit associated with the data itself.

The ability of the TL to return TLX DCP credits shall not be dependent on any action taken by the TLX, including the return of TL credits.

3.2.2 TL data DCP (TL.dcp)

This data credit pool is used when moving immediate data from the host to the AP. Two DCPs are specified {0, 1}. DCP credits are consumed by the TL and are returned by using the TLX using **return_tl_credits**.

Engineering Note

TL DCP credits represent resources in the TLX used for accepting data from the TL. Each credit released by the TLX represents a *unique* data resource and shall not be shared with any other DCP. Doing so would result in breaking the accounting rules implied by this specification. For example, if the TLX used the same resource for two different DCPs, the actual credit available for DCPs using the shared resource would also be diminished, and the TL would be unaware of this change.

See *Section 5.1.3 Data transport, order, and alignment* on page 184 for the association between a DCP credit and the immediate data associated with the command or response.

Resources for holding any meta-data associated with the data resource are accounted for by the DCP credit associated with the data itself.

The ability of the TLX to return TL DCP credits shall not be dependent on any action taken by the TL, including the return of TLX credits.

3.2.3 DCP credit count specification

Table 3-2 specifies the maximum number of DCP credits supported by an OpenCAPI-compliant device. A device is not required to release or support the maximum count. However, the consumer of the credit shall provide a counter that supports the architected maximum count.

Table 3-2. DCP maximum credit count specification

DCP	Maximum credit count
TLX.dcp.0	64K-1
TLX.dcp.2	64K-1
TLX.dcp.3	64K-1
TL.dcp.0	64K-1
TL.dcp.1	64K-1

Table 3-3. Summary VC and DCP assignments (Page 1 of 2)

VC	Classification/ command	DCP	Comments	Command	Response
TL.vc.0	Touch response				X
TL.vc.0	Cacheable read (response)	TL.dcp.0			X
TL.vc.0	DMA read (response)	TL.dcp.0	TL.dcp.0 is used only for read_response and is not used for read_failed .		X
TL.vc.0	Upgrade response				X
TL.vc.0	Write response (OK and failed)				X
TL.vc.0	Interrupt and wake host thread responses		Message responses.		X

Approved

Table 3-3. Summary VC and DCP assignments (Page 2 of 2)

VC	Classification/ command	DCP	Comments	Command	Response
TL.vc.0	xlate_done		Asynchronous notification. Asynchronous command reporting the status of a previously AFU-initiated action (address translation touch).	X	
TL.vc.0	return address tag		Asynchronous notification	X	
TL.vc.0	Interrupt ready		Asynchronous notification.	X	
TL.vc.1	MEM read both multiples of 64 bytes or partial commands; includes only amo_rd atomics			X	
TL.vc.1	amo_rw and amo_w atomic operations	TL.dcp.1		X	
TL.vc.1	MEM read		An MMIO read operation uses a pr_rd_mem CAPP command.	X	
TL.vc.1	MEM write	TL.dcp.1	An MMIO write operation uses a pr_w_mem CAPP command.	X	
TL.vc.1	Wake AFU thread		message	X	
TL.vc.1	Configuration register read			X	
TL.vc.1	Configuration register write	TL.dcp.1		X	
TL.vc.2	cacheable retrieval for example, force_evict			X	
TLX.vc.0	MEM read response; for example, mem_response	TLX.dcp.0	mem_rd_fail does not use TLX.dcp.0.		X
TLX.vc.0	MEM write response				X
TLX.vc.0	Wake AFU response		Message response.		X
TLX.vc.2	Cacheable push	TLX.dcp.2	evicts M, E, S states or used to downgrade state. Data-less version does not use TLX.dcp.2	X	
TLX.vc.3	Non-cacheable read and write operations - all forms			X	
TLX.vc.3	acTag management			X	
TLX.vc.3	address Tag management			X	
TLX.vc.3	Interrupts and wake host thread	TLX.dcp.3	Message commands. TLX.dcp.3 is used only for intrp_req.d commands.	X	
TLX.vc.3	Cacheable Read		Includes upgrade_state command	X	
TLX.vc.3	xlate_touch - ATC prefetch			X	

3.3 TL Virtual channel and service queues

3.3.1 Host TLX command handling

Figure 3-1 on page 169 illustrates the steps a TLX command follows from the VC queue in the TL to its service queue. Commands are removed from the DL frame in slot order from a control flit and are loaded into the VC queue specified by the command. After the command reaches the head of the VC queue, it is examined.

1. Any error found in the command entry is noted. Errors found at this point in the flow shall not be reported until the command reaches the head of the service queue.
2. If the command is **assign_actag**, the command shall be removed from the VC queue and shall be serviced at this time. This service results in the update of the acTag table. Subsequent commands shall see the new state of the acTag table. The **assign_actag** command is removed from the head of the service queue.

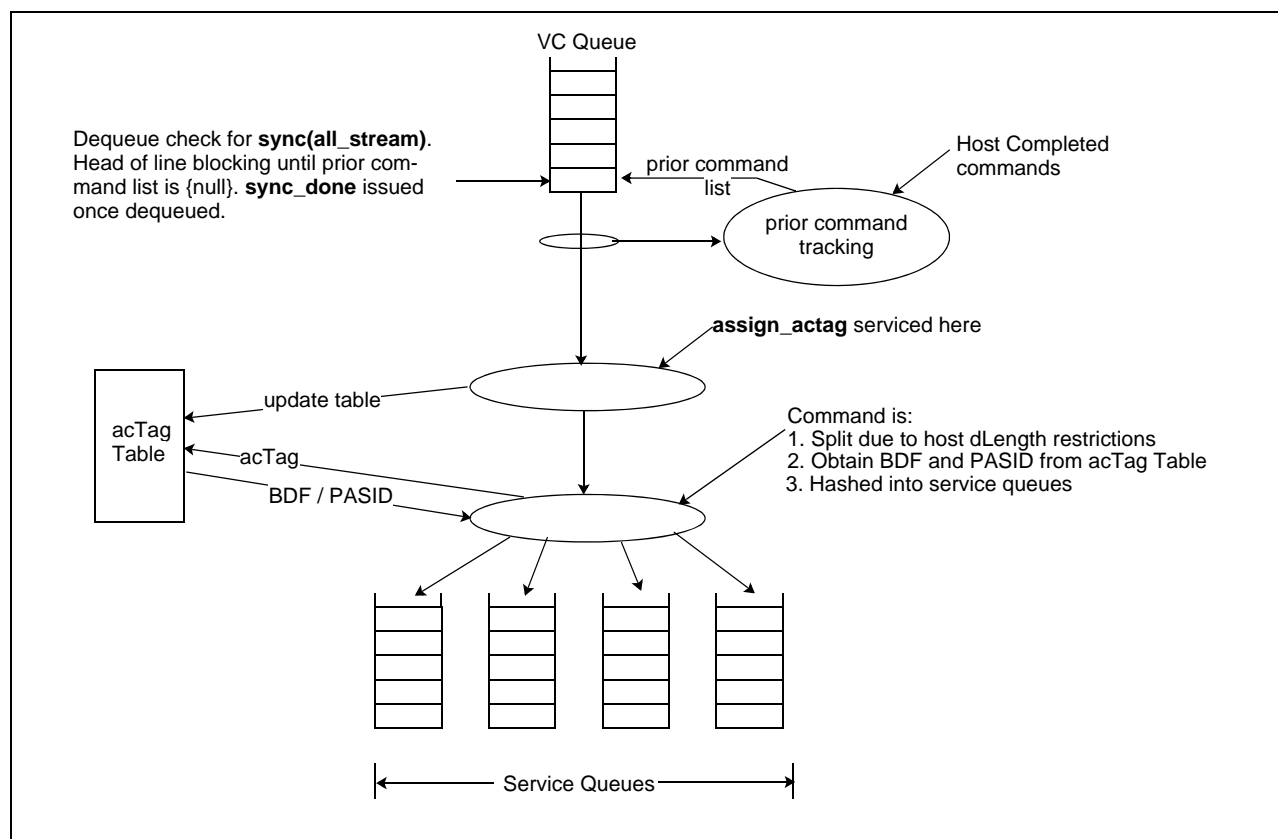
kill_xlate_done is a response that is assigned to the TLX.vc.3 queue. It is the only response assigned to that VC. All other members of TLX.vc.3 are commands. Handling of **kill_xlate_done** is discussed in Section 3.3.2 Host TLX response handling on page 170.

If the command is **sync(all_stream)**, the command remains at the head of the TLX.vc.3 queue, blocking subsequent commands, until the prior command list is {null}. See Section 3.4 for a description of the prior command tracking function. Once the all prior command list is {null} the **sync(all_stream)** command is dequeued and a **sync_done** response is sent to the AFU.

If the command is neither an **assign_actag** or **sync(all_stream)**, the command shall be serviced as follows:

- a. Commands with a dLength specified that is larger than the host's maximum supported length for a single operation are split into commands with dLength set equal to or less than the host's maximum supported length. The address is adjusted for each command to account for the dLength serviced. The AFUtag, acTag, and stream_id are obtained from the original command. Split command entries are noted with the dPart value to be used when returning responses. Split commands are serviced based on increasing address order. All commands continue with the next step.
- b. Using the acTag found in the command entry, the BDF and PASID are obtained. An error detected at this step is fatal. See the description of *Bad BDF and PASID combination on page 199* for additional details.
- c. A VC- and implementation-specific hash is used to determine the service queue to add the command to. See the specification of the hash applied to the command in the definition of a *service queue on page 23*.

The command is then added to the service queue determined in step c and removed from the head of the VC queue.

Figure 3-1. TL command flow from the VC queue to the service queue TLX VC queues shown

As defined in *Terms* on page 17, the difference between a *virtual queue* and a *service queue* is that a *service queue* may contain multiple *virtual queues*.

The architectural model of a service queue specifies that the commands in the body of the service queue may receive the following services in the following order:

1. Error checking of the command.
2. Address translation as required by the command's specification.

Commands with EA or TA undergo address translation in order to determine the RA used by the host protocol. **castout** and **castout.push** are specified with a `host_tag` and perform a look up in the host proxy directory to determine the RA used by the host protocol.

Errors occurring in either of these services shall be noted in the service queue entry and shall not be reported until the command reaches the head of the service queue. The results of the address translation shall be noted in the service queue entry.

The architectural model of a service queue specifies that the head of the service queue shall receive the following services in the following order:

1. Error checking of the command. The check may occur here, or the error noted in the entry may be used to determine the error checked state of the command.

Approved

2. Address translation as required by the command's specification. The translation may occur here, or may have previously occurred. The results of the translation noted in the entry may be used at this time.
3. When the previous two services are completed, the command shall be removed from the head of the service queue if
 - The command is not a **sync** command.
 - The command is a **sync(at_stream)** command and the prior command list is {null}. See *Section 3.4* for a description of the prior command tracking function.

Step 3 requires that a command at the head of the service queue be processed with the indicated precedence as follows:

1. When the command is specified with a dot-s attribute, the command shall be enqueued to its designated presync service queue, See *Section 3.4*.

Or

2. Failed due to an error, which may be due to either an error found with the command or failed address translation attempt.

And

- That error shall result in either a response being returned to the requester, when the command is non-posted, or an error event being asserted,

Or

3. The operation is completed²⁷ and a response shall be returned to the requester when the command is non-posted.

Or

4. Shall be dispatched to the host protocol layer.

Engineering Note

The architecture *requires* that the implementation of a service queue shall appear to behave as if the architectural model of a service queue is implemented.

3.3.2 Host TLX response handling

TLX responses are assigned to a VC and are removed from the DL frame in the same manner as TLX commands. Handling of TLX responses is simpler in that TLX.vc.0 which is used for most TLX responses has no hash involved when selecting a service queue. Responses assigned to TLX.vc.0 are passed directly to a dedicated service queue and allowed to dispatch to the host interface.

The exception to the above is **kill_xlate_done**. This response is assigned to TLX.vc.3, the same VC as read and write commands and due the nature of the protocol of invalidating ATC entries requires special handling. This is described in *Section 3.4.1 TL queuing and service of kill_xlate_done* on page 173.

27. For example, the command required only an address translation action, such as **xlate_touch**, or the address translation was not successful or is required to be retried.

The command is a **sync** command. The command is not sent to the host's protocol layer. A **sync_done** response is sent to the AFU.

The host checks that responses are expected as described in *Bad response received on page 201*.

3.4 TL Presync queues

Figure 3-2 illustrates the architectural model of the steps taken by command with dot-s attributes moving from its service queue to a presync queue before being dispatched to the host protocol layer.

For each command dequeued from the head of the service queue, the prior command function shown in *Figure 3-2* identifies all commands issued prior to the command at the head of the queue that have not completed. As a command is dispatched to the host protocol layer or to a presync queue, the command is added to a list of commands that have not completed. The content of the list of prior commands is associated with the command that is being dequeued from the head of the service queue at the time when the command is dequeued. Commands are removed from the list held in the prior command tracking function when the host protocol layer indicates a command has completed.

To illustrate the prior command tracking function, consider a case where there are 4 commands in the service queue labeled A, B, C, and D, where A is at the head of the service queue and D is at the tail.

1. At time T_0 , command A is dequeued. The prior command list is {null}.
2. At time T_1 , command B is dequeued. The prior command list is {A}.
3. At time T_2 , command C is dequeued. The prior command list is {A, B}.
4. At time T_3 , command D is dequeued. The host has reported that A has completed. The prior command list is {B, C}

Once the prior command list is added to the dot-s command's entry, it is dequeued from the head of its service queue and is enqueued into a presync queue specified by a VC- and implementation-specific hash. See the specification of the hash applied to the command in the definition of a *service queue on page 23*. The implementation may add or remove hash terms when forming the presync queue.

The architectural model of a presync queue specifies that a command shall be dequeued only from the head of the presync queue. While the command is in the presync queue, its prior command list shall be updated by the host protocol layer when it indicates the completion of a command by removing the completed command from the list. Once a command has reached the head of the presync queue, and its prior command list is {null}, the command shall be dequeued from its presync queue and shall be dispatched to the host protocol layer.

Figure 3-2. TL command flow from a service queue with a designated presync queue

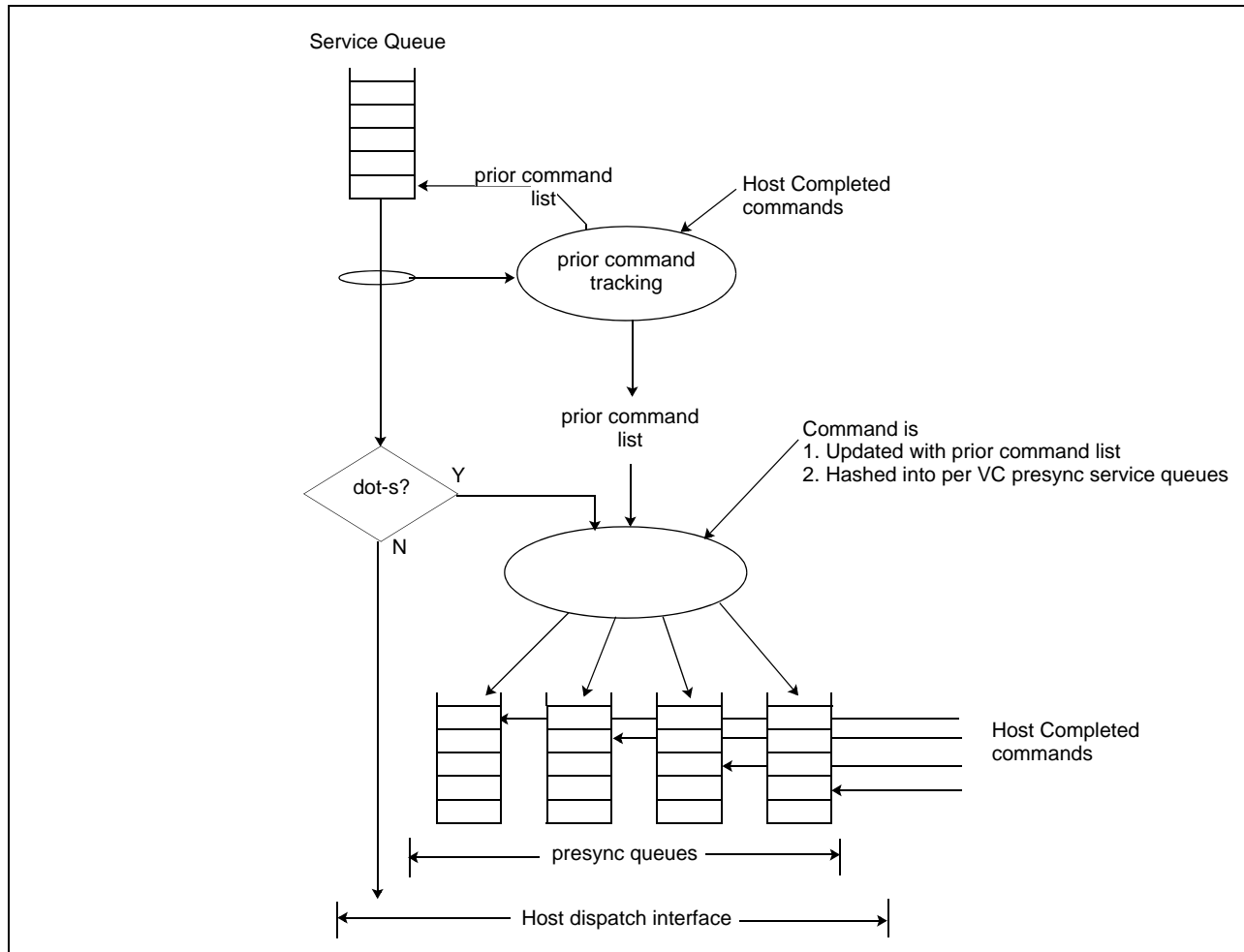


Table 3-4 provides an example of how the service queue, presync queue and dot-s attributed commands work together. The example shown has a group of commands that are allowed to complete in any order, then a presync command to ensure that the prior commands have completed, followed by another set of commands allowed to complete in any order and a final presync. The first group could be a data block write and then a flag write to indicate the first data block had been written and then a second data block and a flag to indicate that the second data block had been written.

Table 3-4. Example sequence of 2 block writes and 2 flag writes (Page 1 of 2)

Time step	Service queue head	Service queue tail	Presync queue head	Presync queue tail	host dispatch interface	host completed command
0	null	null	null	null	-	-
1	A	A	null	null	-	-
2	A	B	null	null	-	-
3	A	C.s	null	null	A	-
4	B	D	null	null	B	-

Approved

Table 3-4. Example sequence of 2 block writes and 2 flag writes (Page 2 of 2)

Time step	Service queue head	Service queue tail	Presync queue head	Presync queue tail	host dispatch interface	host completed command
5	C.s	E	C.s {A,B}	C.s {A, B}	-	-
6	D	F.s	C.s {A,B}	C.s {A, B}	D	A
7	E	F.s	C.s {B}	C.s {B}	E	-
8	F.s	F.s	C.s {B}	F.S {B, C, D, E}	-	-
9	F.s	F.s	C.s {B}	F.S {B, C, D, E}	-	-
10	null	null	C.s {B}	F.S {B, C, D, E}	-	B
11	null	null	C.s {null}	F.s {C, D, E}	C.s	-
12	null	null	F.s {C, D, E}	F.s {C, D, E}	-	D
13	null	null	F.s {C, E}	F.s {C, E}	-	-
14	null	null	F.s {C, E}	F.s {C, E}	-	E
15	null	null	F.s {C}	F.s {C}	-	-
16	null	null	F.s {C}	F.s {C}	-	-
17	null	null	F.s {C}	F.s {C}	-	C.s
18	null	null	F.s {null}	F.s {null}	-	-
19	null	null	F.s {null}	F.s {null}	F.s	-
20	null	null	null	null	-	-
21	null	null	null	null	-	-
22	null	null	null	null	-	F.s

As shown in *Table 3-4*, the first data block write consists of commands {A, B} and the flag write is in command C (noted as C.s to specify that it is issued with a presync attribute). The second block write consists of commands {D, E} and the write flag is in command F (noted as F.s).

Note that dispatching C is dependent on A and B completing, while F dispatching is dependent on A through E completing. Commands A, B, D, E are issued with no dependencies.

3.4.1 TL queuing and service of kill_xlate_done

kill_xlate_done handling is different from other TLX.vc.3 TLX packets because it is a response. Servicing **kill_xlate_done** places requirements on the host's behavior. That is, the host shall complete all previous TLX command use of the translated addresses specified by the **kill_xlate** command, to purge cache entries in an AFU_{C2} that use the specified address translations, and to clear address translations from ATC before reporting back to the host protocol that the address translation is no longer in use.

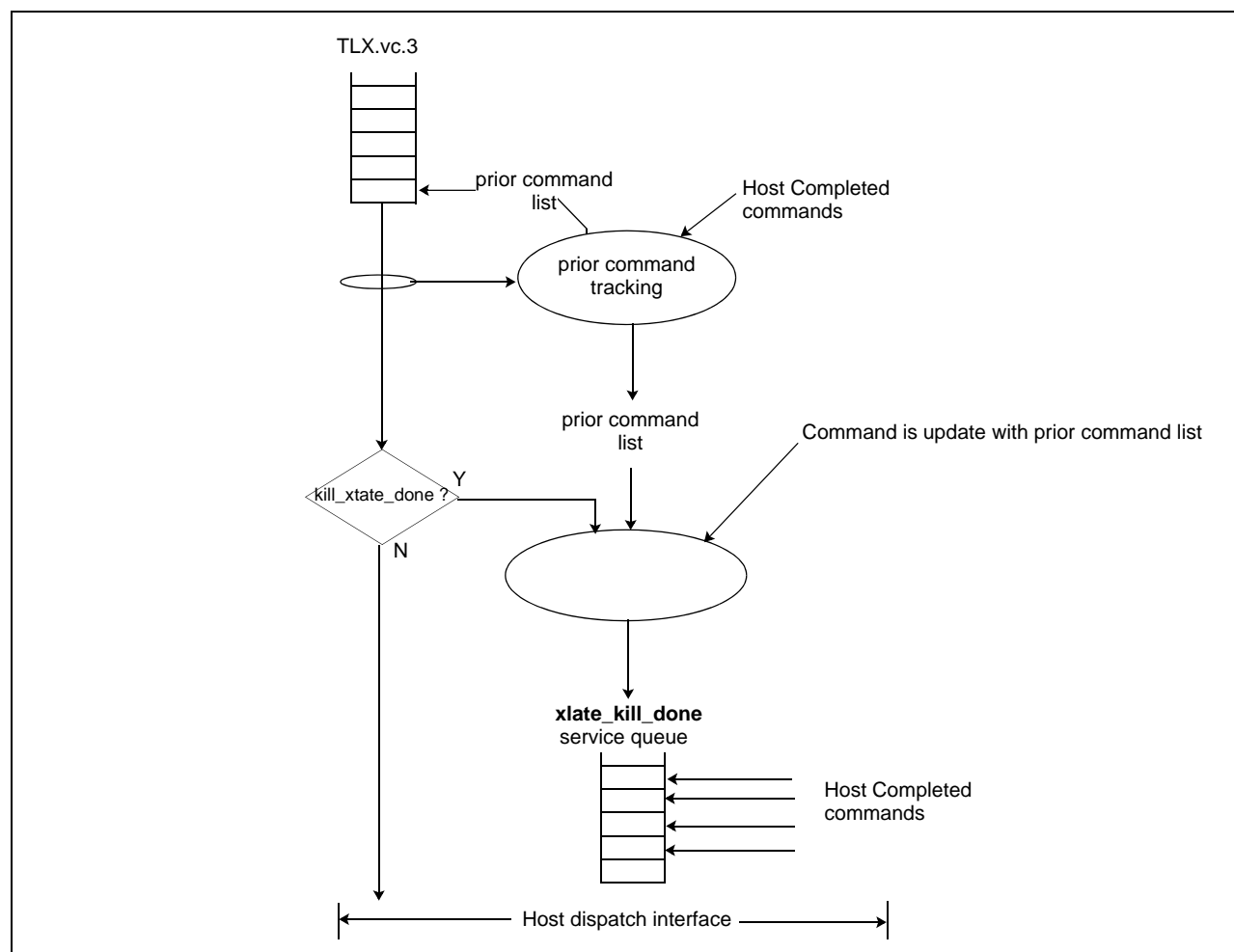
Figure 3-3 illustrates the architectural model of the steps taken when processing a **kill_xlate_done** response. The handling of this response is similar to the model described for the TL presync queues illustrated in *Figure 3-2* on page 172.

Commands are dequeued from the TLX.vc.3 queue as described in *Section 3.3 TL Virtual channel and service queues* on page 168. **kill_xlate_done** is dequeued from the VC queue and is moved to the **xlate_kill_done** service queue. The **kill_xlate_done** response is not released to the host for final processing of the **kill_xlate** command actions until all commands in the TLX.vc.3 queue prior to the **kill_xlate_done** response have been completed.

Approved

The architectural model uses the same technique introduced in the description of *Figure 3-2* on page 172. That is, a prior command list is maintained to determine when the **xlate_kill_done** response is released to the host.

Figure 3-3. *kill_xlate_done* TL flow from TLX.vc.3 to host dispatch



The architectural model of the **xlate_kill_done** service queue ensures that all prior commands that made use of the address translations invalidated by the **kill_xlate** command have completed on the host without introducing head of line blocking at the TLX.vc.3 VC queue.

3.5 Device TL virtual channel queues

Figure 3-4 on page 176 shows the steps a TL VC queue entry follows from the time it is dequeued from the TL VC queue until it is dispatched to the AFU protocol stack interface. The TL VC queue entry can be a TL command- or response-packet. TL command- and response-packets are removed from the DLX frame in slot order from a control flit. They are loaded as queue entries into the TL VC queue specified by the command or response. After the queue entry reaches the head of the TL VC queue, it is examined.

1. Any error found in the queue entry shall be reported, and the entry shall be dequeued. If the TL VC queue entry is a TL command-packet, the errors shall be reported either by a response returned or by an error

Approved

event. If the TL VC queue entry is a response-packet, errors shall be reported through error events such as *Bad response received on page 201*. Other error events are described in *Section 7.1* beginning on page 198.

2. With error checking completed, the entry shall be dequeued.

For TL VC 1 and 2 queues, the dequeued entry shall be sent to the AFU protocol dispatch interface.

Entries in the TL VC 0 queue require additional examination before being dequeued. The TL packets assigned to the TL VC 0 queue that are specified with a `host_tag` shall obtain the lock to the *host_tag entry* before being sent to the AFU protocol dispatch interface²⁸.

If an implementation cannot obtain the lock to the `host_tag` entry, there are two options:

- The queue entry may be dequeued from the VC queue to allow other commands to pass it.
- The queue entry may block the commands behind it from reaching the head of the queue.

Figure 3-4 shows a permitted implementation choice, where a TL VC 0 queue entry with a `host_tag` specified is unable to obtain a lock on the host tag. The TL VC 0 queue entry is dequeued and placed into the `host_tag` locked queue. The entry is held in the `host_tag` locked queue until the lock is released and is obtained by the waiting TL VC 0 queue entry. The implementation shall maintain TL VC 0 queue entry ordering when obtaining locks for the same `host_tag`. This requires that entries in the `host_tag` locked queue obtain `host_tag` locks before the entries in the TL VC 0 queue. Ordering between commands using the same `host_tag` within the `host_tag` locked queue shall be retained.

Engineering note

When a command- or response-packet specifies multiple `host_tag` entries, the entry locks are accumulated until all have been obtained. Once obtained the command is allowed to go to the dispatch interface.

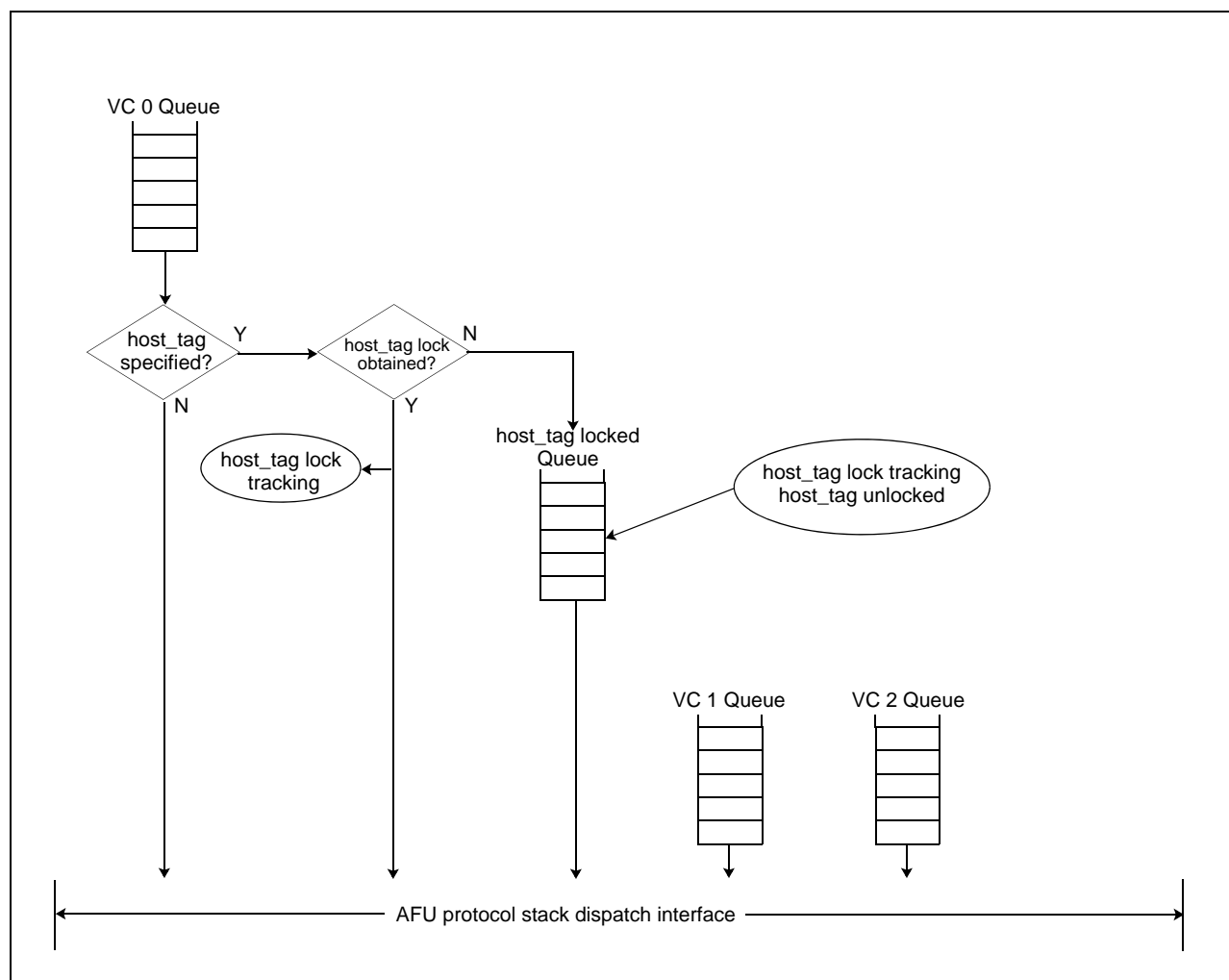
Implementations are warned to ensure that deadlock scenarios are not possible when getting locks on multiple `host_tags` for a single command- or response-packet.

At the dispatch interface, arbitration between VC queues is implementation dependent and beyond the scope of this specification.

Engineering note

The architecture *requires* that the implementation of the device's TL VC queues appear to behave as if the architectural model of the TL VC queue is implemented.

28. A command with a `host_tag` field may imply the specification of up to 4 *host_tags* per the definition of a *host_tag*.

Figure 3-4. TLX command and response flow from the VC to the AFU protocol stack TL VC queues shown

3.6 Virtual channel dependency rules

Commands and responses are assigned to virtual channels. AFU and host designs are cautioned not to implement a design where there is a potential for a dead lock when forward progress of one command is dependent on the forward progress of another.

The TL specification specifies natural VC dependencies. For example a TLX **rd_wntc** using TLX.vc.3 is dependent on a TL **read_response** using TL.vc.0 to complete the operation. Deadlock conditions can occur when:

- The host cannot respond to a TLX command without issuing a TL command and the host design does not allow the TL command to be issued.
- The AFU cannot respond to a TL command without issuing a TLX command and the AFU host design does not allow the TLX command to be issued.

Approved

Since the implementation of the host and the AFU are beyond the scope of this specification, the designers of the host and the AFU shall ensure that such dead lock conditions are prohibited by the implementation.

Figure 3-5 illustrates the interdependencies between VC that occur in the existing specification. An implementation, of either a host or device, shall not introduce dependencies not shown in *Figure 3-5*.

Graphically, the dependencies are shown as a line between two ovals. The ovals specify a virtual channel. For illustrative purposes, consider two ovals connected by an arrow. The tail of the arrow is connected to the oval indicating a TL- or TLX-packet using VC.a. The head of the arrow is connected to a TL- or TLX-packet using VC.b. Reading the graphic becomes:

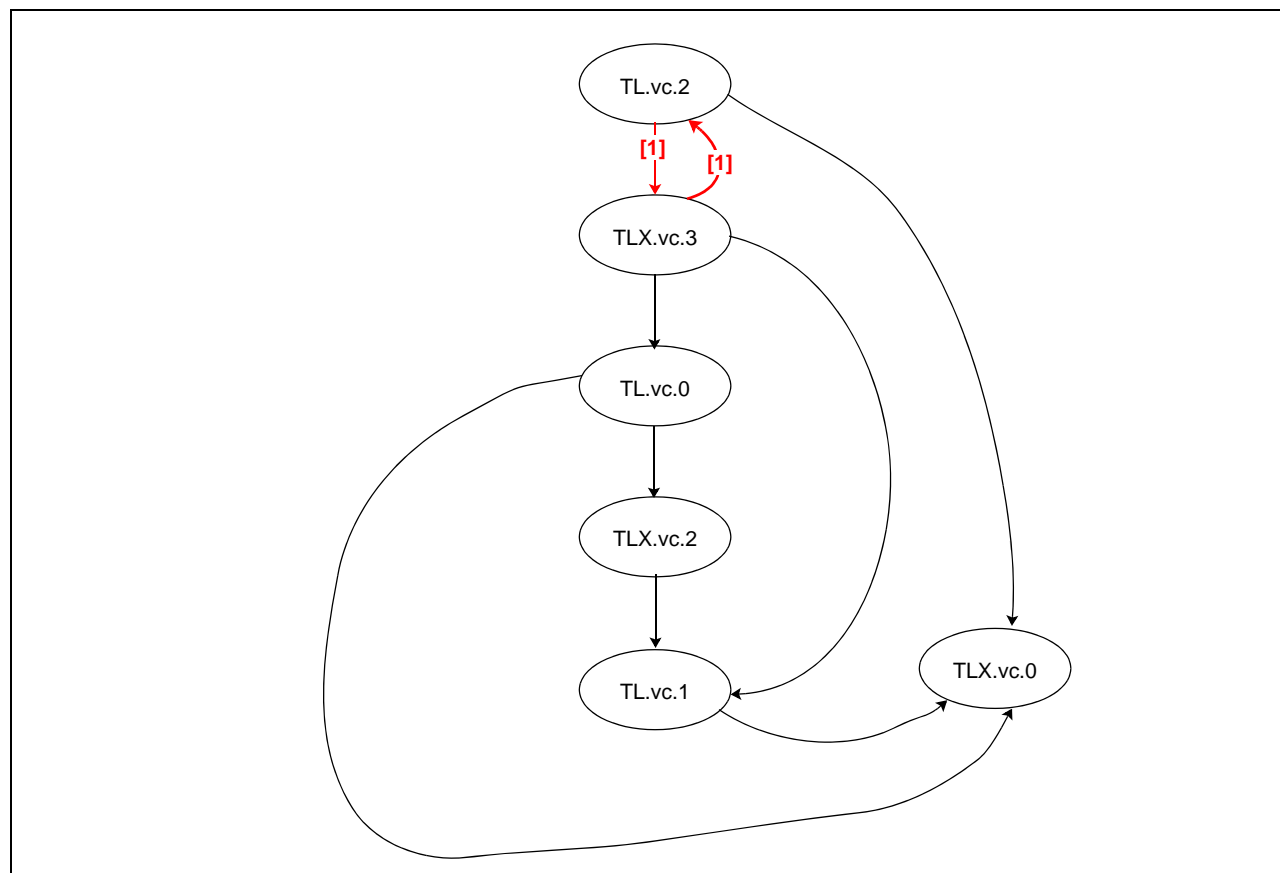
Servicing an incoming packet using VC.a requires issuing a packet using VC.b.

The intent of the architecture is to require implementations to insure that servicing an incoming packet using VC.a does not require the use of the same resources that are required to issue a packet using VC.b. This intent might restrict implementation choices.

Loops through the graph indicate potential deadlock scenarios where this specification provides behavioral rules that the AFU and host implementation shall follow, or shall appear to follow to remove the potential for deadlock conditions.

The dependency arcs shown in red place requirements on the implementation of the host and the AFU implementations.

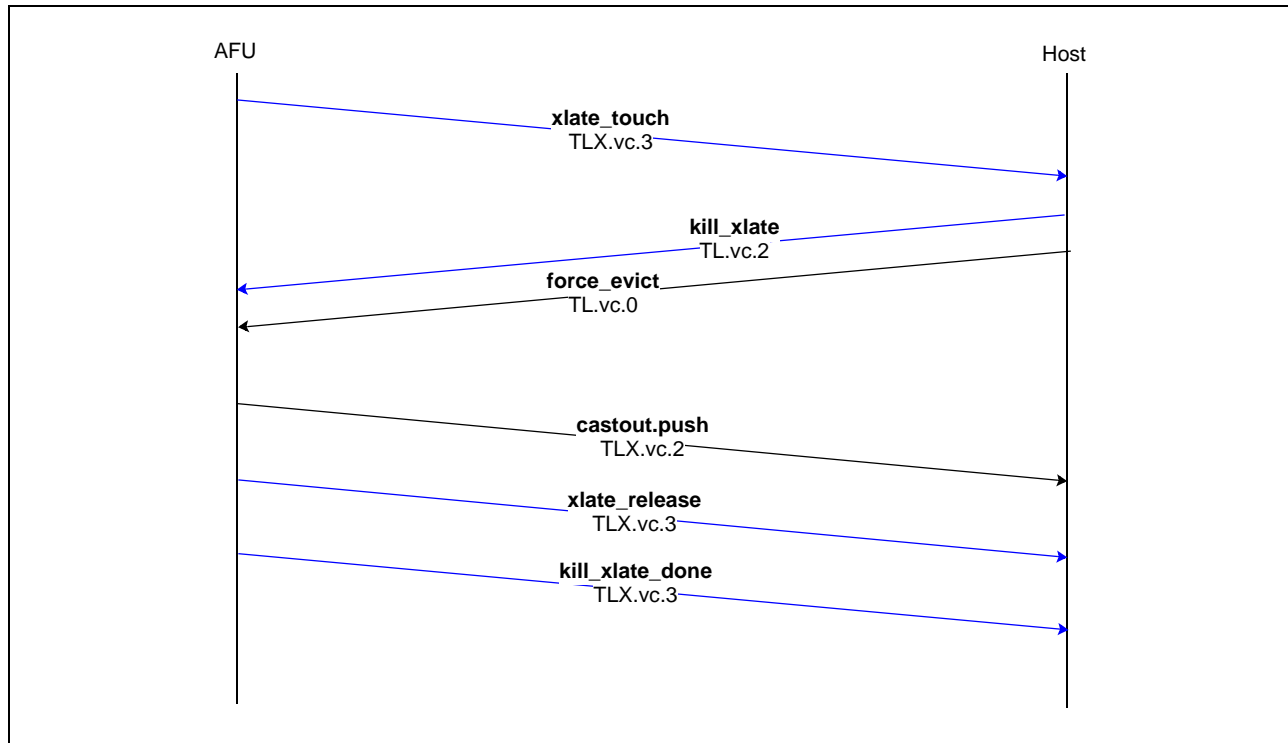
1. See **Dependency loop 1 resolution**.

Figure 3-5. VC dependency graph

3.6.1 Dependency loop 1 resolution

xlata_touch (TLX.vc.3) serviced by the host requires the host to issue a **kill_xlate** (TL.vc.2) and may issue **force_evict** (TL.vc.0) when the host is required to evict a victim from the host's ATC. The **kill_xlate** (TL.vc.2) requires that the AFU to respond with a **kill_xlate_done** (TLX.vc.3). This is loop 1 illustrated in *Figure 3-5* and annotated as [1], and in *Figure 3-6 Loop 1 detail* on page 179. The host shall victimize the ATC entry and shall retry the **xlata_touch** operation (**touch_resp**, Resp_code=rty_req or lw_rty_req). The back off retry indication used is determined by the host's implementation.

The AFU shall release all resources, for example AFU ATC set locks, when the **xlata_touch** is retried by the host. After the back off timer expiration, the AFU may retry the operation. Before starting a **xlata_touch** (TLX.vc. 3) operation, the AFU shall ensure that there are no resource conflicts with servicing running operations, such as a **kill_xlate** (TL.vc.2) that has created an ATC set lock.

Figure 3-6. Loop 1 detail

4. The acTag table

The section provides the architectural specification of the acTag table. Implementations may choose to implement this table using any method. However, the externally observable behavior of the table and contents of the table shall, at a minimum, comply to this architectural specification.

The acTag table shall be included in the host's implementation and shall provide a minimum of one entry.

4.1 acTag table contents

Each entry of the acTag table (acTag entry) shall contain the following fields:

- 1-bit entry valid. Architected states are defined as {valid, invalid}
- 16-bit BDF
- 20-bit PASID

An implementation may choose to add additional fields to the acTag entry.

4.2 acTag table access

The OpenCAPI device maintains a copy of each acTag entry to determine the acTag value used in TLX commands specified with an acTag field.

The acTag table is accessed using the acTag as follows:

- Read access uses the acTag provided in a TLX command specified with an acTag field as an index into the table to locate the acTag entry to be read. Reading an acTag entry returns the entry valid indication and, when valid, a BDF and PASID.
- Write access uses the acTag provided in the **assign_actag** command as an index into the table to locate the acTag entry to be updated. The command provides a BDF and PASID, which are loaded into the acTag entry. Successful completion of the write access sets the entry valid bit to the valid state.

4.2.1 Error cases when accessing the acTag table

Error	Action
acTag entry not valid	This is a fatal error. See "acTag specified in the command points to an invalid entry" in <i>Table 7-1</i> on page 199.
Address context not valid	This is determined either at the time the acTag entry is created or when the acTag entry is used to obtain the address context. This is a fatal error. See "Bad BDF and PASID combination" in <i>Table 7-1</i> on page 199.

4.3 acTag entry management

The entries of the acTag table are managed by the attached OpenCAPI device. The OpenCAPI device shall maintain its own mapping between an acTag and the contents of the acTag entry held in the host's acTag table.

Approved

During configuration, the host writes to the OpenCAPI device's configuration space to indicate the maximum size of the acTag, which indirectly specifies the size of the host's acTag table. An acTag size of 0 indicates a single entry acTag table, and the only valid acTag value is '0'. The OpenCAPI device may use any value within the allowed range to specify an acTag entry and subsequently refer to that entry using a TLX command specifying the acTag.

The OpenCAPI device learns its bus number from the address specified in a **config_write**, T=0 command. Device and function numbers are assigned by the OpenCAPI device and discovered by the host during configuration. See the specification of **config_write** for the format of the address field that contains the bus, device and function numbers.

The OpenCAPI device is configured with one or more PASIDs during initialization and operation.

After an acTag entry is set to a valid state, it is set to an invalid state only by the host upon detection of a link failure that requires resetting the OpenCAPI interface and the OpenCAPI device.

Engineering note

In a host implementation, a configuration register would be useful to vary the size of the acTag table, which allows stress testing of the design. A proposed specification of the configuration register follows:

The register contains a value N where the size of the acTag table is 2^N and the acTag range is limited, as described previously, to a range of $0..2^{N-1}$.

Permitted access methods for this configuration register, as well as the register contents, are specified by the platform architecture.

5. DL frame format

The TL and TLX contain framer and parser functions that work with the DL frame format. The DL frame format is specified as a set of 64-byte flits. There are two types of flits:

- Control flits. The control flit contains TL command/response content and DL content. The DL content contains several DL-generated subfields including the CRC that covers the control flit and any *preceding* data flits. There are fields in the DL content that are generated by the TL. For more information, see *Section 5.1.1 DL content* on page 183.
- Data flits. There are 0 to 8 data flits between each control flit.

Table 5-1. DL frame format showing CRC and “bad data flit” coverage

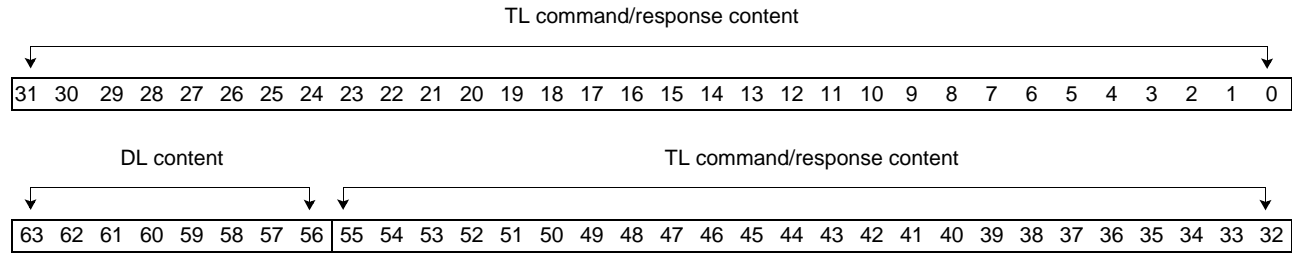
Bytes(63:0)	
DL content	TL command/response/32-, 8-byte data content
Data flit 0	
Data flit 1	
Data flit 2	
Data flit 3	
Data flit 4	
Data flit 5	
Data flit 6	
Data flit 7	
DL content	TL command/response/32-, 8-byte data content
Data flit 0	
Data flit 1	
DL content	TL command/response/32-, 8-byte data content
DL content	TL command/response/32-, 8-byte data content

Table 5-1 uses color to illustrate the coverage of the CRC found in the DL content of the control flit. The CRC covers the control flit that it is contained in and all *previous*, if any, data flits. The DL content found in the control flit also contains “bad data flit indicators” for the previous data flits. The 32- and 8-byte data fields carry bad data indicators for the associated data field in the control flit. A control flit specified by the TL shall always follow data flits as shown in Table 5-1. The DL may insert DL-idle-control flits when the TL interface to the DL is idle.

The transmit order in Table 5-1 is from right to left and top to bottom. That is, data flits are transmitted in increasing address order. Control flits follow this convention.

Approved

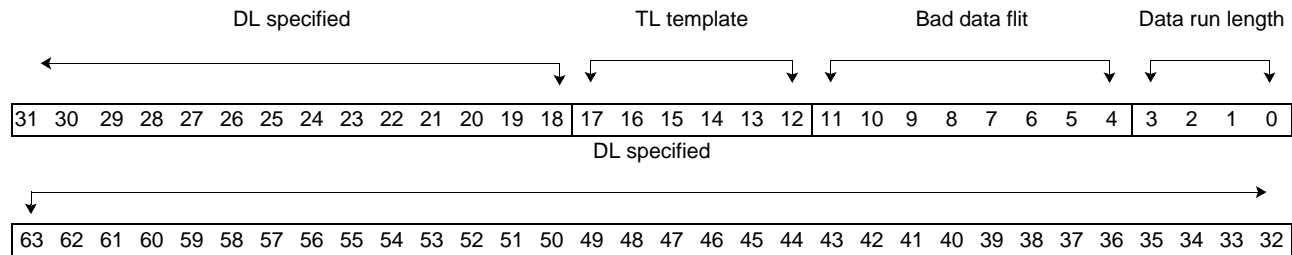
5.1 DL frame control flit (64 bytes)



Bytes	Field name	Description
55:0	TL command/response content	This field contains information provided by the TL. This 448-bit field (447:0) is comprised of sixteen 28-bit slots. One or more slots comprise either a null entry, TL command packet, TL response packet, metadata, extended metadata, or data with the location and length specified by the TL template. See <i>Section 5.1.2 TL command/response content</i> on page 184 for slot layout information.
63:56	DL content	This field contains information added by both the TL and the DL layer. See <i>Section 5.1.1 DL content</i> for the specification of this field.

5.1.1 DL content

This field is bytes 63:56 of the DL frame control flit.



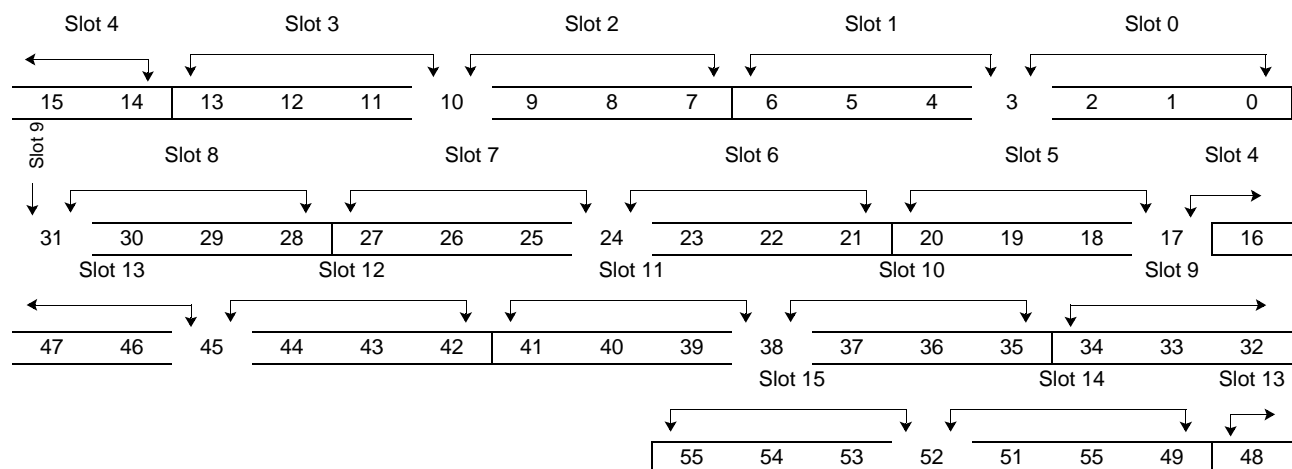
Bit	Field name	Description (Page 1 of 2)
3:0	Data run length	This 4-bit field indicates the number of data flits until the next control flit. A value of 0 indicates that the next flit is a control flit. Valid values are {0...8}.
11:4	Bad data flit indication	This 8-bit field indicates that data flits received prior to this control flit contain bad data and shall not be used without being marked as bad (for example, mark as SUE). Each bit corresponds to one data flit (for example, bit 0 corresponds to data flit 0, which is the first data flit following the previous control flit). See <i>Table 5-1</i> on page 182 to match up the data flit being reported to the bit in this field. 11 Data flit 7 is in error. 10 Data flit 6 is in error. 9 Data flit 5 is in error. 8 Data flit 4 is in error. 7 Data flit 3 is in error. 6 Data flit 2 is in error. 5 Data flit 1 is in error. 4 Data flit 0 is in error.
17:12	TL template	This 6-bit field specifies the locations of opcodes found in the 448-bit TL command and response content field. See <i>Section 6 TL and TLX template specifications</i> on page 189 for the specification of this field.

Approved

Bit	Field name	Description (Page 2 of 2)
63:18	DL specified	The specification for this 46-bit field is found in the OpenCAPI DL specification . This field is expected to contain the CRC, ACK, and ACK count information.

5.1.2 TL command/response content

Slots are packed into the DL control flit as shown in the following figure. Each slot occupies 3.5 bytes (28 bits). The slot number and control flit byte are shown. The number of slots used by a command or response can be found in the specification of the command or response.



5.1.3 Data transport, order, and alignment

Data is transported between the TL and TLX using *data carriers*, which are specified as 64-byte data flits, or 32- or 8-byte data fields in control flits. To transport data, one or more DCP credits shall be obtained atomically when obtaining the VC credit required to send a command or response. A DCP credit is associated with at most 64 bytes of data and may be associated with 32- or 8-bytes of data.

1. When a single command or response specifies 64-bytes of immediate data, a single DCP credit is required. Either a 64-byte data flit or 2 32-byte data carriers may be used.
2. When a single command or response specifies 128- or 256-bytes of immediate data, 2 and 4 DCP credits are required. Multiple 64-byte data flits, or multiple 32-byte data carriers, or a combination of 64- and 32-byte data carriers may be used.
3. When a single command or response specifies 32 or fewer bytes of immediate data, a single DCP credit is required and a single data carrier shall be used. This applies to commands and responses with a pLength field specified, dot-be commands, 8-byte data carrier use, and dot-ow responses.

A data carrier shall be associated with a single command or response. Multiple data carriers may be associated with a single command or response.

Within each control flit, there is an order to the commands and responses as shown in *Section 5.1.2 TL command/response content* on page 184. Commands and responses loaded into lower numbered slots are ordered before commands and responses loaded into higher numbered slots. Data shall be loaded into data carriers by the TL in the same order as the commands and responses are specified in the control flits. Data carried by control flits that are loaded into lower numbered slots are ordered before data carried by higher

Approved

numbered slots. Data carried in a control slot is ordered before the data carried in data flits that follow. In control flits with data carriers, the commands or responses are treated as if they precede the data. For example, a command or response that specifies immediate data may find the data in the same control flit as the command or response.

For each command and response associated with data, there is an implied length or a dLength field and a dPart field specified. These are used to pull the data out of the data carriers and associate the data with the command or response. Each data carrier is examined using the length information and the data is associated with a single command or response. (Additional association can be made using the AFU or CAPP tag provided to locate the machine associated with the command or response.) In some cases (for example, **dma_pr_w**), the dLength field is implicit and is specified as a single data carrier. The minimum size of the data carrier is determined by the command's pLength field.

Data shall be address aligned within a 64-, 32- and 8-byte data carrier. That is, this architecture treats a data carrier as if it were a memory-mapped naturally aligned data block, where each byte of data is loaded into the data carrier based on the address of the byte being loaded. When a command or response with immediate data uses multiple data carriers, the data shall be loaded in increasing address order. That is, offset 0 from the address specified by the command or response, and adjusted by the dPart field, shall be loaded first and the remaining data is loaded in increasing address order.

When the data is not associated with an address, the data shall be placed starting at byte 0 of the data carrier and increasing byte locations until all the data has been loaded into the data carrier. A command or response with this type of data shall use only a single data carrier. Additional restrictions might be found in the command and response description.

Engineering note

There are naturally occurring cases when all bytes of a data carrier are not fully specified by the command or response associated with the data. For example, the data associated with a **dma_pr_w** does not specify all bytes within a 64-byte data flit. That is, there are bytes within the data carrier that are defined by the write operation based on pLength and starting address, and there are bytes that are undefined by the architecture.

It is strongly recommended that the architecturally undefined data bytes found within a data carrier do not contain information associated with any application other than the application associated with the command or response and is limited to the permissions granted to the application.

The method used to ensure that the contents of undefined data locations within a data carrier are not from a different process is determined by the implementation. Suggested methods include, but are not limited to the following:

- Set all undefined byte locations to zero.
- Set all undefined byte locations to a fixed or random non-zero pattern of bits
- Replicate the content of defined byte locations to undefined byte locations.

The architecture does not provide architectural conformance statements regarding the contents of undefined byte locations within a data carrier other than the conformance statement specified by the definition of the architectural term "*undefined*".

A 64-byte data flit may be used with any type of data that is specified for one of the following:

- Any command
- Responses not specified as dot-ow and dot-xw

One to four data flits may be used to provide data associated with an address. The number of data flits is dependent on the number of bytes specified by the command or response associated with the immediate

Approved

data. When multiple data flits are used, the data flits shall be loaded in increasing address order based on the dLength and dPart specified in the command or response.

A 32-byte data field found in a control flit shall be used to carry data associated with an address only. One or more 32-byte data fields may be associated with a single command or response. When multiple 32-byte data carriers are used, data shall be loaded in increasing address order based on the dLength and dPart specified in the command or response.

- dot-ow responses are associated with a single 32-byte data field.
- dot-xw responses shall not use 32-byte data fields as a data carrier.

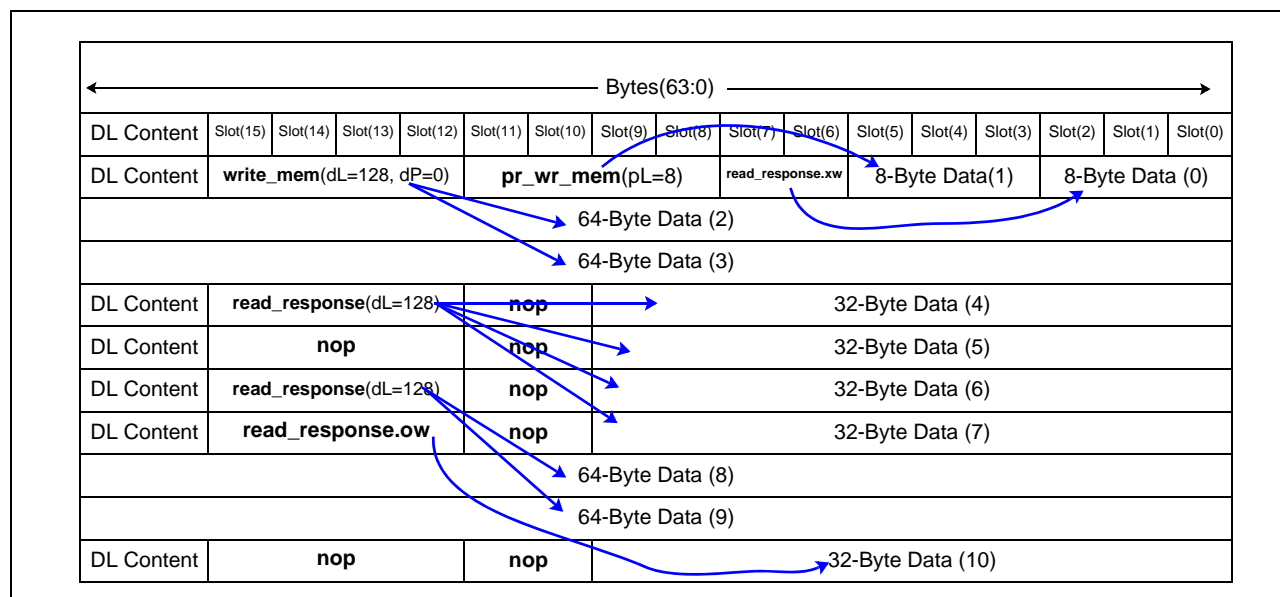
A combination of 64- and 32-byte data carriers may be used for commands and responses with 64- 128- or 256-bytes of immediate data specified. 64-byte carriers shall be loaded on aligned 64 byte boundaries. 32-byte data carriers shall be loaded on aligned 32-byte boundaries. Note that this restricts when an implementation is allowed to switch between using 64- and 32-byte data carriers since the architecture requires that the data be loaded in increasing address order regardless of the mix of data carriers used.

An 8-byte data field found in a control flit may be used to carry data associated with an address only. Only one 8-byte data field shall be associated with a command or response.

- dot-xw responses shall use only 8-byte data fields as a data carrier.

Table 5-2 illustrates how a command and response stream's data might be packed into a series of control and data flits following the ordering rules for data.

Table 5-2. DL frame loading to illustrate data ordering



5.1.3.1 Data alignment for commands and responses specifying a host_tag field.

castout.push, **cl_rd_resp** and **cl_rd_resp.ow** use a host_tag field and do not have an address directly associated with the data.

Approved

Data is placed into data carriers following the same rules as commands and responses with addresses described in the previous section. This section defines how the `host_tag` is associated with the alignment of immediate data in 64- and 32-byte data carriers.

castout.push:

castout.push contains a `dLength` field that specifies the size and alignment of the *naturally aligned data block* associated with the immediate data. The immediate data may be sent using 64- or 32-byte data carriers, or a mix of 32- and 64-byte data carriers. The `host_tag` specified is associated with a naturally aligned 64-byte segment found at offset 0 of the full data block.

- When `dLength` is specified as 64-bytes, there is a single `host_tag` and the data is treated as a naturally aligned 64-byte data block.
- When the `dLength` is specified as 128 bytes, there are two `host_tags`. The `host_tag` specified is associated with the naturally aligned 64-byte segment found at offset 0 of the 128-byte data block. Based on the *host_tag arithmetic* specification, `host_tag + 1` is associated with the naturally aligned 64-byte segment found at offset 64 of the 128-byte data block.
- When the `dLength` is specified as 256 bytes, there are four `host_tags`. The `host_tag` specified is associated with the naturally aligned 64-byte segment found at offset 0 of the 256-byte data block. Based on the *host_tag arithmetic* specification, `host_tag + 1` is associated with the naturally aligned 64-byte segment starting at offset 64 of the 256-byte data block. Similarly, `host_tag + 2` is associated with the naturally aligned 64-byte segment starting at offset 128. `host_tag + 3` is associated with the naturally aligned 64-byte segment starting at offset 192.

cl_rd_resp:

cl_rd_resp is in response to a **read_me**, **read_mes**, **read_s**, or dot-t variants of these commands. These commands contain an EA or TA field and `dLength`. The data requested is a *naturally aligned data block* based on the `dLength` field.

cl_rd_resp contains a `dLength`, `dPart` and a `host_tag` field. The *dPart(1:0)* indicates the `dLength` sized naturally aligned data block associated with the response. See the description of *dPart(1:0)* for the offset of the data block returned with the response within the data block specified by the command. The `host_tag` specified is associated with the naturally aligned 64-byte segment found at offset 0 of the data block associated with the response.

- When `dLength` is specified as 64-bytes, there is a single `host_tag` and the data is treated as a naturally aligned 64-byte data block.
- When the `dLength` is specified as 128 bytes, there are two `host_tags`. The `host_tag` specified is associated with the naturally aligned 64-byte segment found at offset 0 of the 128-byte data block. Based on the *host_tag arithmetic* specification, `host_tag + 1` is associated with the naturally aligned 64-byte segment found at offset 64 of the 128-byte data block.
- When the `dLength` is specified as 256 bytes, there are four `host_tags`. The `host_tag` specified is associated with the naturally aligned 64-byte segment found at offset 0 of the 256-byte data block. Based on the *host_tag arithmetic* specification, `host_tag + 1` is associated with the naturally aligned 64-byte segment starting at offset 64 of the 256-byte data block. Similarly, `host_tag + 2` is associated with the naturally aligned 64-byte segment starting at offset 128. `host_tag + 3` is associated with the naturally aligned 64-byte segment starting at offset 192.

Approved

cl_rd_resp.ow:

cl_rd_resp.ow specifies a single 32-byte data carrier only and is used in response to **read_me**, **read_mes**, **read_s**, or dot-t variants of these commands. *dPart(2:0)* specifies the 32-byte offset of the data block returned with the response within the data block specified by the command.

Bit 0 of *dPart(2:0)* indicates the offset within the 64-byte *naturally aligned data block* associated with the *host_tag* specified in the response. When bit 0 is '0', the offset is 0. When bit 0 is '1', the offset is 32. One and only one *host_tag* is associated with each even and odd value *dPart(2:0)* pair. That is, *dPart(2:0)* values of 0 and 1 are associated with a *host_tag*, *dPart(2:0)* values of 2 and 3 are associated with different *host_tag* and so on.

Since the specification of the response cannot span 64-byte address boundaries, there is only one *host_tag* associated with each **cl_rd_resp.ow**

6. TL and TLX template specifications

This specification defines the allowed placement formats of TL/TLX packets, metadata, extended metadata, 8- and 32-byte data fields within the DL/DLX frame's control flit. The allowed formats are captured in four capability descriptions defined in *Table 6-1* on page 190.

A TL template field is specified within the DL content of a DL packet's control flit. The DL content of the control flit is shown in *Section 5.1 DL frame control flit (64 bytes)* on page 183. The DLX frame has the same format as the DL frame.

The TL architecture specifies all template capability descriptions and specifies a number for each specification that is used in the TL template field. When transmitting a packet:

- The TL shall place the TL transmit template number used to form the control flit of the DL frame.
- The TLX shall place the TLX transmit template number used to form the control flit of the DLX frame.

The template capabilities specify the legal locations of one or more TL/TLX command or response packets' starting slot²⁹ as well as the contiguous number of slots used by the packet. In addition to TL/TLX command and response packets, the templates specify the legal locations of metadata and data and the contiguous number of slots used by the metadata and data fields. The format of the data fields is also specified. Unused control flit slots are reserved. That is, unused control flit slots shall be set to an all zero state when transmitted and shall not be examined on receipt for any purpose other than CRC checking.

The template restricts the maximum length of TL/TLX command or response packets placed in the control flit. The template specification permits a smaller packet to be placed into a larger specification footprint. For example, a six-slot template specification may be filled with a 4-, 2-, or 1-slot packet. The state of the unused slots is undefined.

Developer Note

Allowing smaller TL packets to occupy larger packet-specified locations helps to reduce the number of template specifications.

A template may further restrict the TL/TLX command or response packet placed into packet locations. These restrictions include and are not limited to the following:

- Command
- Response
- VC used
- DCP requirement (present or absent)

Table 6-1 on page 190 defines the template capabilities.

²⁹ Bits 27:0 of the TL/TLX packet specification.

Approved

Table 6-1. Template capability definitions

Capability	Definition	Specified by
TLX receive template	Specifies the templates that the OpenCAPI device supports when receiving DL frames.	OpenCAPI device
TL transmit template	Specifies the templates that the host supports when transmitting a DL frame to the OpenCAPI device.	Host
TL receive template	Specifies the templates that the host supports when receiving DLX frames.	Host
TLX transmit template	Specifies the templates that the OpenCAPI device supports when transmitting a DLX frame to the host.	OpenCAPI device

The intersection of the TL transmit template capability and the TLX receive template capability shall not be a null set. All OpenCAPI devices shall support TLX receive template x'00'.

The intersection of the TLX transmit template capability and the TL receive template capability shall not be a null set. All OpenCAPI hosts shall support TL receive template x'00'.

See the host's platform architecture for additional information about how the receive and transmit capabilities are resolved and what is stored into the OpenCAPI device's configuration space.

Table 6-2 defines the terms used in Section 6.1 and Section 6.2.

Table 6-2. Terms used in template capability specifications (Page 1 of 2)

Term	Width (bits)	Description
<n>-slot <type> packet	n*28	The number <n> of slots specified for a packet <type> of either a TL or TLX.
Data(x:y)		Indicates a data field of x + 1 - y bits in length.
xmeta	72	Extended-metadata. A 72-bit field associated with a 32 byte <i>naturally aligned data block</i> . Extended metadata is placed in the control flit carrying the 32-byte data block. The specification of the extended-metadata is outside the scope of this architecture and is found in the host and OpenCAPI device's documentation.
Meta	7	Metadata. A 7-bit field associated with a <i>naturally aligned data block</i> . The size of the data block is implementation dependent. Metadata may be associated with blocks smaller than the implementation-specified size during data transfer. The specification of the metadata is outside the scope of this architecture and is found in the host and OpenCAPI device's documentation. The implementation defines the transformation of the metadata associated with a smaller block when merged into a larger data block. For example, the implementation's data block size might be 64 bytes and an update to an 8-byte <i>naturally aligned data block</i> might be required. When the 8-byte block is provided with metadata, the implementation determines how the metadata provided with the 8-byte block is used to transform the metadata associated with the 64-byte block. <ul style="list-style-type: none"> mdf(n) indicates metadata associated with the nth data flit following the control flit containing the metadata. Each field is 7 bits wide. meta(6:0) specifies a single metadata field associated with data found in the control flit.
R		Indicates a reserved bit in a slot. <ul style="list-style-type: none"> R(x:y) specifies x + 1 - y reserved bits R(0) specifies a single reserved bit

Approved

Table 6-2. Terms used in template capability specifications (Page 2 of 2)

Term	Width (bits)	Description
V	2	<p>A 2-bit field indicating that the associated data is valid and if it is bad.</p> <p>Bit Description</p> <p>0 Bad data indication. This bit is valid only when bit 1 is set to '1'.</p> <p>0 The associated data is good.</p> <p>1 The associated data is bad.</p> <p>1 Valid field indication.</p> <p>0 The associated data is not valid.</p> <p>1 The associated data and bit 0 are valid.</p>

6.1 TLX receive and TL transmit template capability specification

Table 6-3. TLX receive/TL transmit template (Page 1 of 3)

Slot #	x'00'	x'01'	x'02'	x'03'
0	return_tl原因_credits ^a	4-slot TL packet	2-slot TL packet	4-slot TL packet
1				
2	reserved	4-slot TL packet	2-slot TL packet	4-slot TL packet
3				
4	6-slot TL packet	4-slot TL packet	2-slot TL packet	6-slot TL packet
5				
6		4-slot TL packet	2-slot TL packet	
7				
8			2-slot TL packet	
9	reserved	4-slot TL packet	2-slot TL packet	6-slot TL packet
10				
11		4-slot TL packet	2-slot TL packet	
12				
13			2-slot TL packet	
14				
15			2-slot TL packet	

Approved

Table 6-3. TLX receive/TL transmit template (Page 2 of 3)

Slot #	x'04'	x'05'	x'06'	x'07' ^b
0	2-slot TL packet	2-slot TL packet	2-slot TL packet	Data(27:0)
1				Data(55:28)
2	mdf(3) mdf(2) mdf(1) mdf(0)	mdf(3) mdf(2) mdf(1) mdf(0)	mdf(3) mdf(2) mdf(1) mdf(0)	Data(83:56)
3	mdf(7) mdf(6) mdf(5) mdf(4)	mdf(7) mdf(6) mdf(5) mdf(4)	mdf(7) mdf(6) mdf(5) mdf(4)	Data(111:84)
4	4-slot TL packet	1-slot TL packet	6-slot TL packet	Data(139:112)
5		1-slot TL packet		Data(167:140)
6		1-slot TL packet		Data(195:168)
7		1-slot TL packet		Data(223:196)
8	4-slot TL packet	1-slot TL packet		Data(251:224)
9		1-slot TL packet		mdf(1) mdf(0) R(0) V(1:0) meta(6:0) Data(255:252)
10		1-slot TL packet	2-slot TL packet	
11		1-slot TL packet		
12	4-slot TL packet	4-slot TL packet	6-slot TL packet	4-slot TL packet
13				
14				
15				

Approved

Table 6-3. TLX receive/TL transmit template (Page 3 of 3)

Slot #	x'08' ^c	x'09' ^d	x'0A' ^e	x'0B' ^f
0	Data(27:0)	Data(27:0)	Data(27:0)	Data(27:0)
1	Data(55:28)	Data(55:28)	Data(55:28)	Data(55:28)
2	R(10:0) V(1:0) meta(6:0) Data(63:56)	Data(83:56)	Data(83:56)	Data(83:56)
3	Data(27:0)	Data(111:84)	Data(111:84)	Data(111:84)
4	Data(55:28)	Data(139:112)	Data(139:112)	Data(139:112)
5	R(10:0) V(1:0) meta(6:0) Data(63:56)	Data(167:140)	Data(167:140)	Data(167:140)
6	2-slot TL packet	Data(195:168)	Data(195:168)	Data(195:168)
7		Data(223:196)	Data(223:196)	Data(223:196)
8	4-slot TL packet	Data(251:224)	Data(251:224)	Data(251:224)
9		mdf(1) mdf(0) R(0) V(1:0) meta(6:0) Data(255:252)	xmeta(23:0) Data(255:252)	xmeta(23:0) Data(255:252)
10		2-slot TL packet	xmeta(51:24)	xmeta(51:24)
11			R(5:0) V(1:0) xmeta(71:52)	R(5:0) V(1:0) xmeta(71:52)
12	4-slot TL packet	1-slot TL packet	4-slot TL packet	2-slot TL packet
13		1-slot TL packet		1-slot TL packet
14		1-slot TL packet		
15		1-slot TL packet		1-slot TL packet

a. Template x'0' slots 0 and 1 shall contain either a **nop** or **return_tlx_credits**

b. Template x'07' is limited to a two data flit run length. Slot 9 contains the metadata for data flits 0 and 1 as shown.

c. Template x'08' should only be used when there is at least one 8-byte data carrier valid. That is, a single 8-byte data shall be placed in slots 0 to 2. Violating this rule may cause a *Bad template usage* error to be reported. Metadata fields are not provided in this template. This template shall only be used when metadata is not required for the data in the data flits following the control flit using this template.

d. Template x'09' is limited to a two data flit run length. Slot 9 contains the metadata for data flits 0 and 1 as shown.

e. Template x'0A' specifies extended metadata associated with the 32-byte data carrier in the control flit. This template shall only be used when metadata is not required for the data in the data flits following the control flit using this template.

f. Template x'0B' specifies extended metadata associated with the 32-byte data carrier in the control flit. This template shall only be used when metadata is not required for the data in the data flits following the control flit using this template.

Approved

6.2 TL receive and TLX transmit template capability specification

Table 6-4. TL receive/TLX transmit template (Page 1 of 2)

Slot #	x'00'	x'01'	x'02'	x'03'
0	return_tl_credits ^a	4-slot TLX packet	2-slot TLX packet	4-slot TLX packet
1				
2	reserved	4-slot TLX packet	2-slot TLX packet	4-slot TLX packet
3				
4	6-slot TLX packet	4-slot TLX packet	2-slot TLX packet	6-slot TLX packet
5				
6		4-slot TLX packet	2-slot TLX packet	
7				
8	reserved	4-slot TLX packet	2-slot TLX packet	6-slot TLX packet
9				
10		4-slot TLX packet	2-slot TLX packet	
11				
12	reserved	4-slot TLX packet	2-slot TLX packet	6-slot TLX packet
13				
14		4-slot TLX packet	2-slot TLX packet	
15				
Slot #	x'04'	x'05'	x'06'	x'07' ^b
0	2-slot TLX packet	2-slot TLX packet	2-slot TLX packet	Data(27:0)
1				Data(55:28)
2	mdf(3) mdf(2) mdf(1) mdf(0)	mdf(3) mdf(2) mdf(1) mdf(0)	mdf(3) mdf(2) mdf(1) mdf(0)	Data(83:56)
3	mdf(7) mdf(6) mdf(5) mdf(4)	mdf(7) mdf(6) mdf(5) mdf(4)	mdf(7) mdf(6) mdf(5) mdf(4)	Data(111:84)
4	4-slot TLX packet	1-slot TLX packet	6-slot TLX packet	Data(139:112)
5		1-slot TLX packet		Data(167:140)
6		1-slot TLX packet		Data(195:168)
7		1-slot TLX packet		Data(223:196)
8	4-slot TLX packet	1-slot TLX packet	6-slot TLX packet	Data(251:224)
9		1-slot TLX packet		mdf(1) mdf(0) R(0) V(1:0) meta(6:0) Data(255:252)
10		1-slot TLX packet		2-slot TLX packet
11		1-slot TLX packet		
12	4-slot TLX packet	4-slot TLX packet	6-slot TLX packet	4-slot TLX packet
13				
14				
15				

Approved

Table 6-4. TL receive/TLX transmit template (Page 2 of 2)

Slot #	x'08' ^c	x'09' ^d	x'0A' ^e	x'0B' ^f
0	Data(27:0)	Data(27:0)	Data(27:0)	Data(27:0)
1	Data(55:28)	Data(55:28)	Data(55:28)	Data(55:28)
2	R(10:0) V(1:0) meta(6:0) Data(63:56)	Data(83:56)	Data(83:56)	Data(83:56)
3	Data(27:0)	Data(111:84)	Data(111:84)	Data(111:84)
4	Data(55:28)	Data(139:112)	Data(139:112)	Data(139:112)
5	R(10:0) V(1:0) meta(6:0) Data(63:56)	Data(167:140)	Data(167:140)	Data(167:140)
6	2-slot TLX packet	Data(195:168)	Data(195:168)	Data(195:168)
7		Data(223:196)	Data(223:196)	Data(223:196)
8	4-slot TLX packet	Data(251:224)	Data(251:224)	Data(251:224)
9		mdf(1) mdf(0) R(0) V(1:0) meta(6:0) Data(255:252)	xmeta(23:0) Data(255:252)	xmeta(23:0) Data(255:252)
10		2-slot TLX packet	xmeta(51:24)	xmeta(51:24)
11			R(5:0) V(1:0) xmeta(71:52)	R(5:0) V(1:0) xmeta(71:52)
12	4-slot TLX packet	1-slot TLX packet	4-slot TLX packet	2-slot TLX packet
13		1-slot TLX packet		1-slot TLX packet
14		1-slot TLX packet		
15		1-slot TLX packet		1-slot TLX packet

- a. Template x'00' slots 0 and 1 shall contain either a **nop** or **return_tl_credits**.
- b. Template x'07' is limited to a two data flit run length. Slot 9 contains the metadata for data flits 0 and 1 as shown.
- c. Template x'08' should only be used when there is at least one 8-byte data valid. A single 8-byte data shall be placed in slots 0 to 2. Violating this rule may cause a *Bad template usage* error to be reported. Metadata fields are not provided in this template. This template shall only be used when metadata is not required for the data in the data flits following the control flit using this template.
- d. Template x'09' is limited to a two data flit run length. Slot 9 contains the metadata for data flits 0 and 1 as shown.
- e. Template x'0A' specifies extended metadata associated with the 32-byte data carrier in the control flit. This template shall only be used when meta data is not required for the data in the data flits following the control flit using this template.
- f. Template x'0B' specifies extended metadata associated with the 32-byte data carrier in the control flit. This template shall only be used when meta data is not required for the data in the data flits following the control flit using this template.

6.3 Control-flit rate capability

Each receive and transmit template capability specification has an associated control-flit rate capability. When a template is used to format a control flit, the template's associated control flit rate capability specifies when the next control flit may be sent.

The control-flit rate capability controls the DL flit spacing between control flits. The configuration space for this capability provides 4 bits.

- A value of x'0' indicates that a control flit may be sent in the following *flit-cycle*.
- A value of x'1' indicates that the cycle following the control flit shall contain either a null control flit or a data flit.

Approved

In general, a value of “n” indicates that there shall be a gap of “n” 64-byte flits before the next control flit can be sent. During the gap, either null control flits or data flits shall be inserted.

A null control flit is defined as using template x'00'. The 6-slot packet contains a 1-slot null command, and the remaining five slots are undefined. A return credit response found in slots 0 and 1 may be used to return credits.

Engineering note

At the start of initialization of an OpenCAPI device, the flit rate capability is unknown since the configuration space has not yet been examined. Template x'00' is used to issue **config_read** commands to determine the device's capabilities.

Until the device's capabilities are determined, template 0 shall be used and the control flit rate capability shall be assumed to be x'F'.

6.4 Metadata capability

Templates x'04' through x'06' allow the specification of metadata associated with up to eight data flits.

Templates x'07' and '09' allow specification of metadata associated with up to two data flits. To enable this, a metadata capability is specified in the configuration space for the host and the OpenCAPI device.

Templates x'0A' and x'0B' allows the specification of extended-metadata for the 32-byte data carrier specified in the control flit. To enable this, an extended-metadata capability is specified in the configuration of the host and the configuration space for the OpenCAPI device.

7. Error detection

This section identifies errors and classes of errors detected by the TL and TLX. Error notifications and the collection of error signatures are specified.

- The host or device may provide:
 - Additional error signature information for error events defined by this architecture
 - Additional error events that are implementation-specific.

Specification of these implementation-specific error signature and error event extensions might be found in either the host's platform architecture, the host's user's guide, or the manufacturer's documentation provided with the OpenCAPI device.

Implementation-specific fatal error events shall be summarized in the *AFU Fatal error detected* and the *Host Fatal error detected* error events specified by this specification. This specification provides these summary error events to expose the existence of fatal implementation-specific error events and are included in conformance tests. Note that conformance testing may not test the underlying implementation-specific error events, but shall ensure that the architecturally specified error events are tested.

Actions taken by hypervisors, operating systems, firmware, or device drivers when error notifications are asserted are beyond the scope of this architecture.

Error events are specified in *Table 9-1* on page 213 using the following format:

Error event name	Description of error event <ul style="list-style-type: none">• Action taken	
	Error signature:	The minimum set of information captured by a compliant design. <div>Engineering Note The error signatures specified in <i>Table 9-1</i> on page 213 shall be accessible and may be obtained for diagnostic use by an examination of hardware facilities and may require additional host- or device-specific software manipulation.</div>
	Error Class	Specifies the class and architecture conformance requirements.

Error classes are specified as follows:

- *Correctable error events* are error conditions that the hardware can recover without any loss of function, state, or data.
- *Fatal error events* are error conditions that result in the unrecoverable loss of function, state, or data. Continued use of the link and attached device might not be safe. A reset might be required to return the link and device to a safe operational state. The architecture does not place conformance requirements on an implementation once a fatal error event has occurred. That is, continued use of the link may occur and the results of operations after a fatal error are undefined.

The TL shall report the detection of a fatal error to the device using the method described by the [OpenCAPI DL specification](#).

The TLX shall report the detection of a fatal error to the host using the method described by the [OpenCAPI DL specification](#).

- *Non-fatal error events* are error conditions that affect the operation of a single transaction. The link is considered to be operationally safe. The results of the transaction might not be as intended. That is, the

Approved

results of the operation are undefined. Devices associated with the error might require a reset. Devices not associated with the error are not affected by the error. Continued use of the link may occur and the results of operations after a non-fatal error shall conform to the requirements of the architecture.

Error events are assigned error types and may be assigned an error subtype. These assignments are found in bold text in the description of the error event.

Error events are assigned an error class and a conformance requirement.

- *Required error events* are demanded by the architecture and shall be included in any architecture conformance testing. All error events specified as required shall be included in both the TL and TLX implementation unless otherwise specified.
- *Optional error events* are not required by the architecture. Careful reading of the description of the error events assigned as optional is strongly recommended since detection may be required due to architecturally required actions to be taken. Conformance testing may indicate the presence or absence of the hardware's capability to detect and report the error event.

7.1 Error events

The following error events are specified

acTag specified in a command is outside the configured specification set	acTag specified in the command points to an invalid entry	AFU Fatal error detected
Age out specified for xlate_touch.n	Bad BDF and PASID combination	Bad Cache State Transition
Bad data flit indication error	Bad data received	Bad opcode and template combination
Bad response received	Bad template x'00' format	Bad template usage
Control flit overrun	DCP credit under-run	Host Fatal error detected
Illegal return credit command location	log2_page_size specification in xlate_touch is bad.	Missing Metadata
PA specified is out of bounds	Posted command error	Reserved field not transmitted as 0
Reserved field value used	Reserved opcode used	Returned credit overflows credit counter
ta_req specified for xlate_touch.n	TL response timer expired	Unexpected data carrier
	Unsupported page size specified	Unsupported template format
VC credit under-run		

Approved

Table 7-1. Error event specification (Page 1 of 10)

Error event	Description
acTag specified in a command is outside the configured specification set	On receipt of a TLX command packet containing an acTag field, it is determined that the acTag is specified outside the configured specification set. <ul style="list-style-type: none"> The operation is aborted without changes to the machine state, and a malformed packet error type 2 event is asserted.
	<div> <div>Error signature:</div> <div>opcode(7:0), AFUTag(15:0)</div> <div> Engineering note When the TLX command is assign_actag, the AFUTag in the error signature is reserved. </div> </div>
	<div>Error Class:</div> Fatal/Required (TL only)
acTag specified in the command points to an invalid entry	On receipt of a TLX command packet containing an acTag field, the entry in the acTag table is examined and found to be marked invalid. The operation is aborted without changes to the machine state, and an address context error type 0 event is asserted.
	<div>Error signature:</div> opcode(7:0), AFUTag(15:0), acTag(11:0)
	<div>Error Class:</div> Fatal/Required (TL only)
AFU Fatal error detected	An AFU and device implementation-specific fatal error. This error is specified by the device manufacturer.
	<div>Error signature:</div> See the manufacturer's device documentation.
	<div>Error Class:</div> Fatal/Optional (TLX only)
Age out specified for xlate_touch.n	An xlate_touch.n is specified with a command flag of "age out". This is a nonsensical combination. The dot-n directive is ignored, and the operation proceeds to completion. <ul style="list-style-type: none"> An xlate_touch error type 1 event may be asserted.
	<div>Error signature:</div> AFUTag(15:0)
	<div>Error Class:</div> Non-fatal/Optional (TL only)
Bad BDF and PASID combination	On receipt of a TLX command packet containing an acTag field, the entry in the acTag table is found to be valid. The BDF and PASID specified are not valid for use. The operation is aborted without changes to the machine state, and an address context error type 1 event is asserted.
	<div>Error signature:</div> opcode(7:0), AFUTag(15:0), acTag(11:0)
	<div>Error Class:</div> Fatal/Required (TL only)
Bad Cache State Transition	A castout or castout.push command has specified an illegal cache state transition. Legal cache state transitions can be found in <i>Table 7-2 Cache state transition errors</i> on page 208. <ul style="list-style-type: none"> The cache state of the line is unchanged. A coherency error event is asserted.
	<div>Error signature:</div> opcode(7:0), dLength(1:0), cache_state(2:0) found in the command specification, host_tag(23:0), Initial cache state held by the host.
	<div>Error Class:</div> Fatal / Required (TL only)

Approved

Table 7-1. Error event specification (Page 2 of 10)

Error event	Description																								
Bad data flit indication error	<p>On the receipt of a control flit, the bad data flit field indicates that a data flit is bad and is located beyond the scope of the control flit. That is, the control flit indicates that n data flits follow, and the bad data flit field indicates that data flit n or above is in error. In the following valid combinations, an 'x' indicates that the field may take on either a 0 or 1 state.</p> <table border="1"> <thead> <tr> <th>Data run length</th><th>Bad data flit</th></tr> </thead> <tbody> <tr> <td>x'0'</td><td>'0000 0000'</td></tr> <tr> <td>x'1'</td><td>'0000 000x'</td></tr> <tr> <td>x'2'</td><td>'0000 00xx'</td></tr> <tr> <td>x'3'</td><td>'0000 0xxx'</td></tr> <tr> <td>x'4'</td><td>'0000 xxxx'</td></tr> <tr> <td>x'5'</td><td>'000x xxxx'</td></tr> <tr> <td>x'6'</td><td>'00xx xxxx'</td></tr> <tr> <td>x'7'</td><td>'0xxx xxxx'</td></tr> <tr> <td>x'8'</td><td>'xxxx xxxx'</td></tr> </tbody> </table> <ul style="list-style-type: none"> A malformed control flit error type 3 event is asserted. <p>The bad data flit information beyond the scope of the control flit is ignored.</p> <table border="1"> <tr> <td>Error signature:</td><td>The bad data flit and data run length fields are captured. These are found in the DL content of the control flit found in bits 11:0 as shown in <i>Section 5.1.1 DL content</i> on page 183.</td></tr> <tr> <td>Error Class:</td><td>Non-fatal/Optional (TL and TLX)</td></tr> </table>	Data run length	Bad data flit	x'0'	'0000 0000'	x'1'	'0000 000x'	x'2'	'0000 00xx'	x'3'	'0000 0xxx'	x'4'	'0000 xxxx'	x'5'	'000x xxxx'	x'6'	'00xx xxxx'	x'7'	'0xxx xxxx'	x'8'	'xxxx xxxx'	Error signature:	The bad data flit and data run length fields are captured. These are found in the DL content of the control flit found in bits 11:0 as shown in <i>Section 5.1.1 DL content</i> on page 183.	Error Class:	Non-fatal/Optional (TL and TLX)
Data run length	Bad data flit																								
x'0'	'0000 0000'																								
x'1'	'0000 000x'																								
x'2'	'0000 00xx'																								
x'3'	'0000 0xxx'																								
x'4'	'0000 xxxx'																								
x'5'	'000x xxxx'																								
x'6'	'00xx xxxx'																								
x'7'	'0xxx xxxx'																								
x'8'	'xxxx xxxx'																								
Error signature:	The bad data flit and data run length fields are captured. These are found in the DL content of the control flit found in bits 11:0 as shown in <i>Section 5.1.1 DL content</i> on page 183.																								
Error Class:	Non-fatal/Optional (TL and TLX)																								
Bad data received	<p>On processing data flits, a data flit is marked bad by the bad data flit indication field found in the control flit as described in <i>Section 5.1.1 DL content</i> on page 164.</p> <ul style="list-style-type: none"> A bad data flit error event may be asserted. The command or response associated with the bad data is provided with the data and the bad data indication. The bad data indication shall be propagated to the final destination of the data. It is strongly recommended that the error propagation be implemented in a fashion that allows for error isolation; that is, first error incidence reporting. <table border="1"> <tr> <td>Error signature:</td><td> Data is associated with: TLX response packet: opcode(7:0), CAPPTag(15:0). TLX command packet: opcode(7:0), acTag(11:0), AFUTag(15:0), or opcode(7:0), host_tag(23:0). TL response packet: opcode(7:0), AFUTag(15:0). TL command packet: opcode(7:0), CAPPTag(15:0) </td></tr> <tr> <td>Error Class:</td><td>Non-fatal/Optional (TL and TLX)</td></tr> </table>	Error signature:	Data is associated with: TLX response packet: opcode(7:0), CAPPTag(15:0). TLX command packet: opcode(7:0), acTag(11:0), AFUTag(15:0), or opcode(7:0), host_tag(23:0). TL response packet: opcode(7:0), AFUTag(15:0). TL command packet: opcode(7:0), CAPPTag(15:0)	Error Class:	Non-fatal/Optional (TL and TLX)																				
Error signature:	Data is associated with: TLX response packet: opcode(7:0), CAPPTag(15:0). TLX command packet: opcode(7:0), acTag(11:0), AFUTag(15:0), or opcode(7:0), host_tag(23:0). TL response packet: opcode(7:0), AFUTag(15:0). TL command packet: opcode(7:0), CAPPTag(15:0)																								
Error Class:	Non-fatal/Optional (TL and TLX)																								
Bad opcode and template combination	<p>The format of the control flit specified by the template is in error. Opcodes specified indicate packet sizes larger than allowed by the template found in the control flit. This error is detected in both the TL and TLX.</p> <p>All commands or responses identified in the control flit are aborted and do not cause any machine state changes, and a malformed control flit error type 2 event is asserted.</p> <table border="1"> <tr> <td>Error signature:</td><td> <ul style="list-style-type: none"> Template (5:0) found in the control flit. The opcode (7:0) found in the control flit where it was determined that the template packet size rules were violated. The slot location, a 4-bit field, where it was determined that the template packet size rules were violated. </td></tr> <tr> <td>Error Class:</td><td>Fatal/Required. (TL and TLX)</td></tr> </table>	Error signature:	<ul style="list-style-type: none"> Template (5:0) found in the control flit. The opcode (7:0) found in the control flit where it was determined that the template packet size rules were violated. The slot location, a 4-bit field, where it was determined that the template packet size rules were violated. 	Error Class:	Fatal/Required. (TL and TLX)																				
Error signature:	<ul style="list-style-type: none"> Template (5:0) found in the control flit. The opcode (7:0) found in the control flit where it was determined that the template packet size rules were violated. The slot location, a 4-bit field, where it was determined that the template packet size rules were violated. 																								
Error Class:	Fatal/Required. (TL and TLX)																								

Approved

Table 7-1. Error event specification (Page 3 of 10)

Error event	Description
Bad response received	<p>TL: On receipt of a TLX response packet, the CAPPTag is first examined to determine if a prior command has been issued using the CAPPTag found in the response packet. It is reported as a bad response received variant 0 if the CAPPTag has not been used. If the response packet is not a variant 0, the response opcode is checked to determine if it is a valid response for the command opcode used. It is reported as a bad response received variant 1 if it is not.</p> <hr/> <p>TLX: On receipt of a TL response packet, the AFUTag is first examined to determine if a prior command has been issued using the AFUTag found in the response packet. It is reported as a bad response receive variant 0 if the AFUTag has not been used. If the response packet is not a variant 0, the response opcode is checked to determine if it is a valid response for the command opcode used. It is reported as a bad response received variant 1 if it is not.</p> <hr/> <ul style="list-style-type: none"> The actions specified by the completion of the command due to a correct response are aborted, and a malformed packet error type 5 event is asserted. The state machine representing the command source is left in an undefined state. The undefined state of the machine is bounded, that is, the state is known to the implementation and the actions taken by the state machine in this architecturally undefined state is predictable by the implementation.
	<p>Error signature:</p> <p>TL: CAPPTag(15:0), variant(0). For a variant 1, the command opcode(7:0) is also provided. TLX: AFUTag(15:0), variant(0). For a variant 1, the command opcode(7:0) is also provided. In the error signature, the variant(0) field is set to the variant error type.</p> <ul style="list-style-type: none"> variant(0) = '0' when the error is variant 0. variant(0) = '1' when the error is variant 1.
	<p>Error Class:</p> <p>Fatal/Required (TL and TLX)</p>
Bad template x'00' format	<p>The format of the control flit, specified as using the x'00' template, does not match the template x'00' format. Slot 0 does not contain either a nop, or return_tl_x_credits (detected by the TLX), or return_tl_credits (detected by the TL), opcode.</p> <ul style="list-style-type: none"> Any commands or responses found in the control flit are aborted and do not cause any machine state changes. A malformed control flit error type 0 event is asserted.
	<p>Error signature:</p> <p>Slot 0 (27:0) contents</p>
	<p>Error Class:</p> <p>Fatal/Required (TL and TLX)</p>
Bad template usage	<p>Template x'08' is improperly used.</p> <ol style="list-style-type: none"> The template contains two 8-byte data fields, and the field starting in slot 0 is unused (the valid bit is set to 0). (TL and TLX) A malformed control flit error type 3 event is asserted. Any valid data is used. Command and response packets, if any, are not dropped. The data is associated with a fetch and swap operation (amo_rw, cmd_flag = {x'8'...x'A'}) (TL and TLX) A malformed control flit error type 4 event is asserted. Any valid data associated with the control flit is dropped.
	<p>Error signature:</p> <p>Template (5:0) found in the control flit.</p>
	<p>Error Class:</p> <ol style="list-style-type: none"> Non-fatal/Optional Fatal/Required

Approved

Table 7-1. Error event specification (Page 4 of 10)

Error event	Description
Control flit overrun	The destination is unable to accept a subsequent control flit. A possible cause is a violation of the control flit rate capability for the prior control flit's template. <ul style="list-style-type: none"> The incoming control flit is discarded. The machine state is unchanged. A control flit overrun error event is asserted.
	Error signature: None
	Error Class: Fatal/Required (TL and TLX)
DCP credit under-run	The consumer of DCP credits has issued more commands or responses that consume DCP credits than the number of DCP credits released. DCP credits correspond to data buffer resources in the data sink. This error detects when those resources have been over-run by the data source. Data carriers are discarded. The machine state is unchanged. TL: The TLX has consumed more DCP credits than the TL has released to the TLX. TLX: The TL has consumed more DCP credits than the TLX has released to the TL.
	Error signature DCP channel number(2:0)
	Error Class Fatal/Required (TL and TLX)
Host Fatal error detected	A TL and host implementation-specific fatal error. This error is specified by the host manufacturer.
	Error signature: See the manufacturer's device documentation.
	Error Class: Fatal/Optional (TL only)
Illegal return credit command location	TL: return_tl_credits shall be found only in the following slots based on the template used: x'07' slots (11:10) x'09' slots (11:10) x'0B slots(13:12) All other templates slots(1:0) TLX: Regardless of the template used, return_tlx_credits shall be found only in slots 1:0. <ul style="list-style-type: none"> The credit return, as specified by the command, may occur. A malformed packet error type 3 event shall be asserted.
	Error signature: Template (5:0) found in the control flit; slot where return credit opcode was found (3:0).
	Error Class: Fatal/Required (TL and TLX)
log ₂ _page_size specification in xlata_touch is bad.	This error is detected when an xlata_touch is specified with a cmd_flag specification of age-out, and the page size specified by the ATC entry found does not match the page size specified by the command's log ₂ _page_size field. <ul style="list-style-type: none"> See <i>Figure 2-1 Address translation sequence: xlata_touch</i> on page 111 for actions taken when there is a mismatch. An xlata_touch error type 0 event may be asserted.
	Error signature: acTag(11:0)
	Error Class: Non-fatal/Optional (TL only)

Approved

Table 7-1. Error event specification (Page 5 of 10)

Error event	Description
Missing Metadata	<p>The host and attached OpenCAPI device have been configured to use metadata, and metadata has not been provided for a command or response that is specified with immediate data.</p> <p>The assertion of this error excludes config_write and config_read operations. These operations tolerate the presence of metadata. That is, the use of metadata for config_write and config_read operations is not defined by this architecture and an error shall not be reported.</p> <p>When missing metadata is determined, the following actions may be used to allow the link to continue operation.</p> <p>TL: The host may</p> <ul style="list-style-type: none"> provide non-destructive metadata to the data block. The value used is beyond the scope of the TL architecture. mark the data as bad. <p>TLX: The OpenCAPI device may</p> <ul style="list-style-type: none"> provide non-destructive metadata to the data block. The value used is beyond the scope of the TL architecture. mark the data as bad. <p>A Missing metadata error is asserted.</p>
	<p>Error signature: TL: AFU_MTag; TLX: CAPPTag.</p>
	<p>Error Class: Non-fatal/Optional (TL and TLX)</p>
PA specified is out of bounds	<p>A command specifies a PA that is determined to be out of bounds for the AFU_M. For TL commands, the host has specified a PA that is outside the AFU_{M1} PA range.</p> <ul style="list-style-type: none"> The operation is aborted without changes to the machine state. A PA specification error event is asserted.
	<p>Error signature: opcode(7:0), PA(63:0)</p>
	<p>Error Class: Fatal/Required (TLX)</p>

Approved

Table 7-1. Error event specification (Page 6 of 10)

Error event	Description	
Posted command error	<p>The TL detects an error associated with a TLX posted command that is not covered by other errors defined in this specification.</p> <ul style="list-style-type: none"> The operation is aborted. The machine state is undefined. A Posted command error event is asserted. 	
	<p>Error signature:</p>	<p>AFUtag(15:0), Type(3:0). The Type field is specified as:</p> <p>x'0' Reserved</p> <p>x'9' Unsupported operand length</p> <p>x'B' Bad TA specification. The TA specified by the command is not naturally aligned.</p> <p>x'C' The {TA, address context} is not recognized by the host.</p> <p>x'E' Failed for unspecified reason</p> <p>x'F' Bad TA specification. The {TA, address context} specified by the command is does not have write permission as required by the command.</p> <p>All other code points are reserved.</p> <div> <p>Developer Note</p> <p>This error is detected only by the TL. There is no TLX detection of a posted TL command error. The reasoning is as follows:</p> <p>xlata_done, intrap_rdy, rd_pf and force_evict are posted TL commands that could be found in error by the TLX.</p> <ul style="list-style-type: none"> rd_pf is explicitly specified to not report errors when the TL command packet is found to malformed. The remaining commands report a <i>Reserved field value used</i> error when the operand values are out of bounds or otherwise reserved. </div>
	Error Class:	Fatal/Required (TL)
Reserved field not transmitted as 0	<p>On the receipt of a command or response packet, it is determined that a field specified by the architecture as reserved does not contain 0.</p> <ul style="list-style-type: none"> The packet is used. That is, the operation specified by the command or response occurs normally. This is an architecture conformance violation. A malformed packet error type 4 event is asserted. 	
	Error signature:	None.
	Error Class:	Non-fatal/Optional (TL and TLX)

Approved

Table 7-1. Error event specification (Page 7 of 10)

Error event	Description																																									
Reserved field value used	<p>On the receipt of a command or response packet, it is determined that a field specification contains an architecturally reserved value. If multiple fields contain reserved values, only one field is reported.</p> <ul style="list-style-type: none">The operation is aborted and the machine state is unchanged. A malformed packet error type 1 event is asserted. <p>The following fields have reserved values. Detection occurs at the receiver of the command or response packet.:</p> <ul style="list-style-type: none"><i>cache_state</i>. Detected by TL and TLX.<i>cmd_flag</i>. Detected by TL and TLX.<i>dLength</i>. Detected by TL and TLX.<i>dPart(1:0)</i> or <i>dPart(2:0)</i> detected by TL and TLX.<i>host_tag</i> specified out of negotiated range. Detected by TL and TLX.<i>pLength</i>. Detected by TL and TLX.<i>Resp_code</i>. Detected by TL and TLX.																																									
	Error signature:	opcode(7:0), starting (LSb) field offset within the packet. For example, the acTag in a rd_wntic TLX command packet has a field offset of 24.																																								
	Error Class	Fatal/Required (TL and TLX)																																								
Reserved opcode used	<p>On the receipt of a command or response packet, the opcode field is examined and found to be a value reserved by the architecture.</p> <ul style="list-style-type: none">The packet is dropped, the machine state is unchanged. A malformed packet error type 0 event is asserted.																																									
	Error signature:	opcode(7:0)																																								
	Error Class:	Fatal/Required (TL and TLX)																																								
Returned credit overflows credit counter	<p>On processing of a return_tl_credits or return_tlx_credits response packet, it is determined that the addition of the credits specified by the response will cause the counter to overflow.</p> <ul style="list-style-type: none">The counter may increment and is allowed to saturate. That is, the counter shall not wrap. A credit return error event is asserted.																																									
	Error signature:	<p>opcode(7:0), specification of the counter or counters associated with the error using the following format:</p> <table><tr><td>TL:</td><td>R</td><td>R</td><td>dcp.1</td><td>dcp.0</td><td>R</td><td>R</td><td>vc.2</td><td>vc.1</td><td>vc.0</td></tr><tr><td>TLX:</td><td>dcp.3</td><td>dcp.2</td><td>R</td><td>dcp.0</td><td>R</td><td>vc.3</td><td>vc.2</td><td>R</td><td>vc.0</td></tr><tr><td></td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td></tr><tr><td></td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr></table>	TL:	R	R	dcp.1	dcp.0	R	R	vc.2	vc.1	vc.0	TLX:	dcp.3	dcp.2	R	dcp.0	R	vc.3	vc.2	R	vc.0		↓	↓	↓	↓	↓	↓	↓	↓	↓		8	7	6	5	4	3	2	1	0
	TL:	R	R	dcp.1	dcp.0	R	R	vc.2	vc.1	vc.0																																
TLX:	dcp.3	dcp.2	R	dcp.0	R	vc.3	vc.2	R	vc.0																																	
	↓	↓	↓	↓	↓	↓	↓	↓	↓																																	
	8	7	6	5	4	3	2	1	0																																	
Error Class:	Fatal/Required (TL and TLX)																																									
ta_req specified for xlate_touch.n	<p>An xlate_touch.n is specified with a command flag of ta_req. This is a nonsensical combination since the dot-t indicates that an ATC entry is not formed, while the ta_req directive indicates that a translated address and a pinned ATC entry is required.</p> <p>The dot-n directive is ignored and the operation proceeds to completion.</p> <ul style="list-style-type: none">An xlate_touch error type 2 event may be asserted.																																									
	Error signature:	AFUtag(15:0)																																								
	Error Class:	Non-fatal Optional (TL only)																																								

Approved

Table 7-1. Error event specification (Page 8 of 10)

Error event	Description	
TL response timer expired	<p>The TL or host has determined that the TLX or AFU has not responded within a host architecture specified time out period.</p> <ul style="list-style-type: none"> Changes to the machine state are undefined. A Response time out error event is asserted. <p>The value specified is host specific and should be found in the host's platform architecture documentation.</p>	
	Error signature:	<p>Type(3:0)</p> <p>x'0' General. Non-posted command time out.</p> <p>x'1' force_evict waiting on castout or castout.push.</p> <p>x'3' synonym_detected waiting on synonym_done</p> <ul style="list-style-type: none"> For type x'1' and type x'3,' it is strongly recommended that the error signature include the host_tag.
	Error Class:	Fatal / Required (TL only)

Approved

Table 7-1. Error event specification (Page 9 of 10)

Error event	Description																												
Unexpected data carrier	<p>The destination has detected that the accumulated number of data carriers has exceeded the amount of data expected.</p> <p>The expected data count is determined by an examination of the commands and responses received that specify immediate data. The ordering requirements specifying commands or responses with immediate data are received before the data is found in <i>Section 5.1.3 Data transport, order, and alignment</i> on page 184.</p> <hr/>																												
	<p>TL:</p> <p>TLX packets received by the TL with immediate data specify the amount of data expected from the TLX.</p> <p>TLX packets that specify immediate data:</p> <table><tr><td>mem_rd_response</td><td>dma_w</td><td>dma_w.n</td><td>dma_w.be</td></tr><tr><td>dma_w.be.n</td><td>dma_pr_w</td><td>dma_pr_w.n</td><td>amo_rw</td></tr><tr><td>amo_rw.n</td><td>amo_w</td><td>amo_w.n</td><td>intrp_req.d</td></tr><tr><td>mem_rd_response.ow</td><td></td><td>mem_rd_response.xw</td><td></td></tr><tr><td>castout.push</td><td>intrp_req.d</td><td>dma_w.t.p</td><td>dma_w.t.p.s</td></tr><tr><td>dma_w.be.t.p</td><td>dma_w.be.t.p.s</td><td>dma_pr_w.t.p</td><td>dma_pr_w.t.p.s</td></tr><tr><td>amo_rw.t.s</td><td>amo_w.t.p</td><td>amo_w.t.p.s</td><td></td></tr></table> <hr/>	mem_rd_response	dma_w	dma_w.n	dma_w.be	dma_w.be.n	dma_pr_w	dma_pr_w.n	amo_rw	amo_rw.n	amo_w	amo_w.n	intrp_req.d	mem_rd_response.ow		mem_rd_response.xw		castout.push	intrp_req.d	dma_w.t.p	dma_w.t.p.s	dma_w.be.t.p	dma_w.be.t.p.s	dma_pr_w.t.p	dma_pr_w.t.p.s	amo_rw.t.s	amo_w.t.p	amo_w.t.p.s	
	mem_rd_response	dma_w	dma_w.n	dma_w.be																									
	dma_w.be.n	dma_pr_w	dma_pr_w.n	amo_rw																									
amo_rw.n	amo_w	amo_w.n	intrp_req.d																										
mem_rd_response.ow		mem_rd_response.xw																											
castout.push	intrp_req.d	dma_w.t.p	dma_w.t.p.s																										
dma_w.be.t.p	dma_w.be.t.p.s	dma_pr_w.t.p	dma_pr_w.t.p.s																										
amo_rw.t.s	amo_w.t.p	amo_w.t.p.s																											
<p>TLX:</p> <p>TL packets received by the TLX with immediate data specify the amount of data expected by the TL.</p> <p>TL packets that specify immediate data:</p> <table><tr><td>read_response</td><td>write_mem</td><td>write_mem.be</td><td>pr_wr_mem</td></tr><tr><td>config_write</td><td></td><td></td><td></td></tr><tr><td>read_response.ow</td><td>read_response.xw</td><td></td><td></td></tr><tr><td>cl_rd_resp</td><td>cl_rd_resp.ow</td><td>amo_rw</td><td>amo_w</td></tr></table> <hr/>	read_response	write_mem	write_mem.be	pr_wr_mem	config_write				read_response.ow	read_response.xw			cl_rd_resp	cl_rd_resp.ow	amo_rw	amo_w													
read_response	write_mem	write_mem.be	pr_wr_mem																										
config_write																													
read_response.ow	read_response.xw																												
cl_rd_resp	cl_rd_resp.ow	amo_rw	amo_w																										
	<ul style="list-style-type: none">The unexpected data carrier's contents may be discarded. An unexpected data carrier error event is asserted. <div><p>Developer Note</p><p>Expected data can be counted by determining the number of DCP credits required to send the data as specified by the command or response. Counting DCP credits can be split out by data credit pool number or aggregated.</p><p>Counts are incremented as expected-data-information, as described above, is observed. Counts are decremented as the data is received. That is, the DCP count of expected data is decremented as the data carriers arrive. When the count drops below zero, an unexpected data carrier error is detected.</p><p>Other implementation specific methods may also be used.</p></div>																												
	<table><tr><td>Error signature:</td><td>None</td></tr><tr><td>Error Class:</td><td>Fatal/Required (TL and TLX)</td></tr></table>	Error signature:	None	Error Class:	Fatal/Required (TL and TLX)																								
Error signature:	None																												
Error Class:	Fatal/Required (TL and TLX)																												

Approved

Table 7-1. Error event specification (Page 10 of 10)

Error event	Description
Unsupported page size specified	<p>Page size support is determined during initialization of the OpenCAPI device. The page size capabilities of the host and the AFU are compared and the intersection of the page sizes support are used. The <code>log₂_page_size</code> field is specified in the following commands and is checked for legal page size values.</p> <p>Detected by the TLX on receipt of: touch_resp, touch_resp.t, kill_xlate.</p> <p>Detected by TL on receipt of: xlate_release, xlate_touch(age out request only).</p> <ul style="list-style-type: none"> The operation is aborted and changes to the machine state are undefined. A malformed packet error type 6 is asserted.
	Error signature: None
	Error Class: Fatal/Required
Unsupported template format	<p>An unsupported template is specified in a received control flit.</p> <ul style="list-style-type: none"> Any commands or responses identified in the control flit are aborted. Changes to the machine state are undefined. A malformed control flit error type 1 event is asserted.
	Error signature: Template (5:0) found in the control flit.
	Error Class: Fatal/Required (TL and TLX)
VC credit under-run	<p>The consumer of VC credits has issued more commands or responses that consume VC credits than the number of VC credits released. VC credits correspond to command queuing resources in the command receiver. This error detects when those resources have been over-run by the command source. Commands and responses associated with the VC channel or channels are discarded. Changes to the machine state are undefined.</p> <p>TL: The TLX has consumed more VC credits than the TL has released to the TLX.</p> <p>TLX: The TL has consumed more VC credits than the TLX has released to the TL.</p>
	Error signature VC channel number(2:0)
	Error Class Fatal/Required

Table 7-2 shows all combinations of **castout** and **castout.push** commands and the error conditions that are reported by *Bad Cache State Transition*. An X indicates a legal transition, and “error” indicates an illegal one which shall assert the error.

Table 7-2. Cache state transition errors (Page 1 of 2)

Host proxy cache state	cache_state(2:0)	castout, cmd_flag='0011'	castout.push	Comments
M	M	X	X	update / write through
	E	error	X	clean
	S	error	error	Error
	E _I	error	X	Down grade and retain
	I	error	X	Eviction
E	M	X	X	update / write through
	E	X	X	update / write through
	S	error	error	Error
	E _I	X	X	Down grade or clean and retain
	I	X	X	Eviction

Approved

Table 7-2. Cache state transition errors (Page 2 of 2)

Host proxy cache state	cache_state(2:0)	castout, cmd_flag='0011'	castout.push	Comments
S	M	error	error	Error
	E	error	error	Error
	S	error	error	error
	E _I	error	error	Error
	I	X	error	Eviction
E _I	M	X	X	update / write through
	E	error	X	write through
	S	error	error	Error
	E _I	X	X	write through
	I	X	X	Eviction
I	M	error	error	Error
	E	error	error	Error
	S	error	error	Error
	E _I	error	error	Error
	I	error	error	Error

8. OpenCAPI profiles

A device shall use an OpenCAPI profile to specify groups of commands, responses, and templates supported by the device. Each row in the following tables identifies an architectural feature, and each column indicates the content of a profile. The full specification of a profile is comprised of the same column from all tables in this section.

Within each profile a feature is marked using the notation found in *Table 8-1*. The specification of the compliance notation is taken from the view of the consumer of the command or response packet. That is, a command that is specified as mandatory requires that the consumer of the command shall process and execute the command per the architecture specification. Since the architecture specifies any responses or errors that are reported, those features become mandatory as well.

Table 8-1. Feature compliance requirement notation

Support requirement notation	Description
	(blank / empty) No conformance requirement, no recommendation guidance provided.
M	Mandatory
M.c2	Mandatory when the AFU is C2. Otherwise it is unsupported (U).
M.cx	Mandatory when the AFU is C1 or C2. Otherwise it is unsupported (U).
M.ir	Mandatory when the AFU issues any form of intrp_req or wake_host_thread . Otherwise it is unsupported (U).
M.mx	Mandatory when the AFU is M1. Otherwise it is unsupported (U).
M.sc	Mandatory when the AFU issues the sync command. Otherwise it is unsupported (U).
M.ta	Mandatory when the AFU supports the use of translated addresses (TA). Otherwise it is optional (O) or unsupported (U) as specified by the table note.
M.tp	Mandatory when the required template is supported by the TL and TLX. Otherwise it is unsupported (U).
M.wht	Mandatory when the TLX issues wake_host_thread . Otherwise it is unsupported (U).
M.xt	Mandatory when xlate_touch is issued by the TLX. Otherwise it is unsupported (U).
O	Optional. This feature may be supported. Conformance evaluation to determine the presence of the feature and when present shall test its architectural compliance.
O.E	Compliance specification for endianness support. Either big or little endian data formats used in atomic* class commands may be supported. An implementation shall support one of the formats.
U	Unsupported. This feature is not included in conformance evaluation. Use of a command or response noted as unsupported may result in a fatal error event.

Approved

Table 8-2 specifies the compliance requirements for the TLX and AFU accepting and processing a command from the TL and host.

Table 8-2. Profile specifications for TL commands

TL command	Device interface class, OpenCAPI 3.0	OMI, OpenCAPI 3.1	Device interface class, OpenCAPI 4.0
amo_rd	U	U	O
amo_rw	U	U	O
amo_w	U	U	O
config_read	M	M	M
config_write	M	M	M
disable_atc	U	U	M.ta ¹
disable_cache	U	U	M.c2
enable_atc	U	U	M.ta ¹
enable_cache	U	U	M.c2
force_evict	U	U	M.c2
intrap_rdy	M.ir	M.ir	M.ir
kill_xlate	U	U	M.ta ¹
mem_cntl	U	O	O
nop	M	M	M
pad_mem	U	O	O
pr_rd_mem	M,mx	M	M.mx
pr_wr_mem	M.mx	M	M.mx
rd_mem	M.mx	M	M.mx
rd_pf	U	M	M.mx
write_mem	M.mx	M	M.mx
write_mem.be	M.mx	M	M.mx
xlate_done	M.cx	U	M.cx
Notes			
1. When the device does not support translated addresses (TA), support is Optional (O).			

Table 8-3 specifies the compliance requirements for the TL and host accepting and processing a command from the TLX and AFU.

Table 8-3. Profile specifications for TLX commands (Page 1 of 3)

TLX command	Device interface class, OpenCAPI 3.0	OMI, OpenCAPI 3.1	Device interface class, OpenCAPI 4.0
amo_rd	M	U	M
amo_rd.n	M	U	M
amo_rd.t	M	U	M
amo_rd.t.s	M	U	M

Approved

Table 8-3. Profile specifications for TLX commands (Page 2 of 3)

TLX command	Device interface class, OpenCAPI 3.0	OMI, OpenCAPI 3.1	Device interface class, OpenCAPI 4.0
amo_rw	M	U	M
amo_rw.n	M	U	M
amo_rw.t	M	U	M
amo_rw.t.s	M	U	M
amo_w	M	U	M
amo_w.n	M	U	M
amo_w.t.p	M	U	M
amo_w.t.p.s	M	U	M
assign_actag	M	M	M
castout	U	U	M
castout.push	U	U	M
dma_pr_w	M	U	M
dma_pr_w.n	M	U	M
dma_pr_w.t.p	U	U	M
dma_pr_w.t.p.s	U	U	M
dma_w	M	U	M
dma_w.be	M	U	M
dma_w.be.n	M	U	M
dma_w.be.t.p	U	U	M
dma_w.be.t.p.s	U	U	M
dma_w.n	M	U	M
dma_w.t.p	U	U	M
dma_w.t.p.s	U	U	M
intrap_req	M	M	M
intrap_req.d	M	M	M
intrap_req.d.s	U	U	M
intrap_req.s	U	U	M
nop	M	M	M
pr_rd_wnitc	M	U	M
pr_rd_wnitc.n	M	U	M
pr_rd_wnitc.t	U	U	M
pr_rd_wnitc.t.s	U	U	M
rd_wnitc	M	U	M
rd_wnitc.n	M	U	M
rd_wnitc.t	U	U	M
rd_wnitc.t.s	U	U	M

Approved

Table 8-3. Profile specifications for TLX commands (Page 3 of 3)

TLX command	Device interface class, OpenCAPI 3.0	OMI, OpenCAPI 3.1	Device interface class, OpenCAPI 4.0
read_me	U	U	M
read_me.t	U	U	M
read_mes	U	U	M
read_mes.t	U	U	M
read_s	U	U	M
read_s.t	U	U	M
sync	U	U	M
synonym_done	U	U	M
upgrade_state	U	U	M
upgrade_state.t	U	U	M
wake_host_thread	M	U	M
wake_host_thread.s	U	U	M
xlate_release	U	U	M
xlate_touch	M	U	M
xlate_touch.n	M	U	M

Table 8-4 specifies the compliance requirements for the TLX and AFU accepting and processing a response from the TL and host. Note that in most cases, TL responses are due to TLX commands issued to the host.

Table 8-4. Profile specifications for TL responses (Page 1 of 2)

TL response	Device interface class, OpenCAPI 3.0	OMI, OpenCAPI 3.1	Device interface class, OpenCAPI 4.0
cl_rd_resp	U	U	M.c2
cl_rd_resp.ow ²	U	U	M.c2, M.tp
intrp_resp	M.ir	M.ir	M.ir
nop	M	M	M
read_failed	M.cx	U	M.cx
read_response	M.cx	U	M.cx
read_response.ow ³	U	U	M.cx, M.tp
read_response.xw ⁴	U	U	M.cx, M.tp
return_tlx_credits	M	M	M
sync_done	U	U	M.sc
synonym_detected	U	U	M.c2
touch_resp	M.xt	U	M.xt
touch_resp.t	U	U	M.ta ¹
upgrade_resp	U	U	M.c2
wake_host_resp	M.wht	U	M.wht

Approved

Table 8-4. Profile specifications for TL responses (Page 2 of 2)

TL response	Device interface class, OpenCAPI 3.0	OMI, OpenCAPI 3.1	Device interface class, OpenCAPI 4.0
write_failed	M.cx	U	M.cx
write_response	M.cx	U	M.cx
Notes 1. When the device does not support translated addresses (TA), support is Optional (O). 2. Use of cl_rd_resp.ow is dependent on support of either templates x'07' or x'09'. 3. Use of read_response.ow is dependent on support of either templates x'07' or x'09'. 4. Use of read_response.xw is dependent on support of template x'08'.			

Table 8-5 specifies the compliance requirements for the TL and the host accepting and processing a response from the TLX and AFU. Note that in most cases, TLX responses are due to TL commands issued to the TLX.

Table 8-5. Profile specifications for TLX responses

TLX response	Device interface class, OpenCAPI 3.0	OMI, OpenCAPI 3.1	Device interface class, OpenCAPI 4.0
atc_disabled	U	U	M
atc_enabled	U	U	M
cache_disabled	U	U	M
cache_enabled	U	U	M
kill_xlate_done	U	U	M
mem_cntl_done	U	O	O
mem_rd_fail	M	M	M
mem_rd_response	M	M	M
mem_rd_response.ow^a	U	M.tp	M.tp
mem_rd_response.xw^b	U	M.tp	M.tp
mem_wr_fail	M	M	M
mem_wr_response	M	M	M
nop	M	M	M
return_tl_credits	M	M	M

a. Use of **mem_rd_response.ow** is dependent on support of either templates x'07' or x'09'.

b. Use of **mem_rd_response.xw** is dependent on support of template x'08'.

Table 8-6 and Table 8-7 add template capability specifications to profiles. As discussed in Section 6 TL and TLX template specifications on page 189, the host's platform architecture provides additional information about how receive and transmit capabilities are resolved between the host and the attached OpenCAPI device. Other than the mandatory support for transmitting and receiving template x'00' template control flits, the profile specifications for templates provides guidance as to the recommended templates an implementation should support and is not a conformance requirement. The templates marked as Optional are recom-

Approved

mended. See *Section 6.1 TLX receive and TL transmit template capability specification* on page 191 and *Section 6.2 TL receive and TLX transmit template capability specification* on page 194. All host and devices shall support template x'00'.

Table 8-6 specifies the requirements and recommendations for the

- TLX to accept a control flit using the specified template.
- TL to transmit a control flit using the specified template.

Table 8-6. Profile specifications for TLX receive/TL transmit templates

TLX receive/TL transmit template	Device interface class, OpenCAPI 3.0	OMI, OpenCAPI 3.1	Device interface class, OpenCAPI 4.0
x'00'	M	M	M
x'01'	O	O	O
x'02'	O		O
x'03'	O		O
x'04	U	O	
x'05'	U		
x'06'	U		
x'07' ^a	U	O	
x'08' ^b	U		
x'09' ^c	U		
x'0A' ^d	U	O	
x'0B' ^e	U		
Template notes: To support metadata, an implementation must support one of the following templates: x'04, x'05', or x'06'.			

a. Specifies a 32-byte data carrier. This template is used to support dot-ow response forms.

b. Specifies two 8-byte data carriers. This template is used to support dot-xw response forms.

c. Specifies a 32-byte data carrier. This template is used to support dot-ow response forms.

d. Specifies a 32-byte data carrier with extended-metadata.

e. Specifies a 32-byte data carrier with extended-metadata.

Table 8-7 specifies the requirements and recommendations for the

- TL to accept a control flit using the specified template.
- TLX to transmit a control flit using the specified template.

Table 8-7. Profile specifications for TL receive/TLX transmit templates (Page 1 of 2)

TL receive/TLX transmit template	Device interface class, OpenCAPI 3.0	OMI, OpenCAPI 3.1	Device interface class, OpenCAPI 4.0
x'00'	M	M	M
x'01'	O	O	O
x'02'	O		O
x'03'	O		O

Table 8-7. Profile specifications for TL receive/TLX transmit templates (Page 2 of 2)

TL receive/TLX transmit template	Device interface class, OpenCAPI 3.0	OMI, OpenCAPI 3.1	Device interface class, OpenCAPI 4.0
x'04	U		
x'05 ^a	U	O	
x'06 ^a	U		
x'07 ^a	U		
x'08 ^b	U		
x'09 ^c	U	O	
x'0A ^d	U		
x'0B ^e	U	O	

a. Specifies a 32-byte data carrier. This template is used to support dot-ow response forms.

b. Specifies two 8-byte data carriers. This template is used to support dot-xw response forms.

c. Specifies a 32-byte data carrier. This template is used to support dot-ow response forms.

d. Specifies a 32-byte data carrier with extended-metadata.

e. Specifies a 32-byte data carrier with extended-metadata.

Table 8-8 specifies the operation modes recommended to be supported by the host and the OpenCAPI device for different interface classifications and is not a conformance requirement. The operation modes marked as Optional are recommended. The definition of the host operation modes are found in *Section 1.2 Host operation modes* on page 28.

Table 8-8. Profile specifications host operation modes

AFU type	Device interface class, OpenCAPI 3.0	OMI, OpenCAPI 3.1	Device interface class, OpenCAPI 4.0
AFU _{C0}	O		O
AFU _{C1}	O	O	O
AFU _{C2}	U	U	O
AFU _{M0}	O		O
AFU _{M1}	O	M	O

Table 8-9 adds page size support specification to profiles. The profile specification for page size provides guidance as to the recommended page sizes supported by the host's and AFU's ATC. Address translation and ATC are discussed in *Section 1.8* on page 45. Support for a 4K page size is required for the host and required for the AFU only when the AFU manages an ATC. Page sizes marked as Optional are recommended.

Approved

Table 8-9. Profile specifications supported page size

page size	Device interface class, OpenCAPI 3.0	OMI, OpenCAPI 3.1	Device interface class, OpenCAPI 4.0
4K	O		M.ta ¹
64K	O		O
2M			O
Notes			
1. When the device does not support translated addresses (TA), support is Optional (O).			

Table 8-10 specifies the compliance requirements for the TLX and AFU accepting and processing commands and responses from the TL and host with the dLength specified.

Table 8-10. Profile specifications supported dLength by TLX

dLength specification	Device interface class, OpenCAPI 3.0	OMI, OpenCAPI 3.1	Device interface class, OpenCAPI 4.0
64	M	M	M
128	M	M	M
256	O	O	O

Table 8-11 specifies the compliance requirements for the TL and host accepting and processing commands and responses from the TLX and AFU with dLength specified.

Table 8-11. Profile specifications supported dLength by TL

dLength specification	Device interface class, OpenCAPI 3.0	OMI, OpenCAPI 3.1	Device interface class, OpenCAPI 4.0
64	M	M	M
128	M	M	M
256	M	M	M

Table 8-12 specifies the compliance requirements for the TL and host accepting and processing atomic.* class commands based on the endianness of the data. The *E* field in the command specifies the endianness of the data. See Table 8-3 for the compliance requirements for the TL and host accepting and processing these commands.

Table 8-12. Profile specifications support of endianness data format by the TL (Page 1 of 2)

TLX atomic* class command	Device interface class, OpenCAPI 3.0		OMI, OpenCAPI 3.1		Device interface class, OpenCAPI 4.0	
	E=0	E=1	E=0	E=1	E=0	E=1
amo_rd	O.E	O.E	U	U	O.E	O.E
amo_rd.n	O.E	O.E	U	U	O.E	O.E
amo_rd.t	O.E	O.E	U	U	O.E	O.E
amo_rd.t.s	O.E	O.E	U	U	O.E	O.E
amo_rw	O.E	O.E	U	U	O.E	O.E
amo_rw.n	O.E	O.E	U	U	O.E	O.E

Approved

Table 8-12. Profile specifications support of endianness data format by the TL (Page 2 of 2)

TLX atomic* class command	Device interface class, OpenCAPI 3.0		OMI, OpenCAPI 3.1		Device interface class, OpenCAPI 4.0	
	E=0	E=1	E=0	E=1	E=0	E=1
amo_rw.t	U	U	U	U	O.E	O.E
amo_rw.t.s	U	U	U	U	O.E	O.E
amo_w	U	U	U	U	O.E	O.E
amo_w.n	U	U	U	U	O.E	O.E
amo_w.t.p	U	U	U	U	O.E	O.E
amo_w.t.p.s	U	U	U	U	O.E	O.E

Table 8-13 specifies the compliance requirements for the TLX and the AFU accepting and processing atomic.* class commands based on the endianness of the data. The *E* field in the command specifies the endianness of the data. See Table 8-2 for the compliance requirements for the TLX and AFU accepting and processing these commands.

Table 8-13. Profile specifications support of endianness data format by the TLX

TL atomic* class command	Device interface class, OpenCAPI 3.0		OMI, OpenCAPI 3.1		Device interface class, OpenCAPI 4.0	
	E=0	E=1	E=0	E=1	E=0	E=1
amo_rd	U	U	U	U	M	M
amo_rw	U	U	U	U	M	M
amo_w	U	U	U	U	M	M

Appendix A. AP (TLX) command transaction diagrams

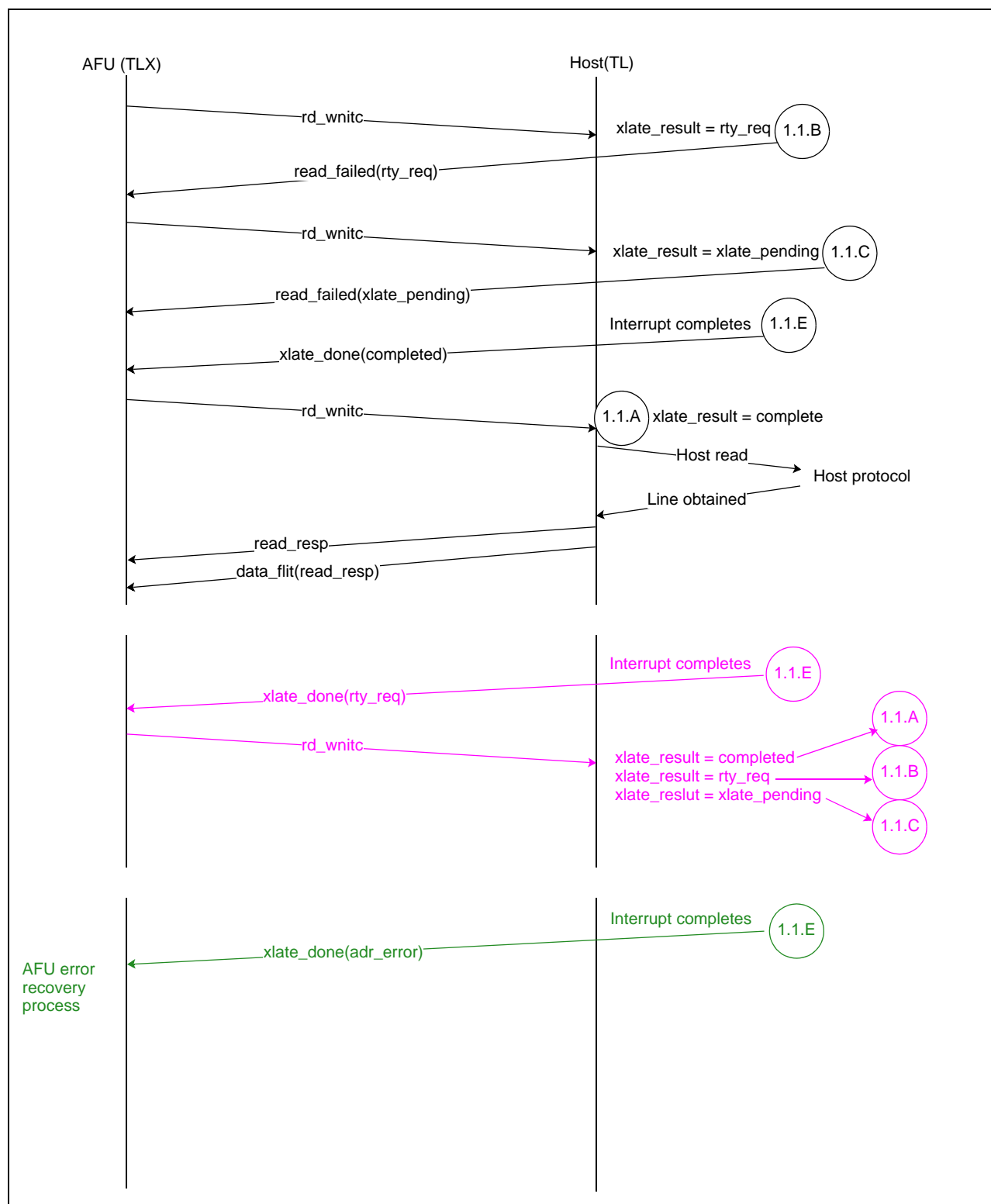
This section contains figures that illustrate AP command flows and TLX and TL interaction.

Rules:

1. Commands received at the TL are not serviced until all data, if any, specified by the AP command has arrived.

A.1 AFU read with no intent to cache; 128 bytes

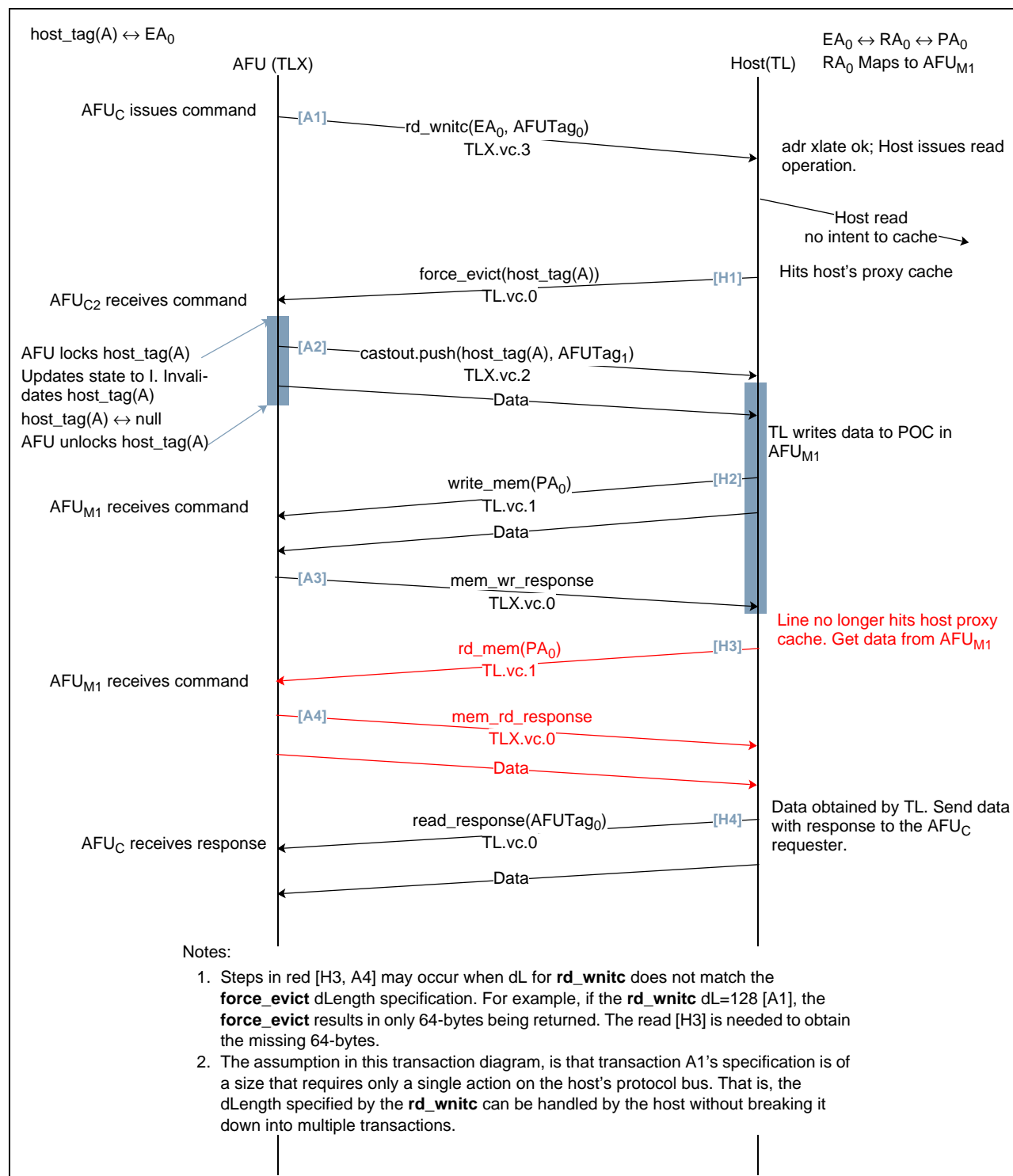
- For the dot-s form of the command, dispatch is stopped until all prior commands associated with the VC, BDF,PASID, and stream_id have completed on the host processor bus. See the description of **presync** on page 23.

Figure A-1. TLX and TL interaction: *rd_wnitc*

Approved

A.2 TLX read with no intent to cache hits device co-located AFU_{C2} and AFU_{M1}

Figure A-2. TLX **rd_wnltc** hits AFU_{C2} and AFU_{M1} TL issues commands to complete operation



A.3 AFU DMA write; non-posted; 128 bytes

- For the dot-s form of the command, dispatch is stopped until all prior commands associated with the VC, BDF,PASID, and stream_id have completed on the host processor bus. See the description of **presync** on page 23.

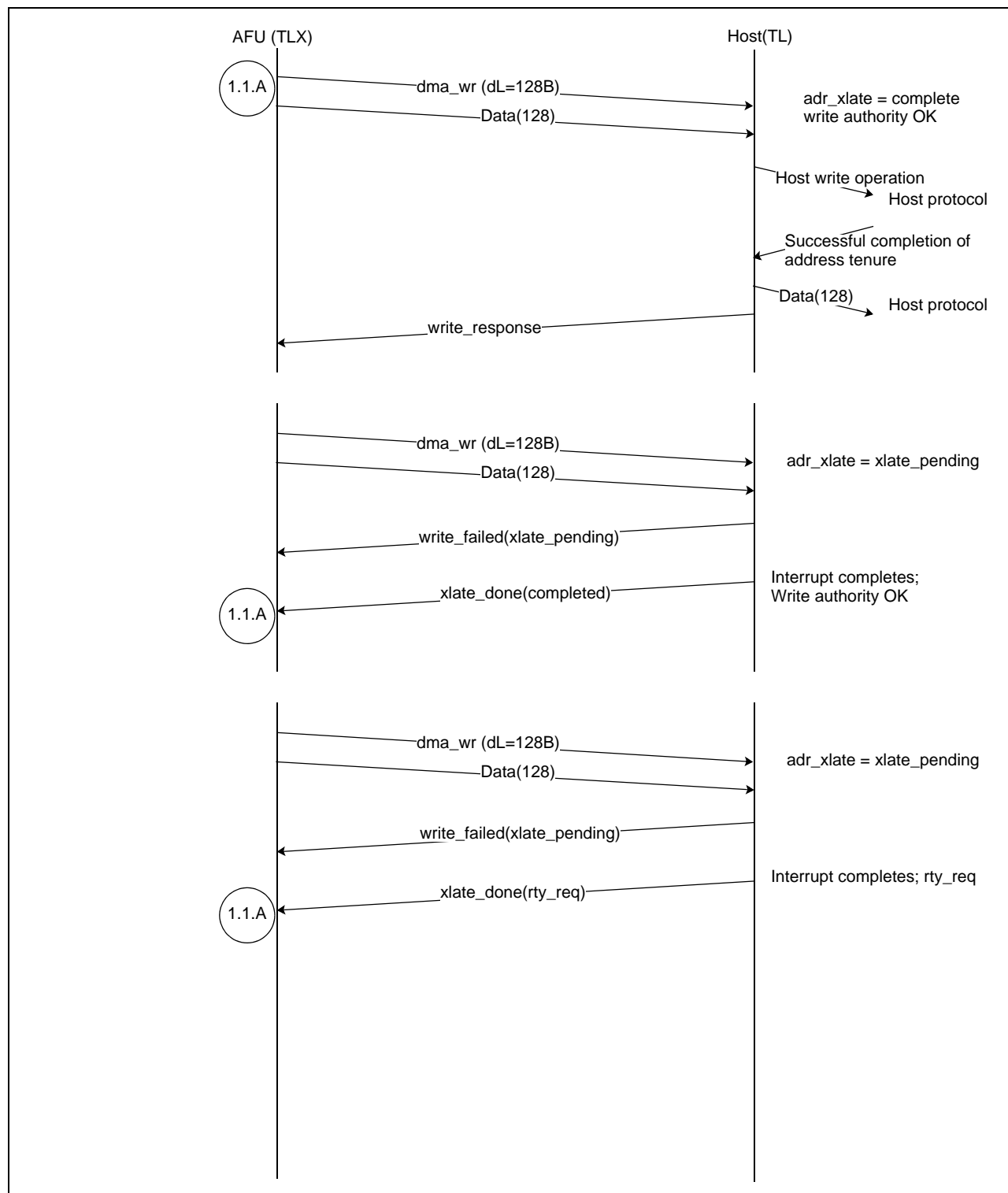
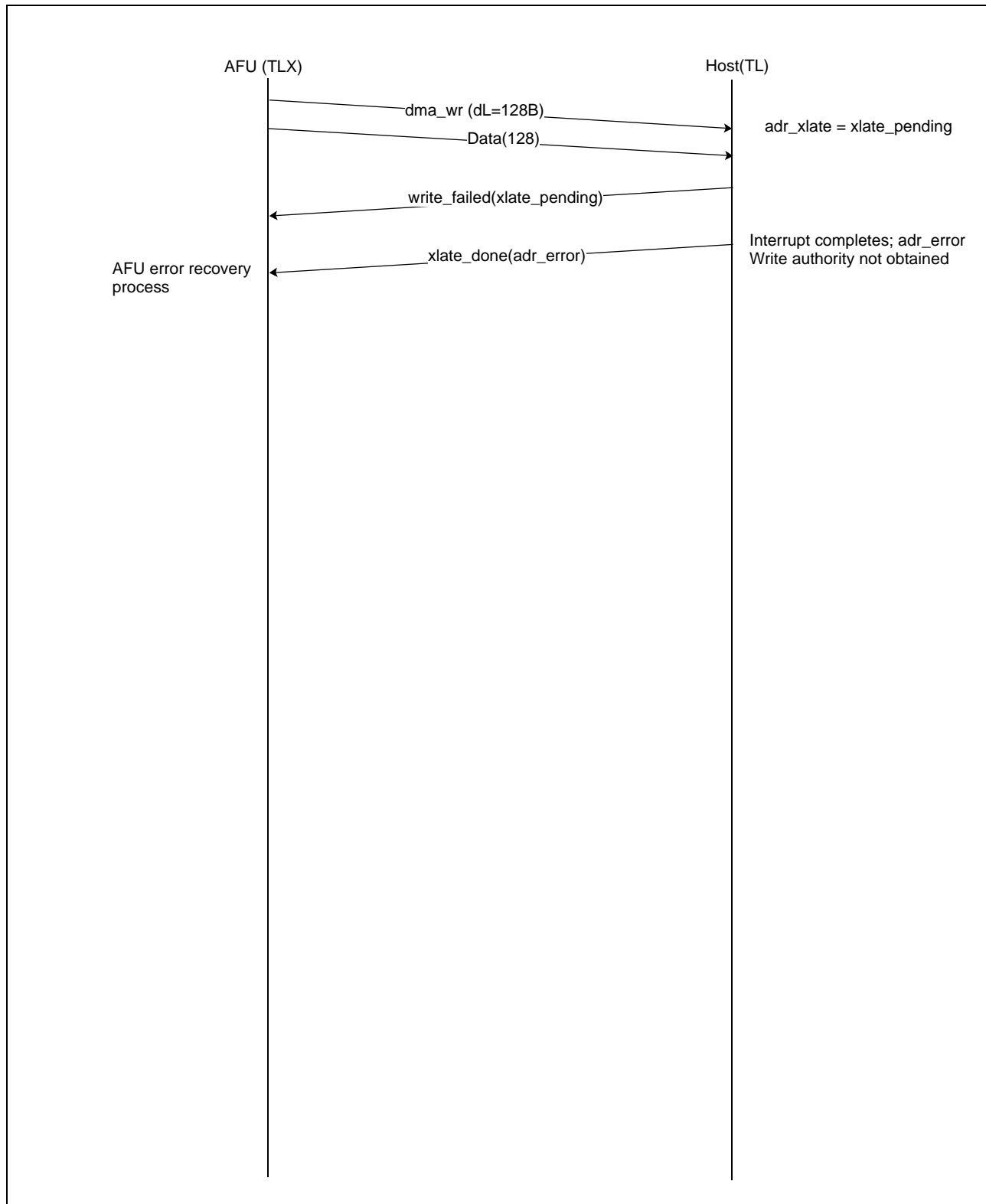
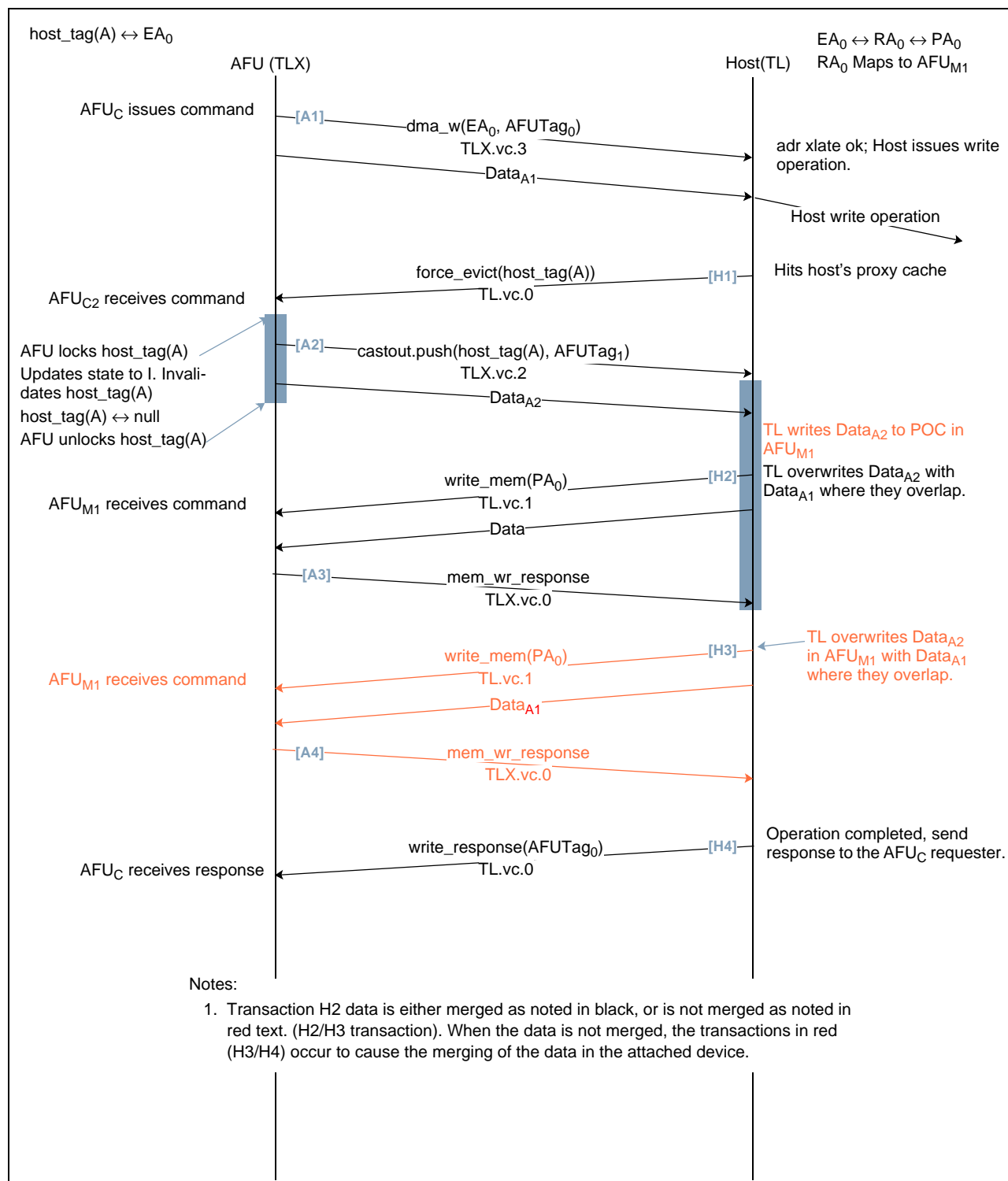
Figure A-3. TLX and TL interaction: ***dma_w*** (Page 1 of 2)

Figure A-3. TLX and TL interaction: ***dma_w*** (Page 2 of 2)

Approved

A.4 AFU DMA Write hits device co-located AFU_{C2} and AFU_{M1}

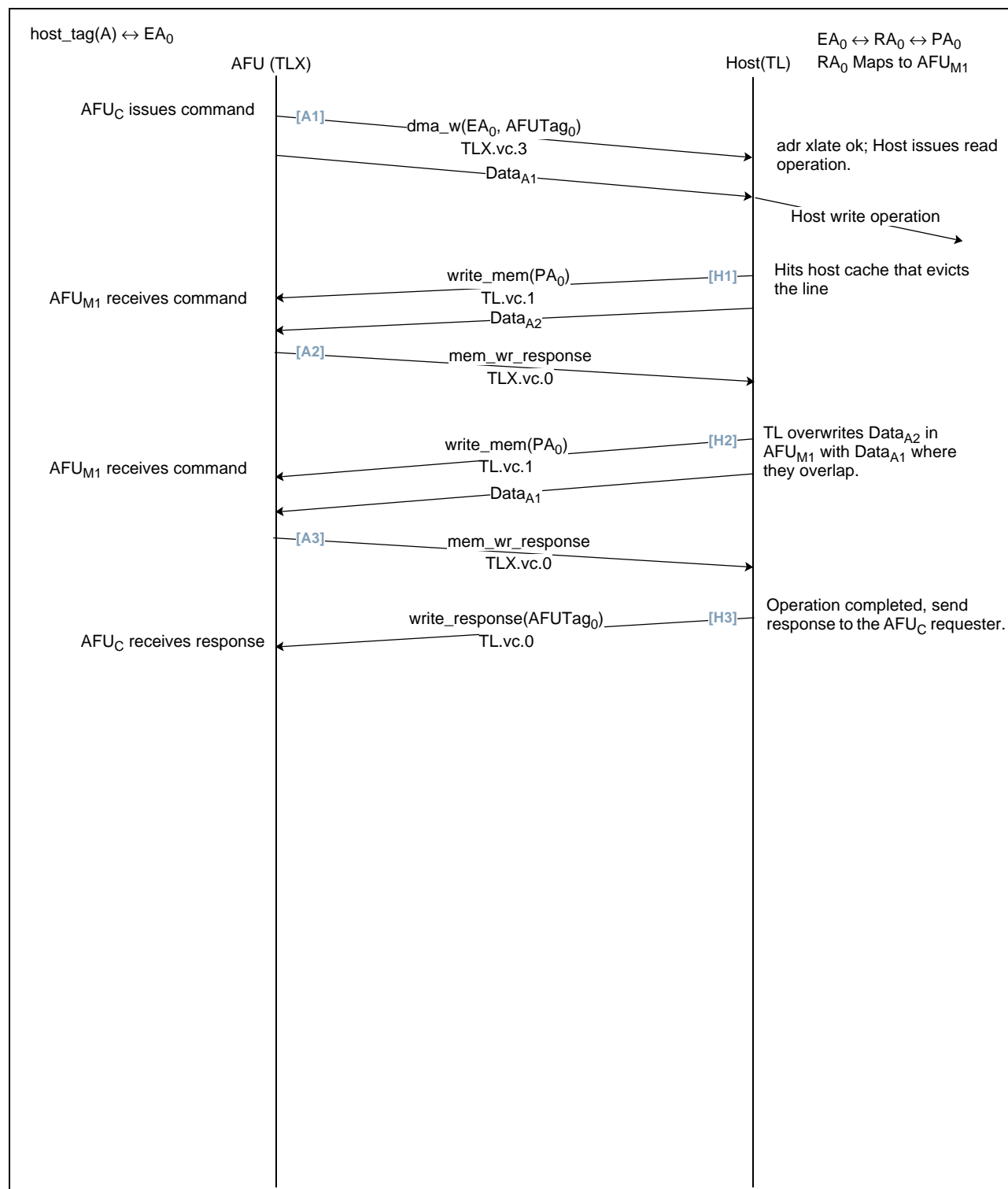
Figure A-4. TLX **dma_w** hits AFU_{C2} and AFU_{M1} TL issues commands to complete operation



Approved

A.5 AFU DMA Write hits device co-located AFU_{M1}

Figure A-5. TLX **dma_w** hits host cache and AFU_{M1} TL issues commands to complete operation

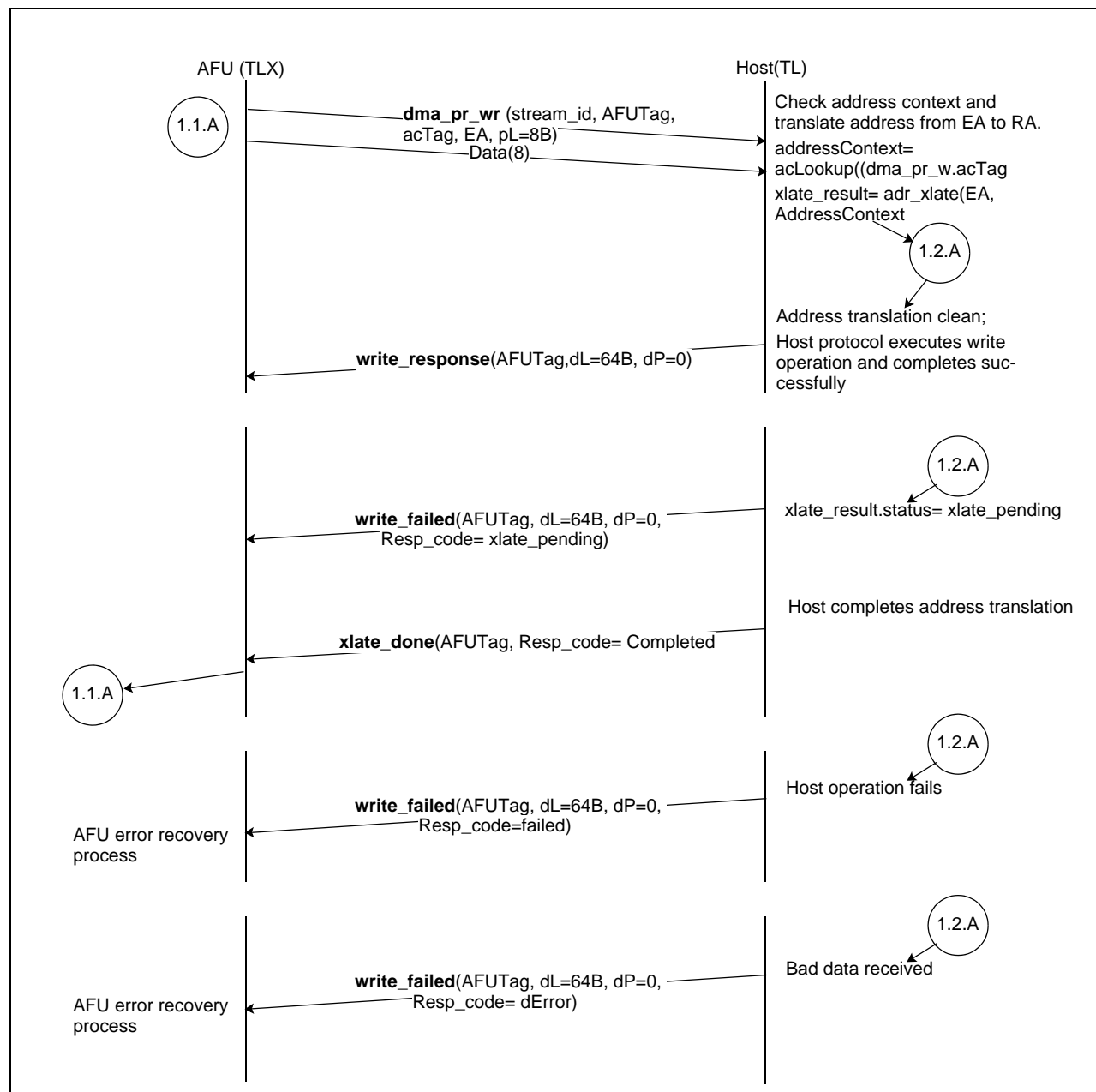


Approved

A.6 AFU DMA partial write; non-posted, 8 bytes

- For the dot-s form of the command, dispatch is stopped until all prior commands associated with the VC, BDF,PASID, and stream_id have completed on the host processor bus. See the description of **presync** on page 23

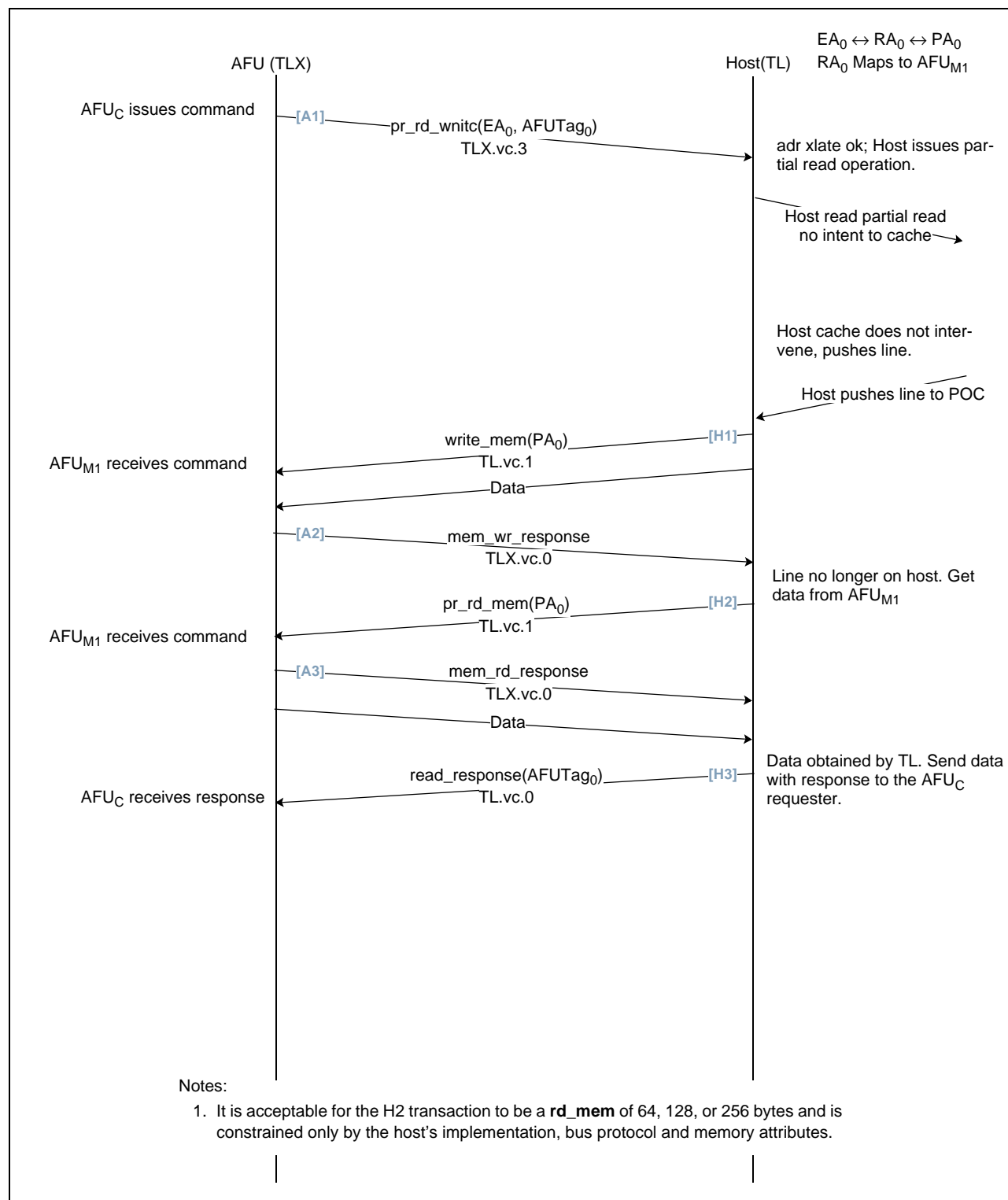
Figure A-6. TL and TLX interaction: **dma_pr_w**



Approved

A.7 AFU Partial read with no intent to cache hits device co-located AFU_{M1}

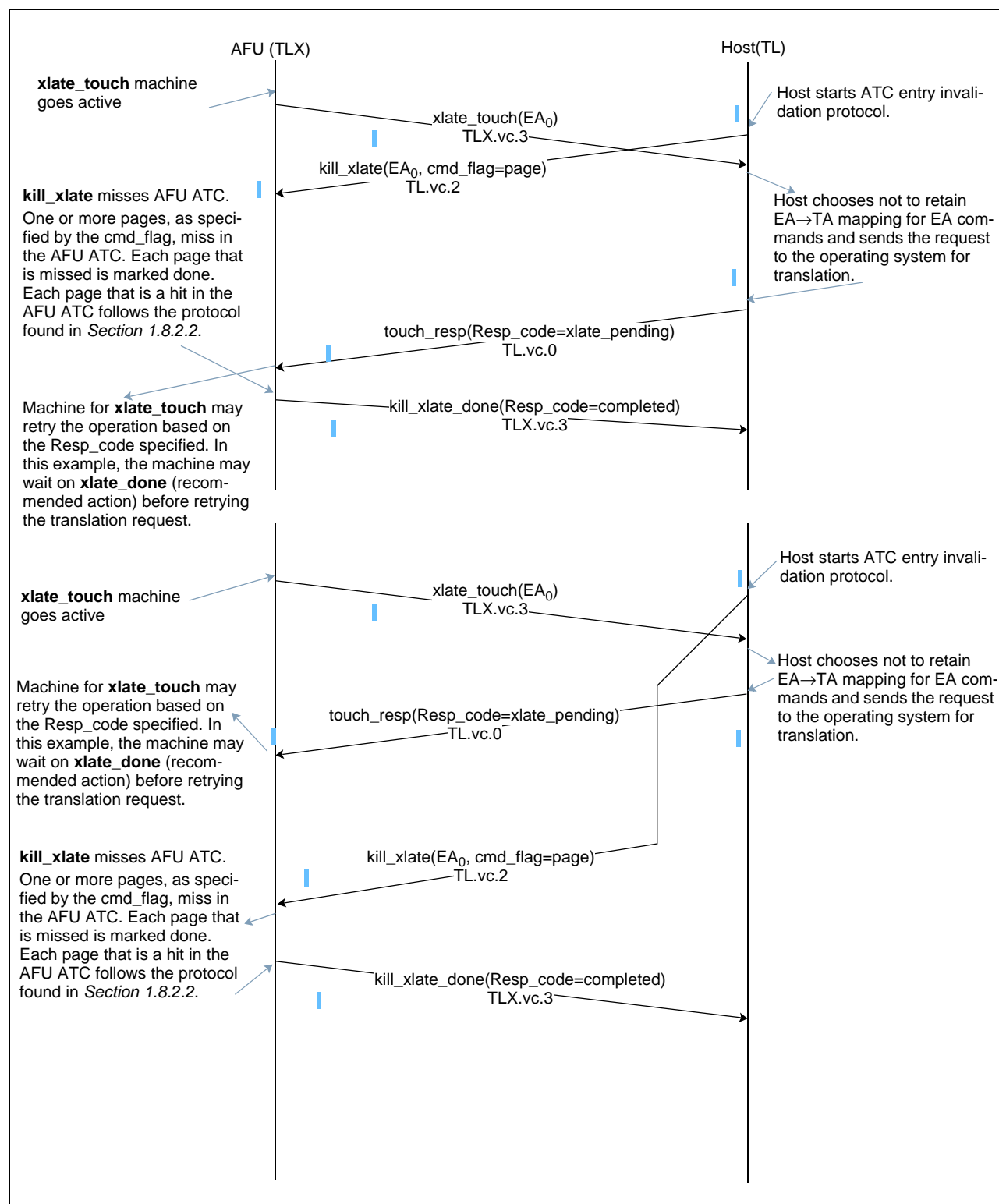
Figure A-7. TLX **pr_rd_wnrtc** hits AFU_{M1} TL issues commands to complete operation



Approved

A.8 Translate touch (xlata_touch, ta_req)

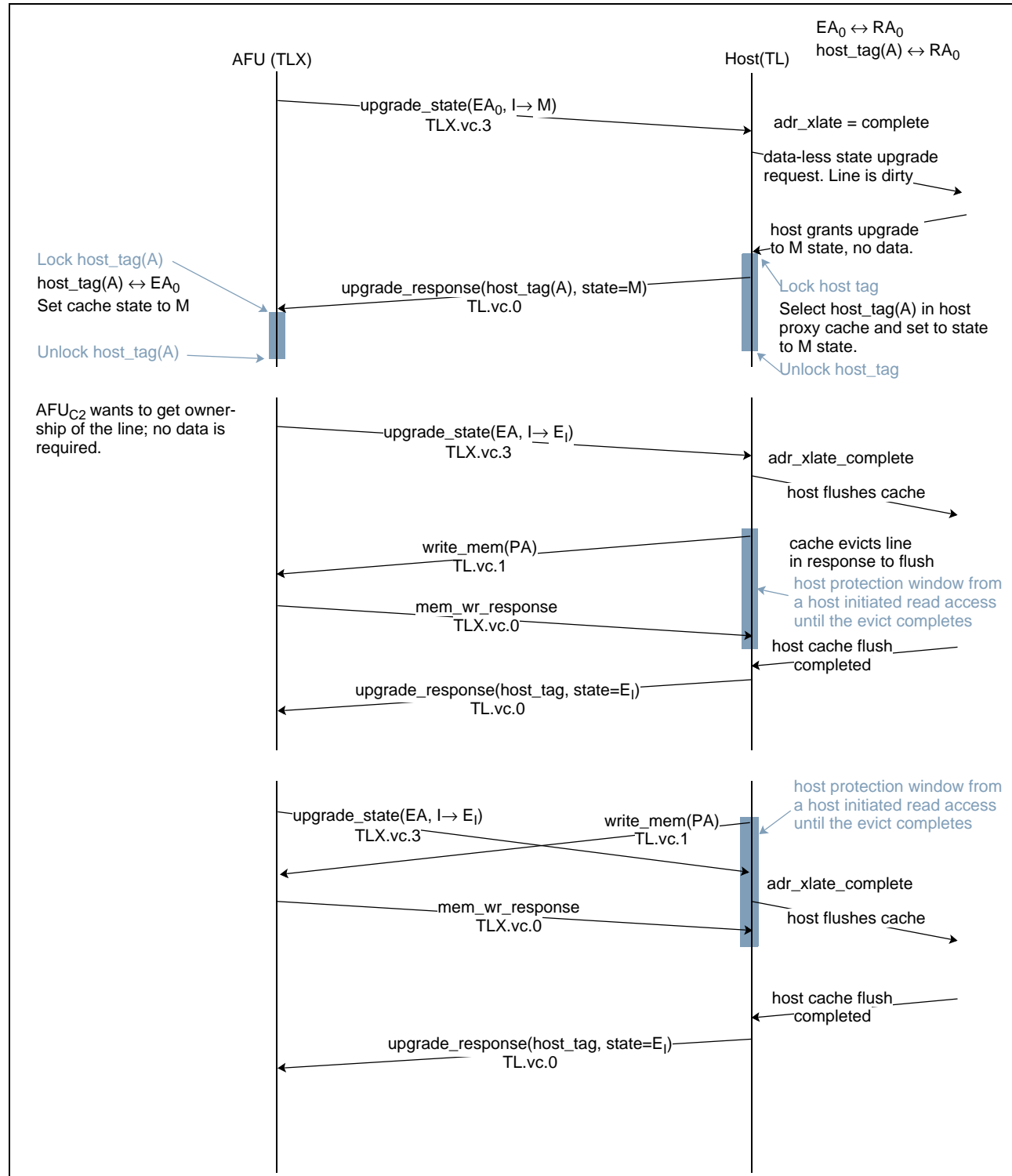
Figure A-8. **xlata_touch** TLX and TL interaction



Approved

A.9 Upgrade state

Figure A-9. *upgrade_state* TLX and TL interaction (Page 1 of 2)



Approved

Figure A-9. **upgrade_state** TLX and TL interaction (Page 2 of 2)

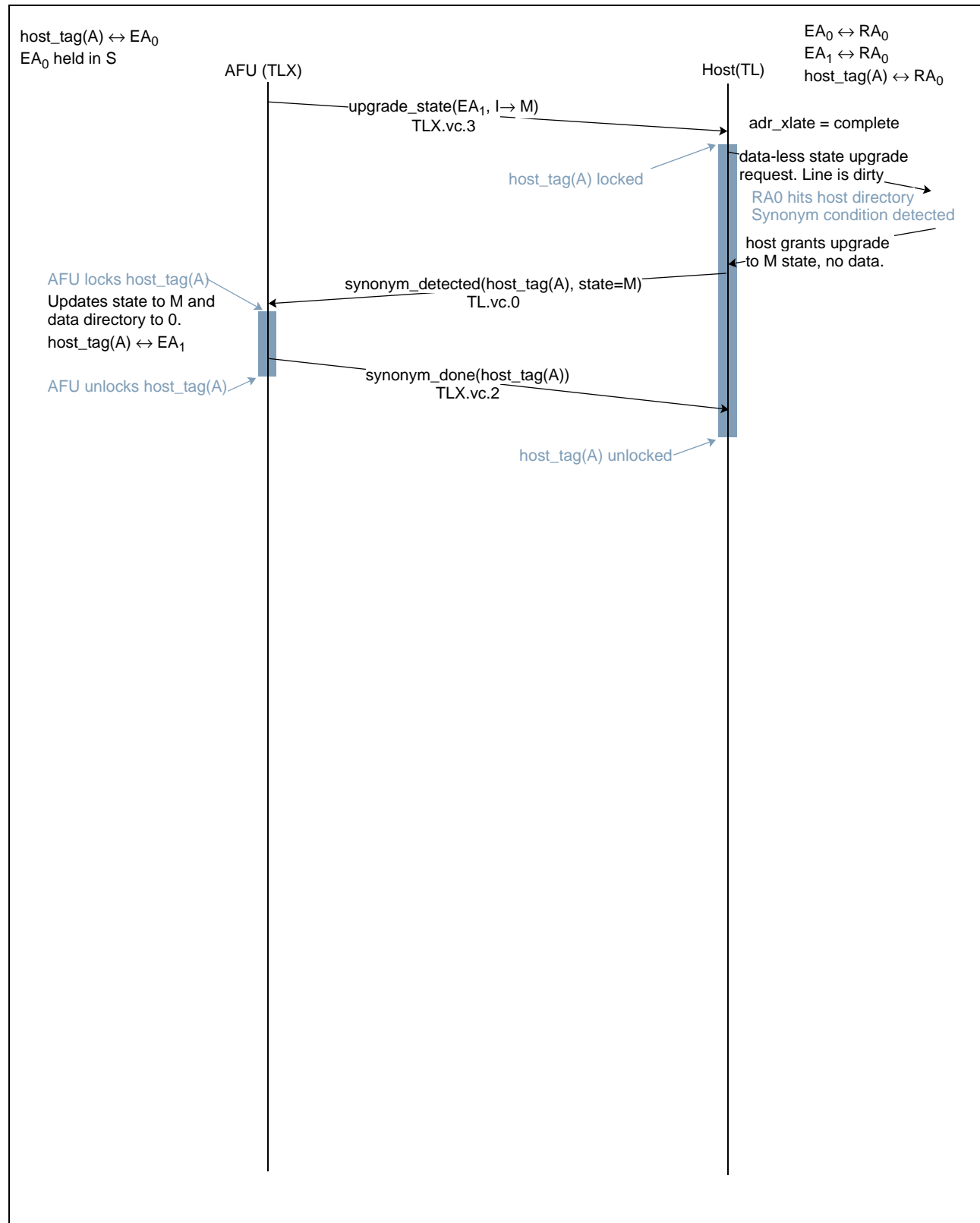


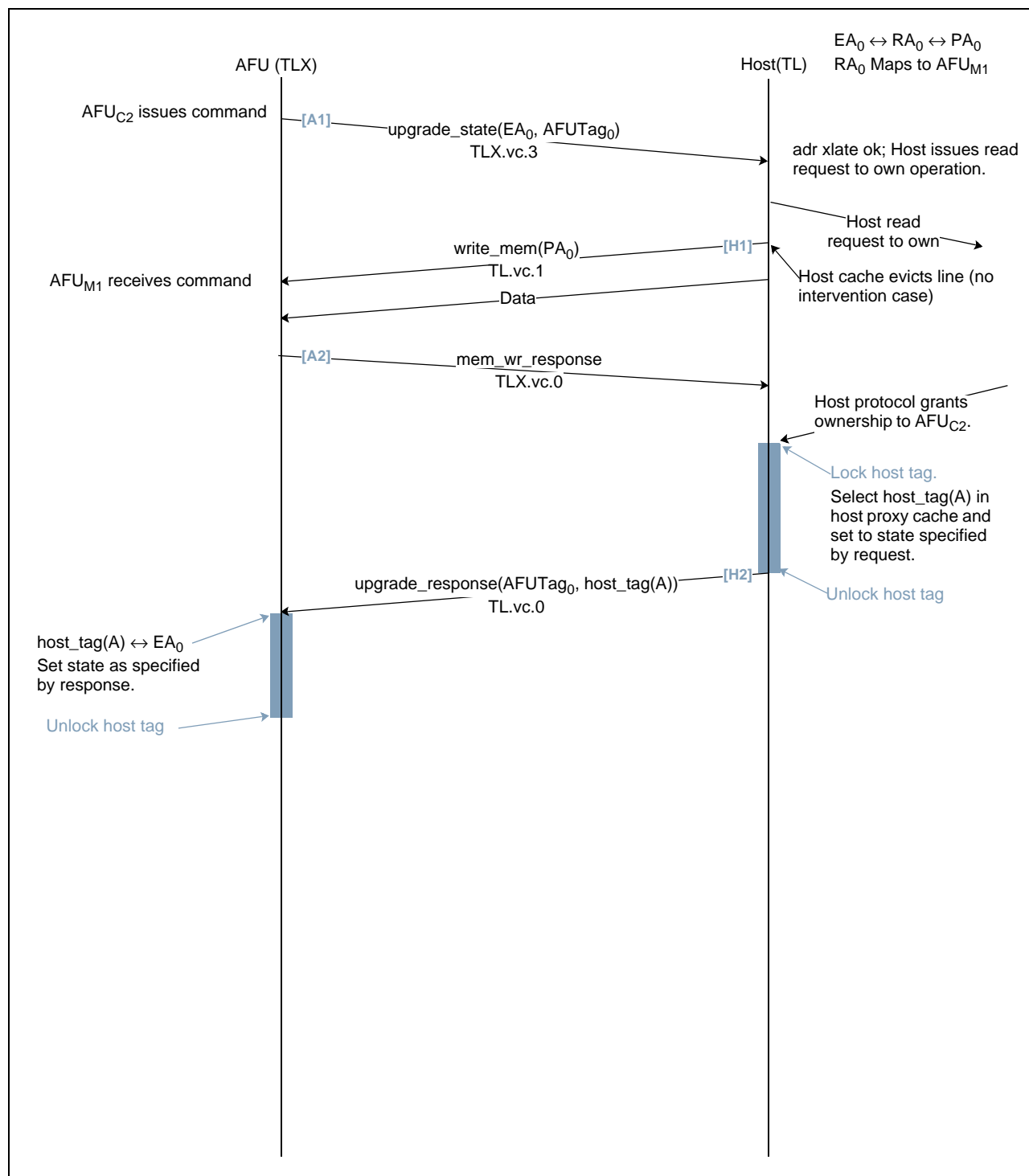
Figure A-10. TLX **upgrade_state** hits host cache and AFU_{M1} TL issues commands to complete operation

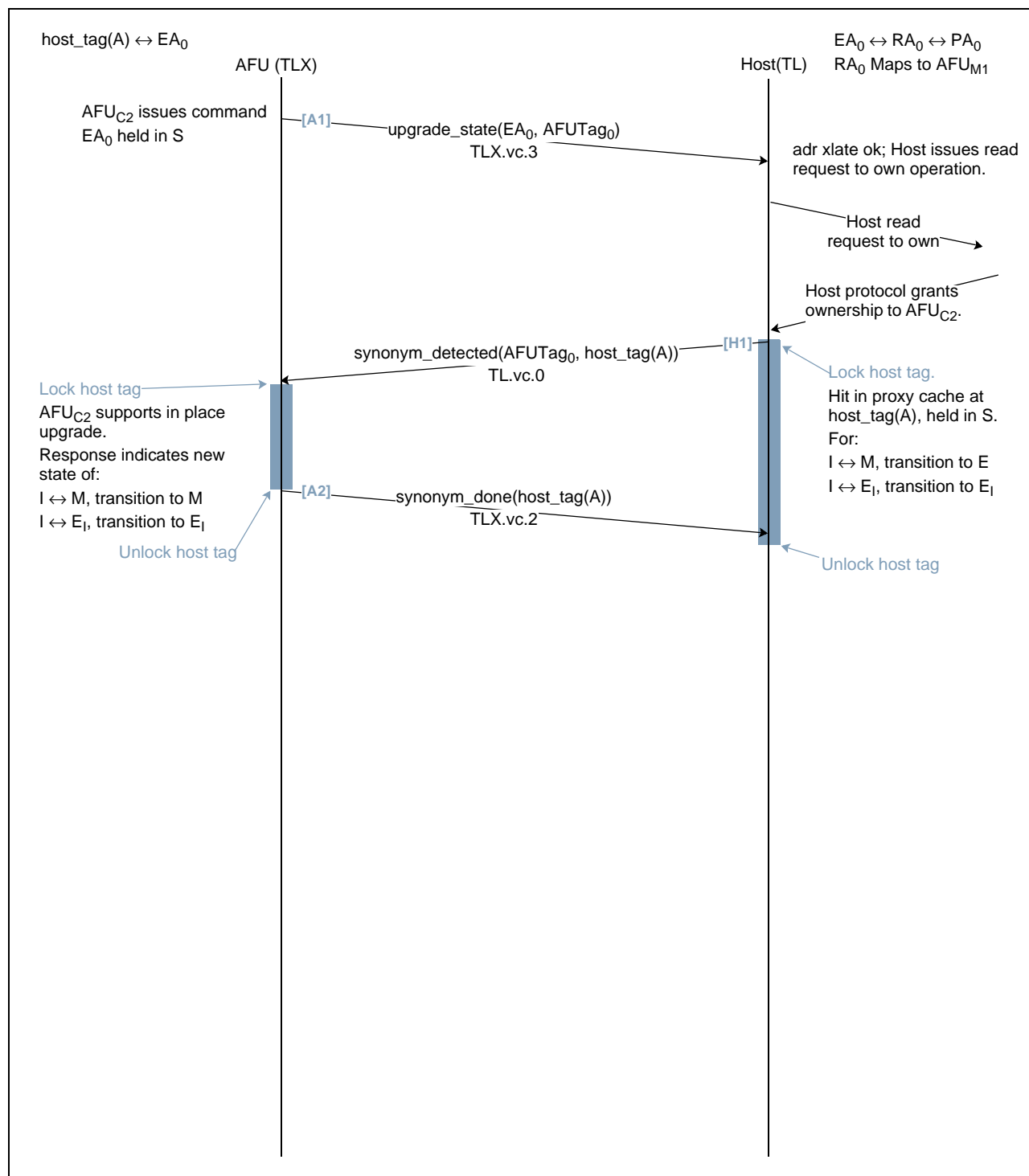
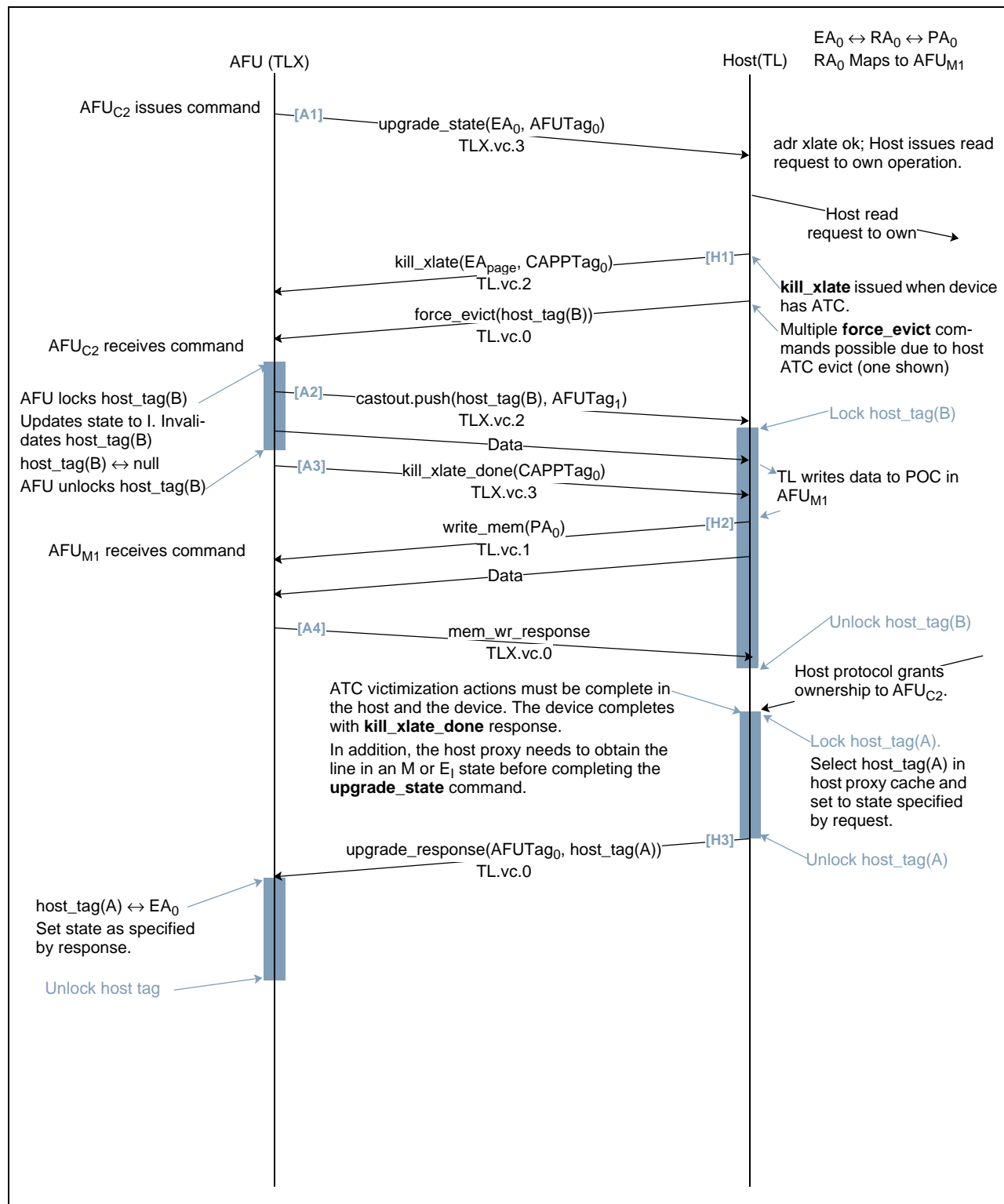
Figure A-11. TLX **upgrade_state** hits AFU_{C2} and AFU_{M1} TL issues commands to complete operation

Figure A-12. TLX **upgrade_state** hits AFU_{M1}, requires host ATC evict TL issues commands to complete operation



Approved

A.10 Host tag locking transactions

Figure A-13. *host_tag reuse* Initial use and first reuse case

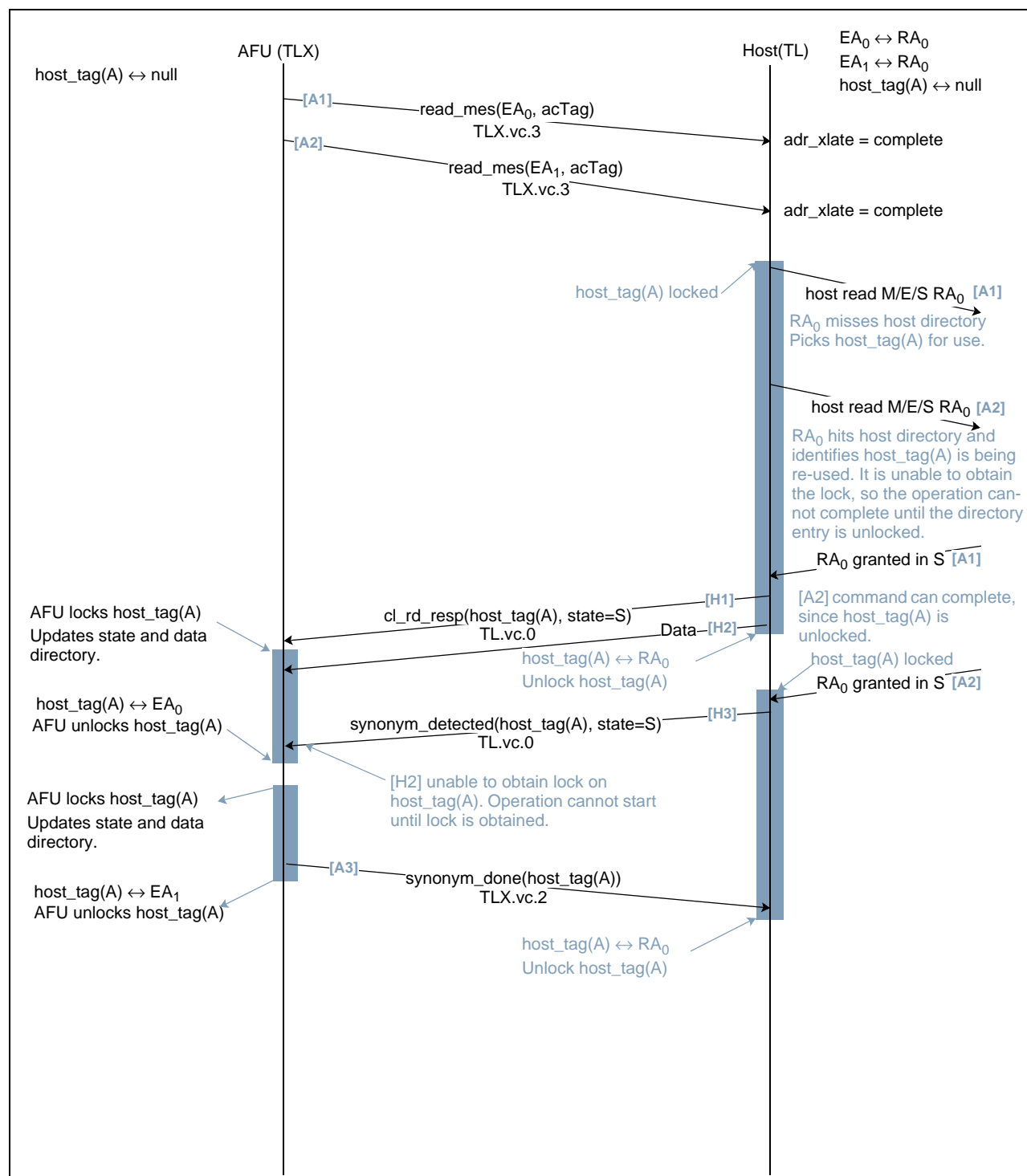


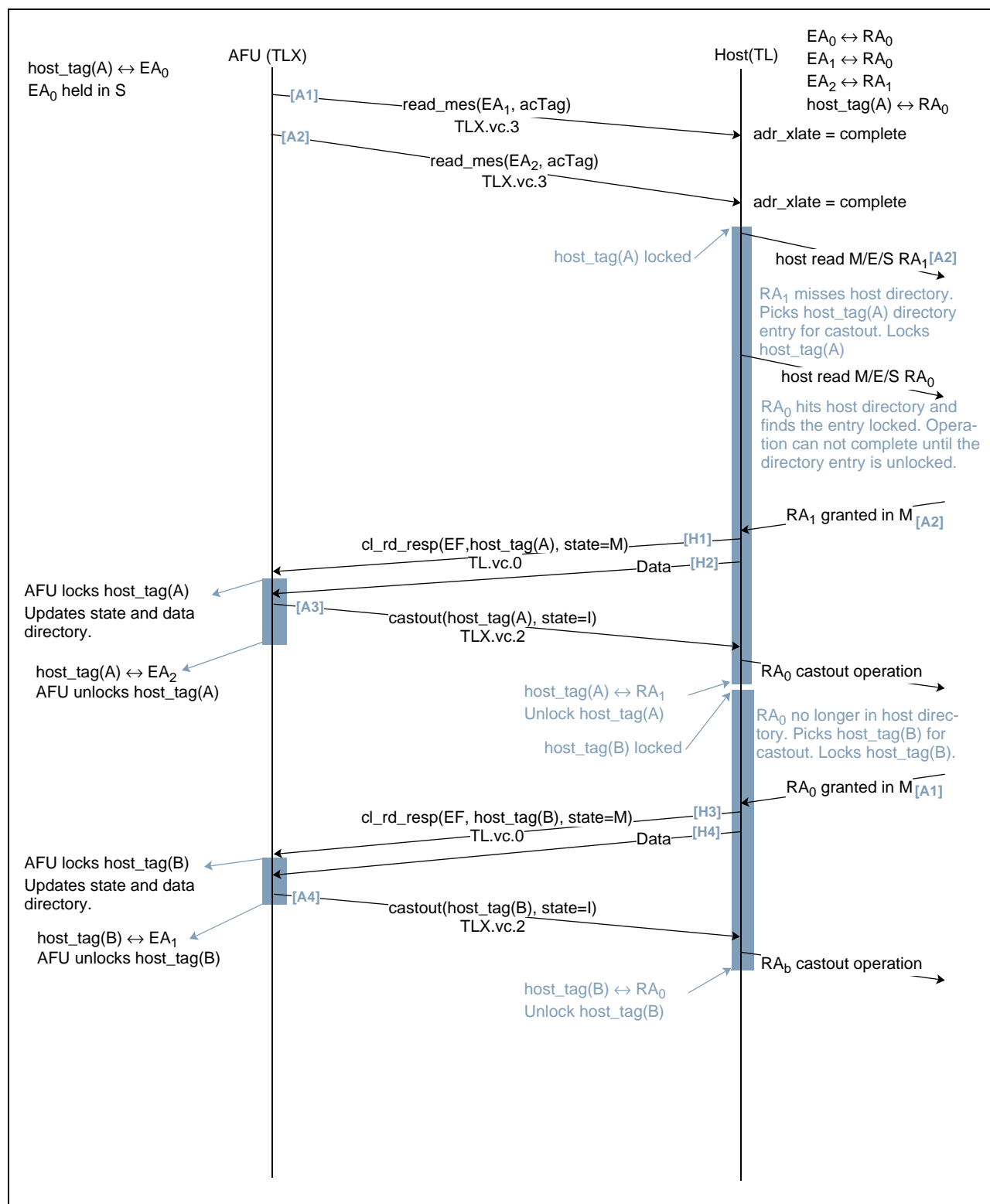
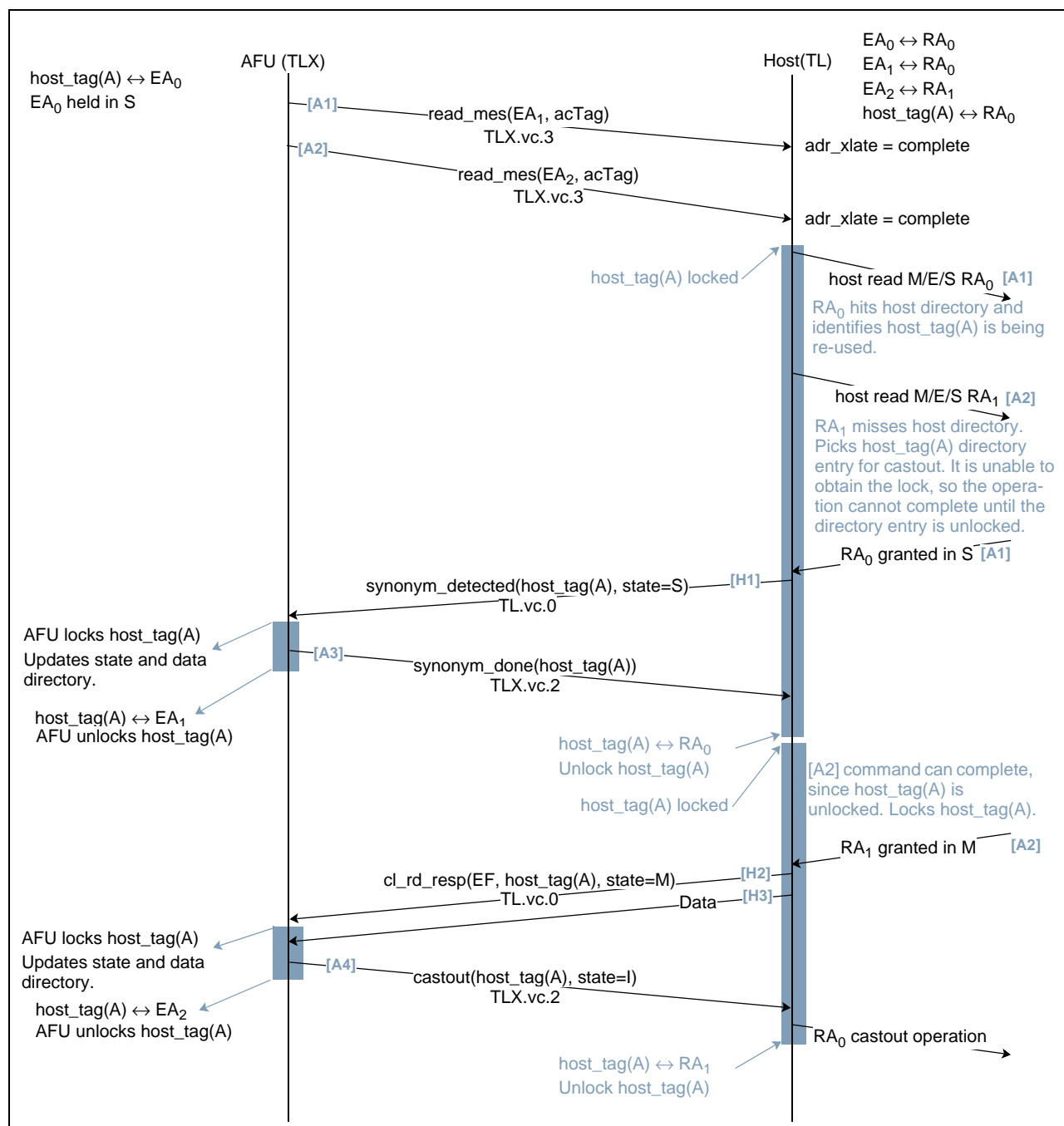
Figure A-14. *host tag reuse* Command inversion on the host. *host_tag* locking illustrated.

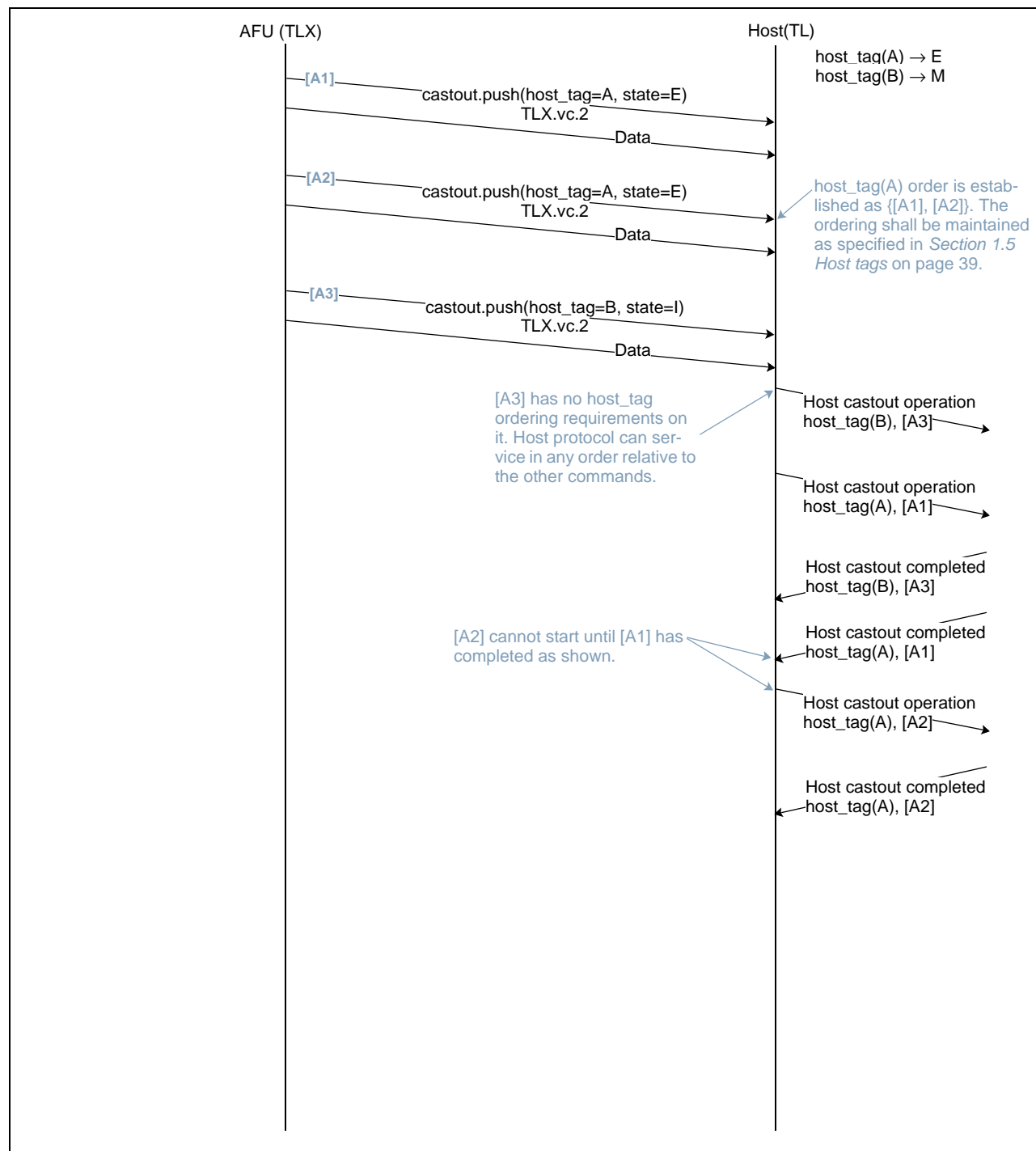
Figure A-15. *host_tag* reuse synonym update shown

[A1], [H1], [A3] synonym response group can repeat for any number of synonym cases and only one synonym will be serviced at a time because the host locks the entry from the time the host starts servicing [A1] through [A3].

Approved

A.11 Castout push

Figure A-16. **castout.push** example showing *host_tag* ordering at the host



Approved

Appendix B. CAPP (TL) command transaction diagrams

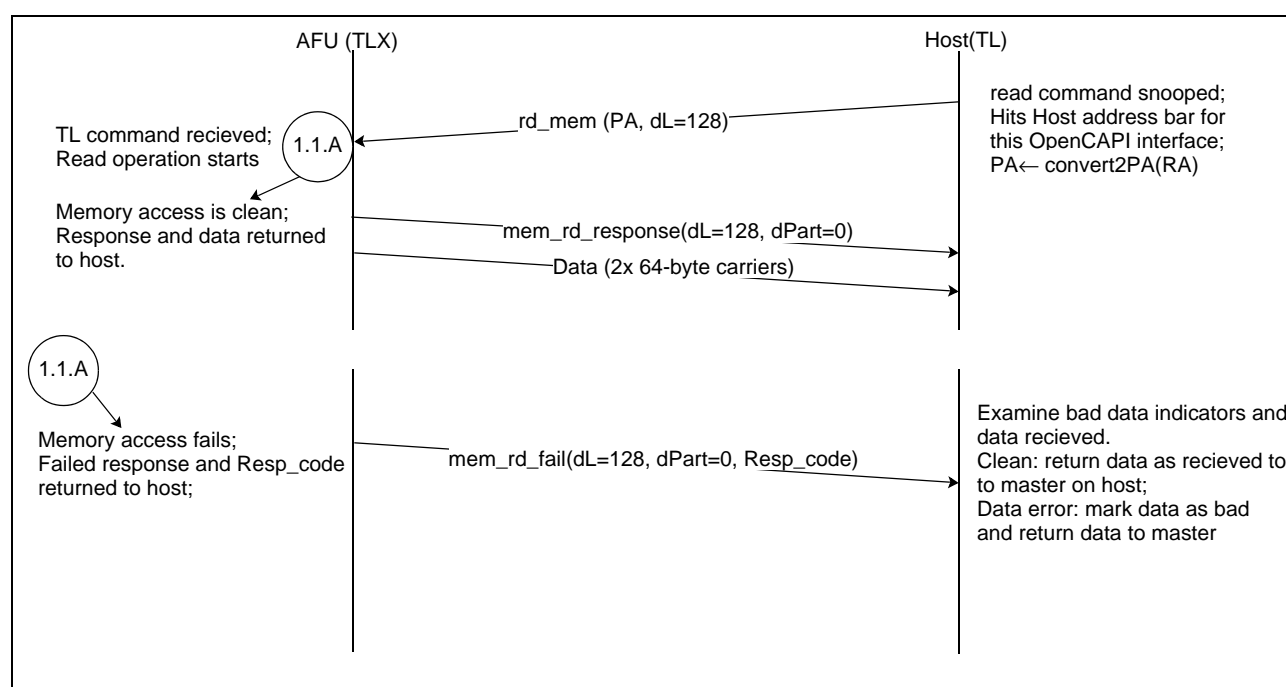
This section contains figures that illustrate CAPP command flows.

Rules:

1. Commands received at the TLX are not serviced until all data, if any, specified by the CAPP command has arrived.

B.1 CAPP memory read; 128 bytes

Figure B-1. TL and TLX transaction: *rd_mem*



Approved

B.2 CAPP memory write; 128 bytes

Figure B-2. TL and TLX transaction: *write_mem*

