AMBA® LTI

Protocol Specification

arm

ARM IHI 0089A (ID040620)

AMBA LTI
Protocol Specification

Release Information

The following changes have been made to this specification:

Change history

Date	Issue	Confidentiality	Change
7 April 2020	A	Non-Confidential	First release

ARM AMBA SPECIFICATION LICENCE

THIS END USER LICENCE AGREEMENT ("LICENCE") IS A LEGAL AGREEMENT BETWEEN YOU (EITHERASINGLE INDIVIDUAL, OR SINGLE LEGAL ENTITY) AND ARM LIMITED ("ARM") FOR THE USE OF THE RELEVANT AMBA SPECIFICATION ACCOMPANYING THIS LICENCE. ARM IS ONLY WILLING TO LICENSE THE RELEVANT AMBA SPECIFICATION TO YOU ON CONDITION THAT YOU ACCEPT ALL OF THE TERMS IN THIS LICENCE. BY CLICKING "I AGREE" OR OTHERWISE USING OR COPYING THE RELEVANT AMBA SPECIFICATION YOU INDICATE THAT YOU AGREE TO BE BOUND BY ALL THE TERMS OF THIS LICENCE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENCE, ARM IS UNWILLING TO LICENSE THE RELEVANT AMBA SPECIFICATION TO YOU AND YOU MAY NOT USE OR COPY THE RELEVANT AMBA SPECIFICATION AND YOU SHOULD PROMPTLY RETURN THE RELEVANT AMBA SPECIFICATION TO ARM. "LICENSEE" means You and your Subsidiaries.

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

1. Subject to the provisions of Clauses 2, 3 and 4, Arm hereby grants to LICENSEE a perpetual, non-exclusive, non-transferable, royalty free, worldwide licence to:
(i) use and copy the relevant AMBA Specification for the purpose of developing and having developed products that comply with the relevant AMBA Specification;
(ii) manufacture and have manufactured products which either: (a) have been created by or for LICENSEE under the licence granted in Clause 1(i); or (b) incorporate a product(s) which has been created by a third party(s) under a licence granted by Arm in Clause 1(i) of such third party's Arm AMBA Specification Licence; and
(iii) offer to sell, sell, supply or otherwise distribute products which have either been (a) created by or for LICENSEE under the licence granted in Clause 1(i); or (b) manufactured by or for LICENSEE under the licence granted in Clause 1(ii).

2. LICENSEE hereby agrees that the licence granted in Clause 1 is subject to the following restrictions:
(i) where a product created under Clause 1(i) is an integrated circuit which includes a CPU then either: (a) such CPU shall only be manufactured under licence from Arm; or (b) such CPU is neither substantially compliant with nor marketed as being compliant with the Arm instruction sets licensed by Arm from time to time;
(ii) the licences granted in Clause 1(iii) shall not extend to any portion or function of a product that is not itself compliant with part of the relevant AMBA Specification; and
(iii) no right is granted to LICENSEE to sublicense the rights granted to LICENSEE under this Agreement.

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

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

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

6. No licence, express, implied or otherwise, is granted to LICENSEE, under the provisions of Clause 1, to use the Arm tradename, or AMBA trademark in connection with the relevant AMBA Specification or any products based thereon. Nothing in Clause 1 shall be construed as authority for LICENSEE to make any representations on behalf of Arm in respect of the relevant AMBA Specification.

7. This Licence shall remain in force until terminated by you or by Arm. Without prejudice to any of its other rights if LICENSEE is in breach of any of the terms and conditions of this Licence then Arm may terminate this Licence immediately upon giving written notice to You. You may terminate this Licence at any time. Upon expiry or termination of this Licence by You or by Arm LICENSEE shall stop using the relevant AMBA Specification and destroy all copies of the relevant AMBA Specification in your possession together with all documentation and related materials. Upon expiry or termination of this Licence, the provisions of clauses 6 and 7 shall survive.

8. The validity, construction and performance of this Agreement shall be governed by English Law.

Confidentiality Status

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

Product Status

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

Web Address

`http://www.arm.com`

Contents
AMBA LTI Protocol Specification

Preface

This preface introduces the *AMBA LTI Protocol Specification*. It contains the following sections:

About this specification

Intended audience

This specification is written for hardware engineers who wish to design components that implement LTI.

Using this specification

Chapter 1 *Introduction*

Introduction to the AMBA Local Translation Interface (LTI) protocol specification

Chapter 2 *Channels*

Describes LTI information flow

Chapter 3 *Properties*

Describes the set of LTI properties that specify the supported behavior and interface signaling requirements

Chapter 4 *Request channel*

Defines the LTI request (LA) channel

Chapter 5 *Response channel*

Defines the LTI response (LR) channel

Chapter 6 *Completion channel*

Defines the LTI completion (LC) channel

Chapter 7 *Interface management*

Describes the LTI interface management function

Chapter 8 *Clock and reset*

Describes a mapping strategy for clock and reset

Chapter 9 *Pipelining*

Defines pipelining requirements for LTI

Appendix A *Mapping LTI to ACE-Lite*

Describes how to map LTI concepts onto ACE-Lite

Appendix B *LTI Mapping to DTI*

Describes how to map LTI concepts onto DTI-TBU

Conventions

The following sections describe conventions that this specification can use:

Typographic conventions

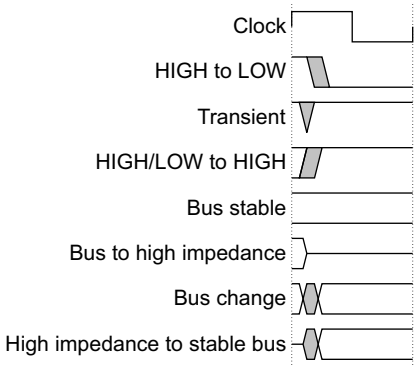
The typographical conventions are:

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

Timing diagrams

The figure that is named *Key to timing diagram conventions* explains the components that are used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

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



Key to timing diagram conventions

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

Signals

The signal conventions are:

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

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

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

Lowercase x At the second letter of a signal name denotes a collective term for both Read and Write. For example, **AxCACHE** refers to both the **ARCACHE** and **AWCACHE** signals.

Numbers

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

Additional reading

This section lists relevant publications from Arm.

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

Arm publications

- *AMBA AXI and ACE Protocol Specification* (ARM IHI 0022)
- *AMBA DTI Protocol Specification* (ARM IHI 0088)
- *Arm System Memory Management Unit Architecture Specification* (ARM IHI 0070)

Feedback

Arm welcomes feedback on its documentation.

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

- The title, AMBA LTI Protocol Specification
- The number, ARM IHI 0089A
- The page number(s) that your comments apply
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter introduces the AMBA Local Translation Interface (LTI) protocol specification:

1.1 About the LTI protocol

The AMBA LTI protocol specification aligns with the Arm System MMU architecture and complements AMBA DTI to provide higher performance and more efficient translation services.

LTI is a point-to-point protocol and defines the communication between an IO master and a TBU. It enables devices to directly request a translation for each transaction while leaving the TBU to manage the TLB, enabling translations to be requested before ordering requirements are met and avoiding the need to pass transactions through the TBU. This enables improved performance and reduced silicon area.

This specification describes the LTI protocol and the components of an LTI-compliant implementation. The LTI protocol is used by implementations of the Arm System MMUv3 (SMMUv3) Architecture Specification.

1.2 Use cases

A system incorporating an LTI-based System Memory Management Unit (SMMU) might have the following components:

- A client device whose transactions require translation.
- A Bus Interface Unit (BIU), which fetches a translation for each transaction using LTI.
- A TLB Unit (TLBU), which caches translations in a Translation Lookaside Buffer (TLB). It receives translation requests using LTI, and if the requested translation is not available in its TLB, it requests translations from a TCU using DTI.
- A Translation Control Unit (TCU), which calculates translations, reading translation tables when required.

Typically the BIU and TLBU are packaged together as a Translation Buffer Unit (TBU). Alternatively, the TBU might consist of just the TLBU.

A device can be integrated in one of the following ways:

- *In-line integration*
- *Look-aside integration* on page 1-20
- *Cached integration* on page 1-20

1.2.1 In-line integration

When in-line integration is implemented, every transaction passes through the SMMU, typically using AXI or ACE-Lite. The device is not aware of translation. This is the simplest integration option, which is shown in Figure 1-1.

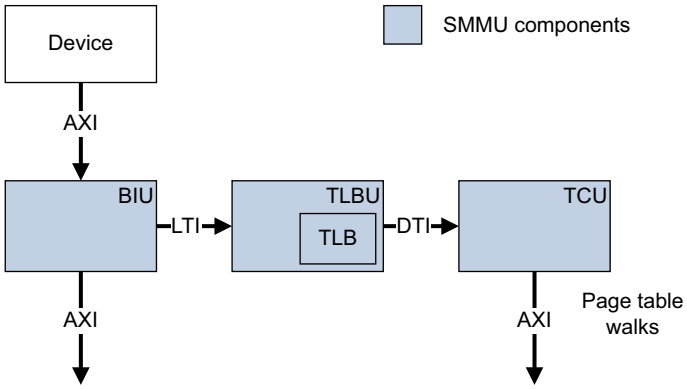


Figure 1-1 In-line SMMU Integration

1.2.2 Look-aside integration

When look-aside integration is implemented, a translation request is made over LTI for each transaction, but the transaction data does not pass through the SMMU. Transaction completion is signaled to the TLBU using the LTI interface. This option is more complex than in-line integration, enables the device to connect to the system using its native interface. Figure 1-2 shows the look-aside integration option.

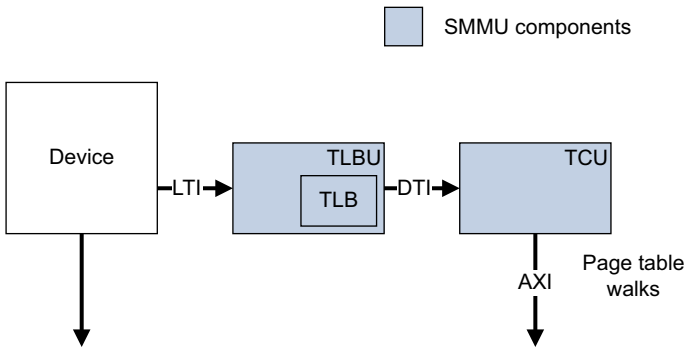


Figure 1-2 Look-aside SMMU integration

1.2.3 Cached integration

When cached integration is implemented, the device requests translations over DTI when required and caches them locally in an integrated TLB. The device must support invalidation messages. This option is the most complex, but gives the device control over how translations are cached, and can be the most efficient option for devices with specific caching requirements. Figure 1-3 shows the cached integration option.

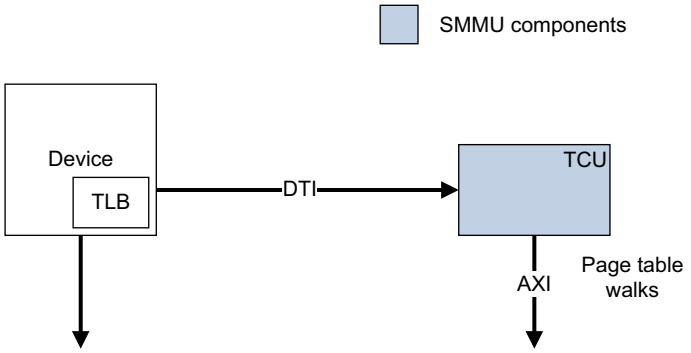


Figure 1-3 Cached SMMU integration

1.3 Differences between DTI and LTI

The DTI and LTI protocols perform similar functions, but are different in several ways:

Table 1-1 LTI and DTI comparison

	LTI	DTI
Topology	Designed for connecting to a single, local TLBU over a short distance.	Designed for connecting multiple TBUs to a TCU over a longer distance.
Caching	Translations must be used immediately and must not be not cached.	Translations can be cached.
Invalidation	No invalidation is required, because translations are not cached.	The DTI master must support invalidation messages to invalidate translations that are previously returned.
Transport	AXI-like channel-based protocol gives dedicated signals for each data field to maximize throughput and avoid protocol conversion latency.	Message-based protocol can easily be passed over generic transports.
Mapping translations to transactions	Provides the translated transaction information directly, making it simple to use.	Requires the master to follow rules on how to map configuration cache entries and TLB entries on to each transaction.
Ordering	Supports ordering of translations to match transaction ordering, or out of order operation.	Fully out-of-order.

1.4 Supported translation flows

Several translation flows are supported by LTI.

1.4.1 Stall flow

When the Stall flow is used, software can configure the SMMU to take one of the following actions when a translation fault occurs:

- Immediately terminate the transaction and optionally record an error record that informs software that the transaction was terminated.
- Stall the translation and inform software that the translation is stalled. Software can then terminate the transaction, or update the translation tables and retry the translation. The LTI master is not aware of the stall.

Stall flow enables software to manage translation faults and demand paging without the client device being aware. However, it has some limitations:

- The LTI master can see long translation times, potentially triggering timeouts.
- Due to the dependence of software activity, the Stall flow can cause deadlocks in some systems.

For example, Stall flow is not recommended for use with PCIe, because of dependencies between outgoing transactions to PCIe from a CPU, and incoming transactions from PCIe through the SMMU.

Stall flow is the most common flow for LTI masters.

Enabling the Stall flow does not necessarily cause a stall when a translation fault occurs. Stalls only occur when enabled by software. Software does not normally enable stalling for PCIe endpoints.

1.4.2 ATST flow

The Address Translation Service Translated (ATST) flow is a flow that is used by PCIe Root Ports only. The flow is:

1. A PCIe endpoint requests a translation over ATS, which is passed to the SMMU by the PCIe Root Port using the DTI-ATS protocol.
2. The SMMU responds to the ATS request over DTI-ATS, which is passed back to the PCIe endpoint by the PCIe Root Port. If a translation fault occurs, then the response indicates the condition, and does not make any software-visible record.
3. The PCIe endpoint uses the ATS translation to translate the address of transaction, and then issues a transaction that is marked as ATS Translated.

An ATS Translated transaction uses the ATST flow. In all other respects, it is translated the same way as any other transaction. Normally this translation is fast because it is already translated by ATS, but some additional translation might still occur. For example, the SMMU can be configured to perform stage 1 translation when an ATS request is made, and perform stage 2 translation when the ATS Translated transaction is presented to the SMMU.

ATS translations are cached in the PCIe endpoint Address Translation Cache (ATC), for future use. These translations can be invalidated by ATS invalidation messages, which conveyed over DTI-ATS.

If a translation fault occurs, the PCIe endpoint can issue a page request using the Page Request Interface (PRI), before retrying the ATS translation request if the PRI request is successful. PRI is an extension to ATS, and is also conveyed to the SMMU using DTI-ATS. The fault is not visible to LTI.

1.4.3 NoStall flow

NoStall flow is used by a master that is not able to be stalled. If a translation fault occurs, the transaction is terminated, even if software has configured the device to be stalled when a translation fault occurs.

1.4.4 PRI flow

The PRI flow is designed for use with a PCIe-enumerated endpoint. The device uses the LTI PRI flow to enable software to respond to translation faults without risking deadlock.

From a software point of view:

- The device is a PCIe endpoint.
- The device uses the ATS protocol to fetch translations.
- When a translation fault occurs, the device makes a page request using PRI.

The behavior of the LTI PRI flow is as follows:

- The SMMU ATS features are not required and not used, even though ATS is enabled in software. Transactions are translated on-demand.
- If a translation fault occurs, no error is reported to software by the SMMU. Instead, a PRI fault response is returned to the LTI master.
- When the LTI master receives a PRI fault response, it uses a DTI-ATS connection to issue a page request. If the PRI response is successful, then the LTI master retries the transaction.

A device using this flow uses DTI-ATS for PRI requests only and does not make any ATS requests. In DTIv2, a device can implement a DTI-ATS connection that just performs PRI requests and does not receive ATS invalidation messages.

A device must be assigned page request credits by system software before it can issue PRI requests. These credits are not visible to the LTI or DTI protocols and are managed by PCIe software.

Chapter 2
Channels

This chapter describes LTI information flow:

2.1 Transaction flow

Table 2-1 shows the three LTI channels. For each channel, the TX is the component that sends a message on the channel and the RX is the component that receives it.

Table 2-1 LTI interface channels

Channel name	Channel prefix	Channel TX	Channel RX
LTI request	LA	Master	Slave
LTI response	LR	Slave	Master
LTI completion	LC	Master	Slave

Additionally, Interface management signals are included with the prefix LM.

A complete LTI transaction consists of a message on all three channels in sequence:

1. The master sends a request on the LA channel.
2. The slave sends a response on the LR channel.
3. The master sends a completion on the LC channel.

There are no flows where any of these three stages can be skipped.

The LTI slave does not correlate the completion messages to a specific LTI transaction, but instead counts the number of completions corresponding to each completion tag. The purpose of the Completion channel is to enable the LTI slave to determine when all transactions with a completion tag have completed. This checking of the number of completions for a tag is part of the translation invalidation process. There are two completion tags to enable invalidation to take place without stopping the transaction flow.

An LTI response is permitted on the same cycle that the corresponding request is made. This response timing enables tightly coupled TLBs with low latency.

An LTI completion is not permitted on the same cycle that the corresponding response is returned. An LTI response can be followed by its completion on the following cycle or later.

All signals are driven by the channel TX, except **LxCREDIT** signals, which are driven by the channel RX.

2.2 Virtual channels

The LA and LR channels can be composed of multiple Virtual Channels (VCs). The LA and LR channels must support the same number of VCs.

The LC channel does not support multiple VCs. The LTI slave must always be able to provide a credit on the LC channel without dependence on progress of other channels.

An LTI transaction uses the same VC for its LA channel and LR messages.

The intent of virtual channels is to enable one VC to progress when the other is blocked, to avoid deadlock scenarios. Components implementing LTI must ensure that if progress is blocked on one VC it does not result in progress being blocked on a different VC.

Because LC messages are not associated with a particular VC, it is necessary that any counter in the LTI slave that is counting outstanding completions does not overflow. The LTI slave must be able to track at least 4095 translation responses awaiting a completion. When this number is exceeded, the LTI slave is permitted to introduce dependencies between virtual channels, by not returning a translation response until translation completions from previous translations are returned.

2.3 Flow control

Flow control in the LTI protocol has the following rules:

- Each channel includes **LxVALID** and **LxCREDIT** signals. If more than one VC is supported, the LA and LR channels include **LxVC**.
- Each cycle that **LxCREDIT[***n***]** is asserted grants one credit on VC *n* to the channel TX.
- The channel TX consumes a credit on VC *n* each cycle that **LxVALID** is asserted with **LxVC**=*n*.
- **LxVALID** cannot be asserted with **LxVC**=*n* when the channel TX has zero credits for VC *n*.
- The maximum number of credits that the channel RX can grant the channel TX for each VC is 15.

In addition, there must not be combinatorial paths between **LxCREDIT** and other signals on a channel in either direction, such as **LxVALID**. This restriction has the following consequences:

- A credit cannot be used by **LxVALID** in the same cycle that it is granted by **LxCREDIT**, when there are no other credits granted.
- A credit cannot be returned on **LxCREDIT** in the same cycle that it is used by **LxVALID**, when the maximum number of credits that are permitted to be granted is reached.

2.4 Reserved encodings

Where encodings are given, unused encodings are Reserved.

Use of a Reserved encoding in a field that is not ignored is a protocol error and can lead to UNPREDICTABLE behavior.

2.5 Signal validity

Sometimes this specification defines a signal as not valid. When a signal is not valid:

- The channel TX can drive it to any value.
- The channel RX must ignore its value.

Chapter 3
Properties

This chapter describes the set of LTI properties that specifies the supported behavior and interface signaling requirements:

3.1 LTI properties

All LTI properties have a minimum value of 0 and have no defined maximum value, unless otherwise specified.

Table 3-1 LTI Properties

Name	Type	Unit	Description
LTI_VC_COUNT	Integer	-	Number of Virtual Channels. Minimum 1.
LTI_ID_WIDTH	Integer	bits	Width of translation request ID.
LTI_SID_WIDTH	Integer	bits	Width of StreamID, maximum 32. Higher-order bits of StreamID might be statically defined by the LTI slave, for example using a tie-off.
LTI_SSID_WIDTH	Integer	bits	Width of SubstreamID, maximum 20.
LTI_OG_WIDTH	Integer	bits	Width of order group.
LTI_TLBLOC_WIDTH	Integer	bits	Width of TLB location.
LTI_LOOP_WIDTH	Integer	bits	Width of loopback signals.
LTI_LRADDR_WIDTH	Integer	-	Width of translated address. LTI_LRADDR_WIDTH must be one of the following values: • 32 • 36 • 40 • 42 • 44 • 48 • 52
LTI_LAUSER_WIDTH LTI_LRUSER_WIDTH LTI_LCUSER_WIDTH	Integer	bits	Width of channel user signals.

When the value of a property results in a signal being zero bits in width, that signal is omitted from the interface.

Chapter 4
Request channel

This chapter defines the LTI request (LA) channel:

4.1 Signals

The signals in the LA channel are described in Table 4-1.

Table 4-1 Request channel signals

Signal	Category	Width	Description
LAVALID	Transport	1	Channel valid. When this signal is LOW, other TX signals on the LA channel are not valid.
LAVC	Transport	ceil($\log_2$(LTI_VC_COUNT))	Virtual Channel number.
LACREDIT	Transport	LTI_VC_COUNT	Channel credit grant. **LACREDIT** is an RX signal that flows in the other direction from other LA channel signals. It is not affected by **LAVALID**.
LAID	Flow	LTI_ID_WIDTH	Translation request ID. The ID must not match the ID of a previous request on the same VC that has not yet returned its LR channel response, unless **LAOGV**=1 in both requests and the value of **LAOG** is the same for both requests.
LAOGV	Flow	1	Order group valid. **0** No ordering required **1** The responses of all requests with this **LAOG** value must be returned in order. This signal is present even if LTI_OG_WIDTH=0. In this case, there is a single order group, and **LAOGV** indicates whether each LTI request is ordered or unordered.
LAOG	Flow	LTI_OG_WIDTH	Order group. When **LAOGV** is LOW, this signal is not valid.
LAFLOW	Flow	2	Indicates the translation flow required. **0: Stall** The SMMU stall fault flow can be used for this request, when it is enabled. **1: ATST** The transaction has already been translated by ATS. **2: NoStall** If a translation fault occurs, then even if the SMMU has enabled stall faulting for this translation context, a fault response is returned without dependence on software activity. **3: PRI** If a translation fault occurs, a fault response is returned indicating that a PRI request might resolve the fault. Architecturally the request is treated as an ATS request, and translation faults do not result in an event record. This option is for use by PCIe enumerated endpoints. For more information, see *PRI flow* on page 1-23. If this field is not Stall, then the slave must return the response in reasonable time, without dependence on software activity. It must not be Stall for PCIe masters.
LASECSID	Context	1	StreamID security level **0** Non-secure StreamID **1** Secure StreamID When **LAFLOW** is ATST, this signal must be 0.

Table 4-1 Request channel signals (continued)

Signal	Category	Width	Description
LASID	Context	LTI_SID_WIDTH	StreamID.
LASSIDV	Context	LTI_SSID_WIDTH > 0 ? 1 : 0	SubstreamID valid. When **LAFLOW** is ATST, this signal must be 0.
LASSID	Context	LTI_SSID_WIDTH	SubstreamID. When **LASSIDV** is LOW, this signal must be 0.
LAPROT	Transaction	3	Protection information. **LAPROT** uses the same encoding as AXI **AxPROT**. **LAPROT[0]: PnU** **0** Unprivileged access **1** Privileged access **LAPROT[1]: NS** **0** Secure access **1** Non-secure access **LAPROT[2]: InD** **0** Data access **1** Instruction access If **LATRANS** is SPEC, **LAPROT[0]** must be 0. If **LASECSID**=0, **LAPROT[1]** must be 1. If **LATRANS** is W, RW, SPEC, W-CMO, DCP, or W-DCP, **LAPROT[2]** must be 0. If **LAFLOW** is ATST, **LAPROT[0]** must be 0. If **LAFLOW** is ATST, **LAPROT[2]** must be 0.
LAADDR	Transaction	64	Address. This signal is always 64 bits because virtual addresses are always 64 bits, even if the system address bus is narrower. **LAADDR[11:0]** does not affect the translation result, but is used to provide information to software when a translation fault occurs.
LATRANS	Transaction	4	Type of the transaction that the LTI request is translating. See *Transaction types* on page 4-37.
LAATTR	Transaction	4	Attributes for the untranslated transaction. See *Transaction attributes* on page 4-38.

Table 4-1 Request channel signals (continued)

Signal	Category	Width	Description
LALOOP	Impdef	LTI_LOOP_WIDTH	IMPLEMENTATION DEFINED loopback signaling.
LATLBLOC	Impdef	LTI_TLBLOC_WIDTH	Location to access in the TLB. This meaning of this signal is IMPLEMENTATION DEFINED. The intended use of this signal is to control allocation and lookup in a TLB. For example, an implementation might guarantee that if a request is made with a particular value of **LATLBLOC**, and this request is followed by another request with the same value of **LATLBLOC**, then any translation that is cached from the first request is available to be used by the second request. Alternatively, **LATLBLOC** might be used to indicate a portion of a TLB to use. The LTI specification does not provide any guarantees about any such functionality.
LAUSER	Impdef	LTI_LAUSER_WIDTH	IMPLEMENTATION DEFINED additional signaling.

4.2 Transaction types

Table 4-2 describes the valid values of **LATRANS** and how those values correspond to types of translated transaction.

Table 4-2 Request channel transactions

Encoding	Name	Definition
0	SPEC	Speculative request. SPEC is intended to prefetch a translation for later use and is not used by a transaction. **Note** The DTI specification permits speculative translation requests being used for speculative read transactions in some circumstances. The LTI SPEC transaction type cannot be used for speculative read transactions. The LTI SPEC transaction type prefetches a translation into translation caches and the translation that is returned cannot be used by transactions. If an LTI master requires a speculative read transaction that is guaranteed not to fault, then it should issue an LTI request with **LATRANS**=R, **LAFLOW**=PRI.
1	R	Read.
2	W	Write.
3	RW	Read and write. RW is intended for atomic operations. Operations that perform any form of read-modify-write sequence on data in memory are defined as atomic operations, even if no read data is returned to the upstream master. For example, the AXI AtomicStore transaction type is atomic and must use **LATRANS**=RW, even though it does not return read data.
4	CMO	Cache Maintenance Operation.
5	R-CMO	Read with Cache Maintenance Operation. R-CMO is a read that also performs a Cache Maintenance Operation.
6	W-CMO	Write with Cache Maintenance Operation. W-CMO is a write that also performs a Cache Maintenance Operation.
8	DCMO	Destructive Cache Maintenance Operation. DCMO is an operation that can cause addressed cache lines to be invalidated without writeback, even if they are dirty. DCMO operations require extra permission because they can enable old versions of a memory location to be recovered after they have been overwritten.
9	R-DCMO	Read with Destructive Cache Maintenance Operation.
12	DCP	Directed Cache Prefetch. DCP is a request to prefetch data into a particular cache.
14	W-DCP	Write with Directed Cache Prefetch. W-DCP is a write that includes a request to stash the written data into a particular cache.

4.3 Transaction attributes

Table 4-3 gives the valid encodings of **LAATTR** and how those values represent the untranslated transaction attributes.

Table 4-3 Request channel transaction attributes

Encoding	Type
0	Device-nGnRnE
1	Device-nGnRE
2	Device-nGRE
3	Device-GRE
4	Normal Non-cacheable
5	Normal Inner Non-cacheable Outer Cacheable
6	Normal Write-back No-allocate Outer Shareable
7	Normal Write-back Allocate Outer Shareable
14	Normal Write-back No-allocate Non-shareable
15	Normal Write-back Allocate Non-shareable

Most transactions should use the value 7 for **LAATTR**, Normal Write-back Allocate Outer Shareable. This value is the common type for accessing normal memory locations when the translation tables have not selected a different set of transaction attributes.

When **LATRANS** is DCP, CMO, DCMO, or W-CMO then **LAATTR** must be one of the following:

- Normal Write-back No-allocate Outer Shareable
- Normal Write-back Allocate Outer Shareable
- Normal Write-back No-allocate Non-shareable
- Normal Write-back Allocate Non-shareable

When **LATRANS** is W-DCP, R-CMO, or R-DCMO then **LAATTR** must be one of the following:

- Normal Write-back No-allocate Outer Shareable
- Normal Write-back Allocate Outer Shareable

4.4 Transaction scope

An LTI translation can only be used for accesses to data within a single 4kB page. If a memory transaction accesses data in more than one 4kB page, then a separate LTI request must be made for the part of the transaction in each 4kB page.

4.5 PCIe integration

This section is informative and provides recommendations for PCIe integration.

A PCI Express (PCIe) Root Port should drive certain LA channel signals as described in Table 4-4.

Table 4-4 Request channel PCIe integration

Signal	Description
LAVC	Use different virtual channels for posted and non-posted requests. If the LTI slave supports more than two virtual channels, they can be used for different traffic classes.
LAOGV	This specification recommends the LTI request is performed before the point of ordering in the PCIe RP. If the LTI request is done, then the signal is 0, because the PCIe RP can accept LR channel responses in any order.
LAFLOW	If the request is an ATS translated request, this signal is 1, ATST. Otherwise this signal is 2, NoStall.
LASECSID	This signal is 0, Non-secure StreamID. The Secure world is not intended to be accessed by PCIe RPs.
LASID	**LASID[15:0]** is the Requestor ID, otherwise known as BDF (Bus, Device, Function). Higher-order bits of **LASID** uniquely identify the PCIe segment in the StreamID space that is used by the SMMU.
LASSIDV	If the request has a PASID header, this signal is 1, otherwise it is 0.
LASSID	This signal is the PASID.
LAPROT	**Bit [0], PnU** If the request has a PASID header and Priv=1, this bit is 1. Otherwise it is 0. **Bit [1], NS** This signal is 1. The Secure world is not intended to be accessed by PCIe RPs. **Bit [2], InD** If the request has a PASID header and Exe=1, this bit is 1. Otherwise it is 0.
LAATTR	If No_snoop is set: Normal Non-cacheable. If No_snoop is not set: Normal Write-back Allocate Outer Shareable.

Chapter 5
Response channel

This chapter defines the LTI response (LR) channel:

5.1 Signals

The signals in the LTI Response (LR) channel are described in Table 5-1.

Table 5-1 Response channel signals

Signal	Category	Width	Description
LRVALID	Transport	1	Channel valid. When this signal is LOW, other TX signals on the LR channel are not valid.
LRVC	Transport	clog2(LTI_VC_COUNT)	Virtual Channel number.
LRCREDIT	Transport	LTI_VC_COUNT	Channel credit grant. **LRCREDIT** is an RX signal that flows in the other direction from other LR channel signals. It is not affected by **LRVALID**.
LRID	Flow	LTI_ID_WIDTH	Translation request ID. The value of **LRID** must match a translation request that has not yet had a response.
LRCTAG	Flow	1	Translation completion tag. **LRCTAG** can be any value and must be returned by the LTI master with the completion.

Table 5-1 Response channel signals (continued)

Signal	Category	Width	Description
LRRESP	Translation	3	Translation response: **0: Success** The translation was successful. **1: Downgrade1** The translation was successful but the transaction type must be downgraded. The meaning of this for each transaction type is described in *Downgrade types* on page 5-45. **2: Downgrade2** The translation was successful but the transaction type must be downgraded. The meaning of this for each transaction type is described in *Downgrade types* on page 5-45. **4: FaultAbort** The translation was not successful and the transaction must be terminated. The master should indicate to the upstream device that the transaction was not successful. **5: FaultRAZWI** The translation was not successful and the transaction must be terminated. If possible, the LTI master should indicate to the requestor that the transaction was successful, by returning zero if the data was a read, and ignoring the transaction if it was a write. Cache maintenance and prefetch effects of the transaction are ignored. **6: FaultPRI** The translation was not successful but it might be resolved by issuing a PRI request. The master should issue a PRI request, and if the response from that indicates success, retry the LTI request. For more information, see *PRI flow* on page 1-23. Restrictions that are associated with this field are described in more detail in *Attribute restrictions for specific transaction types* on page 5-47.
LRPROT	Translation	3	Translated protection information. **LRPROT** uses the same encoding as **LAPROT**. If **LATRANS** is SPEC, **LRPROT[0]** must be 0. If **LASECSID**=0, **LAPROT[1]** must be 1. If **LATRANS** is W, RW, SPEC, W-CMO, DCP, or W-DCP, **LRPROT[2]** must be 0. When **LRRESP** is FaultAbort, FaultRAZWI or FaultPRI, this signal is not valid.

Table 5-1 Response channel signals (continued)

Signal	Category	Width	Description
LRADDR	Translation	LTI_LRADDR_WIDTH	Translated address. The least significant 12 bits must equal the least significant 12 bits of **LAADDR**. These bits are included in the response to avoid them needing to be included in loopback and provided in the request. When **LRRESP** is FaultAbort, FaultRAZWI or FaultPRI, this signal is not valid.
LRATTR	Translation	4	Translated transaction attributes. **LRATTR** uses the same encoding as **LAATTR**. When **LRRESP** is FaultAbort, FaultRAZWI or FaultPRI, this signal is not valid.
LRHWATTR	Translation	4	IMPLEMENTATION DEFINED hardware attributes. When **LRRESP** is FaultAbort, FaultRAZWI or FaultPRI, this signal is not valid.
LRMPAM	Translation	11	MPAM information. **LRMPAM[0]** MPAMNS **LRMPAM[9:1]** PARTID **LRMPAM[10]** PMG When **LASECSID**=0, the MPAMNS field must be 1. When **LRRESP** is FaultAbort, FaultRAZWI or FaultPRI, this signal is not valid.
LRLOOP	Impdef	LTI_LOOP_WIDTH	IMPLEMENTATION DEFINED loopback signaling. Must match the value of **LALOOP** in the request.
LRUSER	Impdef	LTI_LRUSER_WIDTH	IMPLEMENTATION DEFINED additional signaling.

5.2 LRRESP details

This section gives further information about **LRRESP**.

5.2.1 Restrictions based on LATRANS

Table 5-2 shows the permitted encodings of LRRESP, depending on the value of LATRANS in the request.

Table 5-2 Response channel LATRANS restrictions

LATRANS	LRRESP encodings permitted
SPEC	Success, FaultRAZWI, FaultPRI
R	Success, FaultAbort, FaultRAZWI, FaultPRI
W	Success, FaultAbort, FaultRAZWI, FaultPRI
RW	Success, FaultAbort, FaultRAZWI, FaultPRI
CMO	Success, FaultAbort, FaultRAZWI, FaultPRI
R-CMO	Success, Downgrade1, FaultAbort, FaultRAZWI, FaultPRI
W-CMO	Success, Downgrade1, FaultAbort, FaultRAZWI, FaultPRI
DCMO	Success, Downgrade2, FaultAbort, FaultRAZWI, FaultPRI
R-DCMO	Success, Downgrade1, Downgrade2, FaultAbort, FaultRAZWI, FaultPRI
DCP	Success, FaultRAZWI, FaultPRI
W-DCP	Success, Downgrade1, FaultAbort, FaultRAZWI, FaultPRI

5.2.2 Downgrade types

The LTI response might require the transaction type to be downgraded when the translation gives permission for part of the operation but does not give permission for another part of the operation. Table 5-3 shows the transaction type changes when a downgrade response is received.

Table 5-3 Response channel downgrade types

LATRANS	LRRESP	Translated transaction type	Effect
R-CMO	Downgrade1	R	The CMO part of the operation is not performed
W-CMO	Downgrade1	W	The CMO part of the operation is not performed
DCMO	Downgrade2	CMO	Change to the equivalent non-destructive CMO
R-DCMO	Downgrade1	R	The CMO part of the operation is not performed
R-DCMO	Downgrade2	R-CMO	Change to the equivalent non-destructive R-CMO
W-DCP	Downgrade1	W	The directed cache prefetch part of the operation is not performed

5.2.3 Restrictions based on LAFLOW

The permitted encodings of **LRRESP** depend on the value of **LAFLOW** in the request as follows:

Table 5-4 Response channel restrictions based on LAFLOW

LAFLOW	LRRESP encodings permitted
Stall	Success, Downgrade1, Downgrade2, FaultAbort, FaultRAZWI
NoStall	Success, Downgrade1, Downgrade2, FaultAbort, FaultRAZWI
PRI	Success, Downgrade1, Downgrade2, FaultAbort, FaultRAZWI, FaultPRI
ATST	Success, Downgrade1, Downgrade2, FaultAbort, FaultRAZWI

5.3 Attribute restrictions for specific transaction types

Some transaction types require certain attributes. The following table gives restrictions that apply to **LRATTR** depending on the transaction type that is given by combining **LATRANS** and **LRRESP**, taking account of any downgrade required by **LRRESP**, where required.

Table 5-5 Attribute restrictions for specific transaction types

Transaction type after downgrade	Legal values of LRATTR
CMO, DCMO	Normal Write-back Allocate Outer Shareable,
	Normal Write-back Allocate Non-shareable
R-CMO, R-DCMO, W-DCP	Normal Write-back No-Allocate Outer Shareable,
	Normal Write-back Allocate Outer Shareable
W-CMO, DCP	Normal Write-back No-Allocate Outer Shareable,
	Normal Write-back Allocate Outer Shareable,
	Normal Write-back No-Allocate Non-shareable,
	Normal Write-back Allocate Non-shareable

Chapter 6
Completion channel

This chapter defines the LTI completion (LC) channel:

6.1 Signals

Table 6-1 describes the signals in the LC channel.

Table 6-1 Completion channel signals

Signal	Category	Width	Description
LCVALID	Transport	1	Channel valid. When this signal is LOW, other TX signals on the LC channel are not valid.
LCCREDIT	Transport	1	Channel credit grant. **LCCREDIT** is an RX signal which flows in the other direction from other LC channel signals. It is not affected by **LCVALID**.
LCCTAG	Flow	1	Translation completion tag. **LCCTAG** must match the value that is given in **LRCTAG**.
LCUSER	Impdef	LTI_LCUSER_WIDTH	IMPLEMENTATION DEFINED additional signaling.

6.2 Completion channel characteristics

Completions can be returned in any order.

6.2.1 Deadlock avoidance

When a translation response is received by an LTI master, the completion must be returned without dependence on either:

- Subsequent software activity.
- Return of other translation responses for which **LAFLOW**=Stall.

6.2.2 Use of LTI translations for multiple transactions

In most cases, each untranslated transaction causes an LTI request and response. However, it is possible to use a single LTI transaction for multiple untranslated transactions if the following conditions apply:

- All data that is accessed by the transactions is within the same single 4KB address region.
- All Context and Transaction fields in the translation request would be the same for all transactions.
- The completion is still returned on the LC channel in reasonable time.

All transactions using the translation must be globally observed before the completion message is sent by the LTI master.

CPU barrier instructions that must wait for completion of an invalidation might block while awaiting LC messages. The block may occur even if the translation used for that transaction has not been invalidated. If LC messages take a long time to return, it can impact CPU performance.

This specification recommends that a single LTI response is only shared between transactions that can be issued together and are all outstanding at the same time, so that the overall latency to return the LC message is not substantially larger than the DRAM access latency. In most situations, it is better for the LTI master to rely on the translation caching of the LTI slave and use a separate LTI request for each transaction, even if the LTI master knows that the translation is likely to be reused.

Chapter 7
Interface management

This chapter describes the LTI interface management function:

7.1 Interface management overview

Signals with the LM prefix manage opening and closing the interface to enable power and clock control.

Table 7-1 shows the interface management signals.

Table 7-1 Interface management signals

Signal	Driven by	Width	Description
LMOPENREQ	Master	1	LTI interface open request
LMOPENACK	Slave	1	LTI interface open acknowledge
LMACTIVE	Master	1	Request to open the interface or keep it open
LMASKCLOSE	Slave	1	Request to close the interface

7.2 Open and close handshake

The **LMOPENREQ** and **LMOPENACK** signals control the LTI interface state. The LTI interface can be in one of the following four states:

Table 7-2 LTI states

Interface state	LMOPENREQ	LMOPENACK
ST_CLOSED	0	0
ST_OPENING	1	0
ST_OPEN	1	1
ST_CLOSING	0	1

The LTI interface transitions through the states in the following order:

1. **LMOPENREQ** can transition from LOW to HIGH only if **LMOPENACK** is LOW.
2. **LMOPENREQ** can transition from HIGH to LOW only if **LMOPENACK** is HIGH.
3. **LMOPENACK** can transition from LOW to HIGH only if **LMOPENREQ** is HIGH.
4. **LMOPENACK** can transition from HIGH to LOW only if **LMOPENREQ** is LOW.

There must be no combinatorial paths between **LMOPENACK** and **LMOPENREQ** in either direction. A consequence of this is that the interface must spend at least one cycle in each state before moving to the next state.

Upon reset, the LTI interface is in state ST_CLOSED.

Figure 7-1 shows how **LMOPENREQ** and **LMOPENACK** correspond to the interface states.

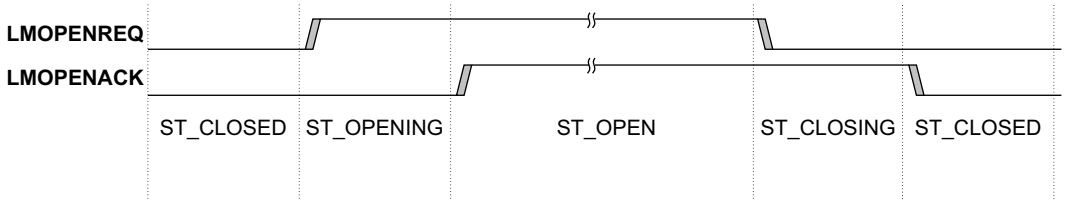


Figure 7-1 Interface states

7.3 Properties of interface states

The master can only issue new requests on the LA channel when in state ST_OPEN. **LAVALID** and **LCVALID** must be deasserted when **LMOPENREQ** is deasserted.

The master must wait for all outstanding responses on the LR channel and send all outstanding completions on the LC channel before transitioning from state ST_OPEN to state ST_CLOSING. **LMOPENREQ** must be asserted while there are outstanding transactions on the interface. The master must not deassert **LMOPENREQ** until the cycle after it has asserted **LCVALID** for the completion.

LACREDIT and **LCCREDIT** can only be asserted in states ST_OPEN and ST_CLOSING. **LACREDIT** and **LCCREDIT** must be deasserted when **LMOPENACK** is deasserted.

LRCREDIT can only be asserted in state ST_OPEN. **LRCREDIT** must be deasserted when **LMOPENREQ** or **LMOPENACK** are deasserted.

All credits on all channels are lost when in state ST_CLOSED. When the state ST_OPEN is entered, each channel TX has zero credits.

7.4 Management Signals

This section describes the behavior of the interface management signals.

7.4.1 LMASKCLOSE

The **LMASKCLOSE** signal enables the slave to request that the interface is to be closed. For example, **LMASKCLOSE** may be asserted in response to a quiescence request on a Q-Channel interface on the slave.

When **LMASKCLOSE** is asserted by the slave and **LMACTIVE** is deasserted by the master, the master must, in a timely manner, either:

- Deassert **LMOPENREQ**, to move into ST_CLOSING.
- Assert **LMACTIVE**, to enable the master to issue new LTI requests. The slave will usually then deassert **LMASKCLOSE**

LMASKCLOSE must not be asserted when **LMOPENACK** is deasserted.

7.4.2 LMACTIVE

The **LMACTIVE** signal serves two purposes:

- When asserted while the interface is closed, it indicates that the slave has clock and power and the interface can be opened.
- When asserted while the interface is opened, it indicates that the slave should not request closing the interface.

There are no protocol rules linking **LMACTIVE** to outstanding LTI requests, however **LMACTIVE** is normally asserted while any LTI requests are outstanding.

LMACTIVE must be glitch-free and suitable for sampling in a different clock domain.

The slave is permitted to remain in state ST_OPENING indefinitely when **LMACTIVE** is not asserted.

The slave is permitted to assert **LMASKCLOSE** when **LMACTIVE** is asserted. This is because there might be a delay between **LMASKCLOSE** being asserted and it affecting **LMACTIVE**. However, it is expected that the slave deasserts **LMASKCLOSE** when it detects that **LMACTIVE** is asserted.

The master is permitted to assert and deassert **LMACTIVE** without asserting **LMOPENREQ**. This is because **LMACTIVE** might be driven combinatorially, before **LMOPENREQ** can be asserted.

Expected usage of LMACTIVE

A component can use an AXI **AWAKEUP** interface as a combinatorial input to **LMACTIVE**, because **AWAKEUP** is also required to be driven from a register and be glitch-free. It is expected that when **LMACTIVE** is asserted:

- If the slave implements a Q-Channel, the slave asserts **QACTIVE**.
- The slave will deny Q-Channel quiescence requests.
- The master ignores **LMASKCLOSE**.

Examples of the use of **LMACTIVE** include:

- **LMASKCLOSE** might be asserted by the slave when **LMACTIVE** is deasserted, but the new activity begins at the master before it begins to close the LTI interface. In this case, it asserts **LMACTIVE** and ignores **LMASKCLOSE**.
- The master might assert **LMACTIVE** to give early notice to the slave to start its clocks, before the master is ready for the interface to be opened.

Opening sequence example

Figure 7-2 shows an example of an interface opening sequence.

1. The master asserts **LMACTIVE** and **LMOPENREQ** to request opening the interface.
   a. The master might assert **LMACTIVE** before **LMOPENREQ**.
   b. It is permitted, but not expected, for the master to assert **LMACTIVE** after asserting **LMOPENACK**.
2. The slave asserts **LMOPENACK** to accept opening the interface, and asserts **LACREDIT** and **LCCREDIT** to send request and completion credits. These can be asserted in any cycle that **LMOPENACK** is asserted..
3. The master sees that **LMOPENACK** is asserted and in the next cycle, asserts **LRCREDIT** to send response credits and starts providing LR credits. It also uses LA credits to start issuing requests on the LA channel.

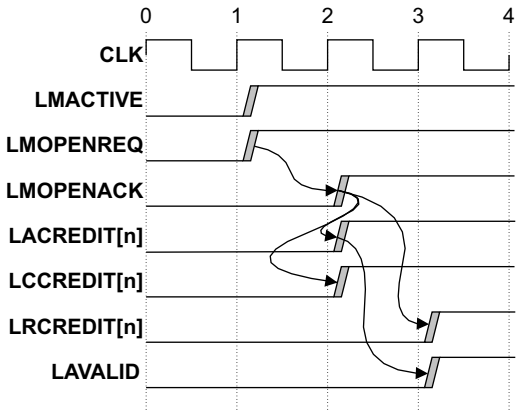


Figure 7-2 Example interface opening sequence

Closing sequence examples

Figure 7-3 shows an interface closing sequence that is initiated by the slave.

1. The slave observes **LMACTIVE** is not asserted and requests that the interface be closed by asserting **LMASKCLOSE**.
2. The master accepts the request to close the interface by deasserting **LMOPENREQ**.
3. The slave completes the close sequence by deasserting **LMOPENACK** and **LMASKCLOSE**.

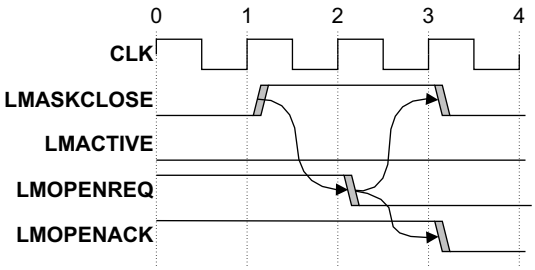


Figure 7-3 Closing sequence initiated by the slave

Figure 7-4 shows a close request that is ignored by the master.

1. The slave requests that the interface be closed by asserting **LMASKCLOSE**.

   The slave asserts **LMASKCLOSE** in the same cycle that the master asserts **LMACTIVE**, which indicates that the master has work to do, and will ignore **LMASKCLOSE**.

2. The slave observes **LMACTIVE** and withdraws the interface closing request by deasserting **LMASKCLOSE**.

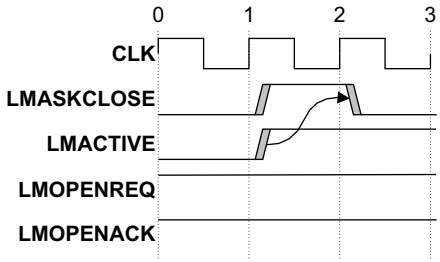


Figure 7-4 Denied interface closing sequence

The assertion of **LMACTIVE** and **LMASKCLOSE** are independent events, so there is no fixed relationship between them. Figure 7-4 shows them being asserted in the same cycle, but this is coincidental. A combinatorial path between the signals is not permitted.

Chapter 8
Clock and reset

8.1 Clock and reset

An LTI interface is associated with a clock signal and a reset signal.

Typically, an LTI interface shares its clock and reset with other interfaces on a component. For this reason, the clock and reset signals do not have an LTI-specific prefix in this specification.

An implementation might give the clock and reset signals different names from those signals used in the specification, or employ a different clock and reset strategy. This specification recommends that there is a mapping between the clock and reset signals in an implementation and the signals that are defined here.

Signals that are not defined as asynchronous are sampled on the rising edge of the clock signal.

The reset signal is asserted asynchronously and deasserted synchronously with the clock signal.

The reset signal is active LOW.

- When the reset signal is LOW, it is asserted and the other interface signals can be any value.
- When the reset signal is HIGH, it is deasserted and the interface runs normally.

On the first rising edge of the clock signal where the reset signal is HIGH, the following signals must be 0:

- **LxVALID**
- **LxCREDIT**
- **LMOPENREQ**
- **LMOPENACK**
- **LMASKCLOSE**

Chapter 9
Pipelining

This chapter defines pipelining requirements for LTI:

- *Pipelining between master and slave interfaces* on page 9-64

9.1 Pipelining between master and slave interfaces

Pipeline stages or registers can be added to the signals between LTI master and slave interfaces. The number of pipeline stages applied must comply with the following rules:

- All signals on each of the LA, LR, and LC channels must be pipelined by the same number of cycles as all other signals in the same channel going in the same direction.
- **LMOPENREQ** must be pipelined by at least as many cycles as **LAVALID**, **LRCREDIT**, and **LCVALID**.
- **LMOPENACK** and **LMASKCLOSE** must each be pipelined by the same number of cycles, and both by at least as many cycles as **LACREDIT**, **LRVALID** and **LCCREDIT**.

It is expected that in most implementations that require pipelining, all LTI signals that go in the same direction are pipelined by the same number of cycles.

Appendix A
Mapping LTI to ACE-Lite

This appendix describes how to map LTI concepts onto ACE-Lite.

A.1 Use of virtual channels

Some systems require guarantees that the AW channel can make progress when the AR channel is blocked. For example, this behavior is often required to ensure PCIe systems do not deadlock when peer-to-peer transactions occur.

To ensure independence of AW and AR channels, this specification recommends that LTI requests corresponding to AR and AW transactions are placed on different LTI virtual channels.

A system can support multiple PCIe Traffic Classes by placing them on different VCs. Two virtual channels are recommended for each Traffic Class, one for AW transactions and one for AR transactions.

A.2 LATRANS mapping

Table A-1 shows the expected mapping of ACE-Lite transactions to **LATRANS**.

Table A-1 LATRANS mapping

LATRANS	ACE-Lite AR channel transactions	ACE-Lite AW channel transactions
SPEC	-	StashTranslation
R	ReadNoSnoop, ReadOnce	-
W	-	WriteNoSnoop, WriteUnique, WriteLineUnique
RW	-	AtomicLoad, AtomicSwap, AtomicCompare, AtomicStore[a]
CMO	CleanShared, CleanInvalid, CleanSharedPersist, CleanSharedDeepPersist	CleanShared, CleanInvalid, CleanSharedPersist, CleanSharedDeepPersist
R-CMO	ReadOnceCleanInvalid	-
W-CMO	-	WriteNoSnoopCMO, WriteUniquePtlCMO, WriteUniqueFullCMO
DCMO	MakeInvalid	-
R-DCMO	ReadOnceMakeInvalid	-
DCP	-	StashOnceShared, StashOnceUnique
W-DCP	-	WriteUniquePtlStash, WriteUniqueFullStash

a. AtomicStore operations are defined as type RW, not W, even though no read data is returned. This is required by the SMMUv3 architecture.

A.3 LAATTR mapping

In a coherent memory system, all coherent masters must use consistent attributes to the same memory location. The mapping of Armv8 memory types to AMBA memory types must be done consistently so this property is not broken.

Table A-2 shows the recommended mapping of the ACE-Lite **AxCACHE** and **AxDOMAIN** signals to **LAATTR** values.

Table A-2 AxCACHE and AxDOMAIN mapping to LAATTR

AxCACHE	AxDOMAIN	LAATTR
Device Non-bufferable	System	Device-nGnRnE
Device Bufferable	System	Device-nGnRE
Normal Non-cacheable Bufferable, Normal Non-cacheable Non-bufferable	System	Normal Non-cacheable
Write-Through No-allocate[a], Write-Through Read and Write-Allocate[b]	Non-shareable, Inner Shareable, Outer Shareable,	Normal Non-cacheable
Write-Back No-allocate[a]	Inner Shareable, Outer Shareable	Normal Write-Back No-allocate Outer Shareable
Write-Back Read and Write-Allocate[b]	Inner Shareable, Outer Shareable	Normal Write-Back Allocate Outer Shareable
Write-Back No-allocate[a]	Non-shareable	Normal Write-Back No-allocate Non-shareable
Write-Back Read and Write-Allocate[b]	Non-shareable	Normal Write-Back Allocate Non-shareable

a. Includes Read-Allocate for writes, and Write-Allocate for reads
b. Includes Read-Allocate for reads, and Write-Allocate for writes

A.4 LRATTR mapping

In a coherent memory system, all coherent masters must use consistent attributes to the same memory location. The mapping of Armv8 memory types to AMBA memory types must be done consistently so this property is not broken.

Table A-3 shows the recommended mapping of the **LRATTR** values to the ACE-Lite **AxCACHE** and **AxDOMAIN** signals.

Table A-3 LRATTR mapping to AxCACHE and AxDOMAIN

LRATTR	AxCACHE	AxDOMAIN
Device-nGnRnE	Device Non-bufferable	System
Device-nGnRE	Device Bufferable	System
Device-nGRE		
Device-GRE		
Normal Non-cacheable	Normal Non-cacheable Bufferable	System
Normal Inner Non-cacheable Outer Cacheable		
Normal Write-Back No-allocate Outer Shareable	Write-back No-allocate	Outer Shareable
Normal Write-Back Allocate Outer Shareable	Write-back Read and Write-Allocate	Outer Shareable
Normal Write-Back No-allocate Non-shareable	Write-back No-allocate	Non-shareable
Normal Write-Back Allocate Non-shareable	Write-back Read and Write-Allocate	Non-shareable

If **AxBURST**=FIXED, **AxDOMAIN** is not permitted to be Outer Shareable. If an ACE-Lite transaction with **AxBURST**=FIXED translates to any Normal Write-Back memory type, this specification recommends being output as Non-shareable. This specification recommends that masters that are translated by an SMMU do not use the FIXED burst type.

If **AxLOCK**=1, **AxCACHE** is not permitted to indicate a cacheable type without additional knowledge of the structure of the downstream interconnect. If an ACE-Lite transaction with **AxLOCK**=1 translates to any Normal Write-Back memory type, this specification recommends that it be output with **AxLOCK**=0. This specification recommends that masters requiring semaphore-type operations use the AXI5 atomic transactions.

If **AWSNOOP**=WriteUniqueFull, **AWDOMAIN** is required to be Inner Shareable or Outer Shareable. If translation of a WriteUniqueFull transaction results in an **AWDOMAIN** of Non-shareable or System, then **AWSNOOP** should be output as WriteNoSnoop.

A.5 xRESP mapping

Table A-4 shows the relative mapping of **LRRESP** to ACE-Lite **RRESP** and **BRESP** signals when the transaction is terminated by the LTI master.

Table A-4 LRRESP to xRESP mapping

LRRESP	xRESP
Success, Downgrade1, Downgrade2	**xRESP** is propagated from the downstream transaction
FaultAbort	SLVERR
FaultRAZWI	OKAY
FaultPRI	TRANSFAULT

A.6 Transactions that are legal in ACE-Lite and illegal in LTI

Certain transactions are legal in ACE-Lite but illegal in LTI. The following modifications are recommended to be made to transactions so that they are legal in LTI:

- Write transactions marked as Instruction (**AWPROT[2]** = 1) should be converted to Data (**AWPROT[2]** = 0).
- ATST transactions (**AxMMUFLOW** = ATST) marked as Instruction (**AxPROT[2]** = 1) should be converted to Data (**AxPROT[2]** = 0).
- ATST transactions (**AxMMUFLOW** = ATST) marked as Privileged (**AxPROT[0]** = 1) should be converted to Unprivileged (**AxPROT[0]** = 0).

Appendix B
LTI Mapping to DTI

This appendix describes how to map LTI concepts onto DTI-TBU.

B.1 DTI_TBU_TRANS_REQ.PERM mapping

The following table shows how the value of **LATRANS** in an LTI request is expected to map to a value of DTI_TBU_TRANS_REQ.PERM in a DTI request.

Table B-1 DTI_TBU_TRANS_REQ.PERM mapping

LATRANS	DTI_TBU_TRANS_REQ.PERM
SPEC, DCP	SPEC
R, CMO, R-CMO, DCMO, R-DCMO	R
W, W-DCP	W
RW, W-CMO	RW

B.2 Special handling for specific LATRANS values

Some transaction types require special handling when calculating permissions and attributes.

B.2.1 LATRANS = DCP

A cache stashing operation is only meaningful for cacheable shareable memory locations. If the final transaction attributes are not Normal Write-Back, convert **LRRESP**=Success into **LRRESP**=FaultRAZWI. The allocate hint does not affect **LRRESP**.

If DCP permission is not granted, convert **LRRESP**=Success into **LRRESP**=FaultRAZWI.

If R, W or X permission are not granted at the appropriate privilege level, convert **LRRESP**=Success into **LRRESP**=FaultRAZWI.

B.2.2 LATRANS = W-DCP

A cache stashing operation is only meaningful for cacheable shareable memory locations. If the final transaction attributes are not Normal Write-Back, or if the final shareability is Non-shareable, convert **LRRESP**=Success into **LRRESP**=Downgrade1.

If DCP permission is not granted, convert **LRRESP**=Success into **LRRESP**=Downgrade1.

B.2.3 LATRANS = CMO

Cache Maintenance Operations do not have a memory type. If the calculated shareability is Non-shareable, return **LRATTR** = Normal Write-Back Allocate Non-shareable, otherwise return **LRATTR** = Normal Write-Back Allocate Outer Shareable.

B.2.4 LATRANS = R-CMO

R-CMOs are intended to operate on cacheable shareable memory locations. If the final transaction attributes are not Normal Write-Back, or if the final shareability is Non-shareable, convert **LRRESP**=Success into **LRRESP**=Downgrade1. The allocate hint does not affect **LRRESP**.

B.2.5 LATRANS = W-CMO

CMOs are only meaningful when operating on cacheable memory locations. If the final transaction attributes are not Normal Write-Back, convert **LRRESP**=Success into **LRRESP**=Downgrade1. The shareability and allocate hint do not affect **LRRESP**.

B.2.6 LATRANS = DCMO

Cache Maintenance Operations do not have a memory type. If the calculated shareability is Non-shareable, return **LRATTR** = Normal Write-Back Allocate Non-shareable, otherwise return **LRATTR** = Normal Write-Back Allocate Outer Shareable.

If either W permission is not granted at the appropriate privilege level or DRE permission is not granted, convert **LRRESP**=Success into **LRRESP**=Downgrade2.

B.2.7 LATRANS = R-DCMO

R-DCMOs are intended to operate on cacheable shareable memory locations. If the final transaction attributes are not Normal Write-Back, or if the final shareability is Non-shareable, convert **LRRESP**=Success into **LRRESP**=Downgrade1. The allocate hint does not affect **LRRESP**.

Otherwise, if either W permission is not granted at the appropriate privilege level or DRE permission is not granted, convert **LRRESP**=Success into **LRRESP**=Downgrade2.

B.3 Attribute mapping

This section describes the mapping between LTI attributes and Armv8 attributes used by DTI.

Table B-2 shows the meanings of the abbreviations used in this section.

Table B-2 Attribute abbreviations

Abbreviation	Meaning
iNC	inner Non-cacheable
iWT	inner Write-Through
iWB	inner Write-Back
oNC	outer Non-cacheable
oWT	outer Write-Through
oWB	outer Write-Back attributes

B.3.1 LTI to Armv8 conversion

In all cases the ARMv8 type allocate and transient hints are the same for inner and outer cacheability domains.

Table B-3 LTI to Armv8 attribute conversion

LTI Attribute	Armv8 type	Armv8 shareability
Device-nGnRnE	Device-nGnRnE	Outer Shareable
Device-nGnRE	Device-nGnRE	Outer Shareable
Device-nGRE	Device-nGRE	Outer Shareable
Device-GRE	Device-GRE	Outer Shareable
Normal Non-cacheable	Normal-iNC-oNC	Outer Shareable
Normal Inner Non-cacheable Outer Cacheable		
Normal Write-Back No-allocate Outer Shareable	Normal-iWB-oWB Read No-allocate Write No-allocate Non-transient	Outer Shareable
Normal Write-Back Allocate Outer Shareable	Normal-iWB-oWB Read-Allocate Write-Allocate Non-transient	Outer Shareable
Normal Write-Back No-allocate Non-shareable	Normal-iWB-oWB Outer Write-Back Read No-allocate Write No-allocate Non-transient	Non-shareable
Normal Write-Back Allocate Non-shareable	Normal-iWB-oWB Read-Allocate Write-Allocate Non-transient	Non-shareable

B.3.2 ARMv8 to LTI conversion

Table B-4 Armv8 to LTI attribute conversion

ARMv8 type	ARMv8 shareability	LTI Attribute
Device-nGnRnE	Outer Shareable	Device-nGnRnE
Device-nGnRE	Outer Shareable	Device-nGnRE
Device-nGRE	Outer Shareable	Device-nGRE
Device-GRE	Outer Shareable	Device-GRE
Normal-iNC-oNC, Normal-iWT-oNC, Normal-iWB-oNC	Outer Shareable	Normal Non-cacheable
Normal-iNC-oWT, Normal-iWT-oWT, Normal-iWB-oWT, Normal-iNC-oWB, Normal-iWT-oWB	Non-shareable, Outer Shareable, Inner Shareable	Normal Inner Non-cacheable Outer Cacheable
Normal-iWB-oWB	Outer Shareable, Inner Shareable	Either: • Normal Write-Back No-allocate Outer Shareable • Normal Write-Back Allocate Outer Shareable Depending on the transaction type and Armv8 Outer Read-Allocate and Write-Allocate hints, as described in Table B-5.
Normal-iWB-oWB	Non-shareable	Either: • Normal Write-Back No-allocate Non-shareable • Normal Write-Back Allocate Non-shareable Depending on the transaction type and Armv8 Outer Read-Allocate and Write-Allocate hints, as described in Table B-5.

For transactions with Armv8 memory type Normal-iWB-oWB, the choice between No-allocate and Allocate in LTI is chosen as follows:

Table B-5 Armv8 to LTI allocation hint mapping

LATRANS	LTI Allocation hint depends on
R, R-CMO, R-DCMO	ARMv8 Outer Read-Allocate
W, W-DCP, RW, W-CMO, DCP	ARMv8 Outer Write-Allocate
CMO, DCMO, SPEC	Always allocate

Although this section defines an allocation hint for all transaction types, it is ignored by the system for many transactions. For example:

- SPEC transactions are terminated after translation is complete, and so the allocation hint is ignored.
- CMO, R-CMO, DCMO, R-DCMO, and W-CMO transactions are designed to de-allocate memory locations from caches and so the allocation hint is expected to be ignored by the system.

- DCP and W-DCP transactions are explicit cache allocating transactions, so the allocation hint is expected to be ignored by the system.

B.4 DTI_TBU_TRANS_FAULT.TYPE mapping

Table B-6 DTI_TBU_TRANS_FAULT.TYPE mapping

DTI_TBU_TRANS_FAULT.TYPE	LRRESP
NonAbort	FaultRAZWI
Abort	FaultAbort This value of DTI_TBU_TRANS_FAULT.TYPE does not occur when **LATRANS** = SPEC or DCP.
StreamDisabled	If (**LATRANS** = SPEC or DCP) FaultRAZWI else FaultAbort
GlobalDisabled	If (**LATRANS** = SPEC or DCP) FaultRAZWI else FaultAbort
TranslationPRI	FaultPRI
TranslationStall	N/A