AMBA® DTI

Protocol Specification

arm

ARM IHI 0088E (ID082620)

AMBA DTI
Protocol Specification

Release Information

The following changes have been made to this specification:

Change history

Date	Issue	Confidentiality	Change
18 November 2016	0000-00	Non-Confidential	Edition 0 (First release)
09 May 2017	0000-01	Non-Confidential	Edition 1
11 September 2017	0000-02	Non-Confidential	Edition 2
13 July 2018	0000-03	Non-Confidential	Edition 3
27 August 2020	E	Non-Confidential	Addition of v2 protocols

Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.
LES-PRE-21451 version 2.2

AMBA SPECIFICATION LICENCE

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

"LICENSEE" means You and your Subsidiaries.

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

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

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

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

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

1. 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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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 DTI Protocol Specification

Preface

This preface introduces the *AMBA Distributed Translation Interface Protocol Specification*.

It contains the following:

About this specification

Intended audience

This specification is intended for the following audiences:

- Root Complex designers implementing ATS functionality.
- Designers of components implementing TBU functionality.

Using this specification

This book is organized into the following chapters:

Chapter 1 *Introduction*

This chapter introduces the DTI protocol.

Chapter 2 *DTI Protocol Overview*

This chapter provides an overview of the DTI protocol.

Chapter 3 *DTI-TBU Messages*

This chapter describes the message groups of the DTI-TBU protocol.

Chapter 4 *DTI-TBU Caching Model*

This chapter describes the caching model for the DTI-TBU protocol.

Chapter 5 *DTI-ATS Messages*

This chapter describes the message groups of the DTI-ATS protocol.

Chapter 6 *Transport Layer*

This chapter describes the transport layer of the DTI protocol.

Appendix A *Pseudocode*

This appendix provides example implementations of the requirements specified in this document.

Conventions

The following sections describe conventions that this specification can use:

- *Typographic conventions*
- *Signals* on page xiii
- *Numbers* on page xiii

Typographic conventions

The typographical conventions are:

italic	Highlights important notes, introduces special terminology, and indicates 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.

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.

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

Lower-case 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

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

Arm publications

- *Arm AMBA Distributed Translation Interface (DTI) Protocol Specification Edition 3* (100225_0000_03)
- *Arm® System Memory Management Unit Architecture Specification*
  SMMU architecture versions 3.0, 3.1 and 3.2 (IHI 0070C)
- *AMBA® 4 AXI4-Stream Protocol*(IHI 0051A)

Other publications

- *PCI Express Base Specification, Revision 5*, PCI-SIG
- Compute Express Link Specification, Compute Express Link™ Consortium, Inc., Revision 1

Feedback

Arm welcomes feedback on its documentation.

Feedback on this specification

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

- The title, AMBA DTI Protocol Specification
- The number, ARM IHI 0088E
- 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 DTI protocol.

It contains the following section:

1.1 About DTI protocols

This section introduces the AMBA Distributed Translation Interface (DTI) protocols and describes the components of a DTI-compliant implementation.

The DTI protocol is used by implementations of the *Arm® System MMUv3* (SMMUv3) *Architecture Specification*. An SMMUv3 implementation that is built using the DTI interface consists of the following components:

- A Translation Control Unit (TCU) that performs translation table walks and implements the SMMUv3 programmers' model.
- At least one Translation Buffer Unit (TBU). The TBU intercepts transactions in need of translation and translates the addresses of those transactions. The TBU requests translations from the TCU and caches those translations for use by other transactions. The TCU communicates with the TBU to invalidate cached translations when necessary.
- A PCI Express (PCIe) Root Port with Address Translation Services (ATS) support. For more information, see the PCI Express Base Specification. When PCIe ATS functionality is required, this component communicates directly with the TCU to retrieve ATS translations, and then uses a TBU to:
  - Translate transactions that have not already been translated using ATS.
  - Perform stage 2 translation for transactions that have been subject to stage 1 translation using ATS.
  - Ensure that only trusted PCIe endpoints can issue transactions with ATS translations, by performing security checks on ATS translated traffic.
- A DTI interconnect that manages the communication between TBUs and the TCU, and between PCIe Root Ports implementing ATS and the TCU.

This specification specifies two protocols, which have different purposes:

- DTI-TBU protocol defines communication between a TBU and a TCU.
- DTI-ATS protocol defines communication between a PCIe Root Port and a TCU.

These two protocols are collectively termed the DTI protocol. Version 2 of the two protocols (DTIv2) adds to, or changes some functionality of, Version 1 (DTIv1). When there are differences between versions, the following conventions are used:

DTI-TBUv1 Describes DTI-TBU version 1

DTI-TBUv2 Describes DTI-TBU version 2

DTI-ATSv1 Describes DTI-ATS version 1

DTI-ATSv2 Describes DTI-ATS version 2

1.1.1 Protocol interaction

The DTI protocol is a point-to-point protocol. Each channel consists of a link between a TBU or PCIe Root Port implementing ATS, and a TCU.

Components using the SMMU must provide the correct StreamID and SubstreamID. For ATS translated transactions, a PCIe Root Port must provide additional information.

Figure 1-1 on page 1-19 shows an example SMMU system that implements DTI.

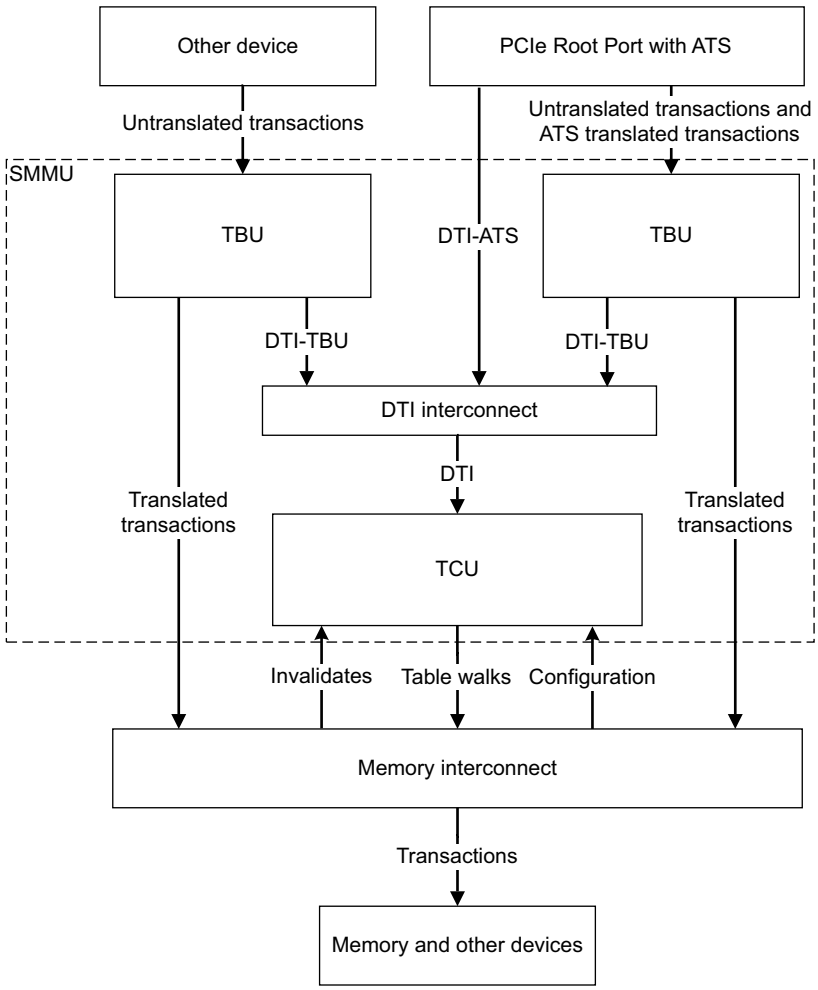


Figure 1-1 An example SMMU system

Figure 1-1 includes the necessary components of a DTI-compliant implementation. However, DTI connections can cover large distances across an SoC. Most implementations do not include a standalone SMMU component. DTI allows an implementation to distribute the functions of the SMMU across the SoC, with TBUs located close to the devices that require translation.

It is possible for a device to implement its own TBU functionality. This allows the following behavior:

- A device can incorporate advanced or specialized prefetching or translation caching requirements that cannot be met by a general-purpose TBU design.
- A device that can require a fully coherent connection to the memory interconnect and require very low latency translation. For fully coherent operations, all caches in the device must be tagged with physical addresses. This requires that translation is performed before the first level of caching. In such systems, the translation must be fast and is normally tightly integrated into the design of the device.

1.1.2 Field references

The behavior or values returned by the component sometimes depends on previous messages. Since some message pairs have the same field names, it is necessary to specify which message has the field (FIELD) being referenced. Fields from the corresponding message (MSG) are referenced as "MSG.FIELD". Fields from the message are described are referenced as FIELD, without the qualifier.

1.2 DTI Protocol Specification Terminology

This document uses the following terms and abbreviations.

ASID

Address Space ID, distinguishing TLB entries for separate address spaces. For example, address spaces of different PE processes are distinguished by ASID.

ATS

PCI Express term, Address Translation Services, which are provided for remote endpoint TLBs.

Downstream

A direction of information flow where the information is flowing away from the TBU or the Root Complex

DTI-ATSv1

Describes characteristics of DTI-ATS version 1 that are different from subsequent versions.

DTI-ATSv2

Describes characteristics of DTI-ATS version 2 that are different from previous versions.

DTI-TBUv1

Describes characteristics of DTI-TBU version 1 that are different than subsequent versions.

DTI-TBUv2

Describes characteristics of DTI-TBU version 2 that are different than previous versions.

E2H

EL2 Host mode. The Virtualization Host Extensions, introduced in *Arm Architecture Reference Manual ARMv8, for ARMv8-A architecture profile issue B*, extend the EL2 translation regime providing ASID-tagged translations.

Endpoint

A PCI Express function, which is used in the context of a device that is a client of the SMMU.

HTTU

Hardware Translation Table Update. The act of updating the Access flag or Dirty state of a page in a given TTD that is automatically done in hardware on an access or write to the corresponding page.

IMPLEMENTATION DEFINED

Means that the behavior is not architecturally defined, but must be defined and documented by individual implementations.

IPA

Intermediate Physical Address.

PA

Physical Address.

PASID

PCI Express term: Process Address Space ID, an endpoint-local ID. There might be many distinct uses of a specific PASID value in a system.

PCI

Peripheral Component Interconnect specification

PCIe

PCI Express

PCIe Root Complex

A PCIe System Element that includes at least one Host Bridge, Root Port, or Root Complex Integrated Endpoint.

PCIe RP

A port on a PCIe Root Complex

PRI

ATS Page Request Interface mechanism.

SMMU

System MMU. Unless otherwise specified, this term is used to mean SMMUv3.

StreamWorld

SMMUv3 translations have a StreamWorld property that denotes the translation regime and is directly equivalent to an Exception level on a PE.

StreamID

A StreamID uniquely identifies a stream of transactions that can originate from different devices, but are associated with the same context.

SubstreamID

A SubstreamID might optionally be provided to an SMMU implementing stage 1 translation.

The SubstreamID differentiates streams of traffic originating from the same logical block in order to associate different application address translations to each.

Upstream

A direction of information flow where the information is flowing towards the TBU or Root Complex.

VA

Virtual address.

VMID

Virtual Machine ID, distinguishing TLB entries for addresses from separate virtual machines.

Chapter 2
DTI Protocol Overview

This chapter is an overview of the DTI protocol.

It contains the following sections:

2.1 DTI protocol messages

This section contains the following subsections:

2.1.1 Message groups

DTI protocol messages are grouped according to function. Table 2-1 shows the DTI message groups.

Table 2-1 Message groups of the DTI Protocol

Message group	Direction of first message	DTI-TBU protocol function	DTI-ATS protocol function
Connection and disconnection	Downstream	Establishes or terminates the connection	Establishes or terminates the connection
Translation request	Downstream	Retrieves a non-ATS translation Performs permission checks and Stage 2 translations, if necessary, on translations that have been translated by ATS	Retrieves an ATS translation.
Invalidation and synchronization	Upstream	Invalidates cached translations	Invalidates cached ATS translations
Page request	Downstream	-	Requests that pages are available using the *ATS Page Request Interface* (PRI) mechanism
Register access	Upstream	Provides access to local IMPLEMENTATION DEFINED registers	-

2.1.2 Message listing

DTI messages are fixed length and have a whole number of bytes in size. The transport medium must preserve the correct number of bytes for each message.

The four least-significant bits of every message are used to encode the message-type.

Some message types include a protocol field. In that case, the messageis identified by the combination of its message-type and protocol field values.

The message-type encodings are defined independently for upstream and downstream messages.

DTI-TBU protocol downstream messages

The following table shows the downstream messages of the DTI-TBU protocol.

Table 2-2 DTI-TBU protocol downstream messages

Message group	Message	MST_MSG_TYPE field encoding	Message length in bits
Connection and disconnection.	DTI_TBU_CONDIS_REQ	0x0	32
Translation request.	DTI_TBU_TRANS_REQ	0x2	160
Invalidation and synchronization.	DTI_TBU_INV_ACK	0x4	8
	DTI_TBU_SYNC_ACK	0x5	8
Register access.	DTI_TBU_REG_WACK	0x6	8
	DTI_TBU_REG_RDATA	0x7	64
IMPLEMENTATION DEFINED. .	-	0xE	-
	-	0xF	-

DTI-TBU protocol upstream messages

The following table shows the upstream messages of the DTI-TBU protocol.

Table 2-3 DTI-TBU protocol upstream messages

Message group	Message	SLV_MSG_TYPE field encoding	Message length in bits
Connection and disconnection.	DTI_TBU_CONDIS_ACK	0x0	32
Translation request.	DTI_TBU_TRANS_FAULT	0x1	32
	DTI_TBU_TRANS_RESP	0x2	160
Invalidation and synchronization.	DTI_TBU_INV_REQ	0x4	128
	DTI_TBU_SYNC_REQ	0x5	8
Register access.	DTI_TBU_REG_WRITE	0x6	64
	DTI_TBU_REG_READ	0x7	32
IMPLEMENTATION DEFINED.	-	0xE	-
	-	0xF	-

DTI-ATS protocol downstream messages

The following table shows the downstream messages of the DTI-ATS protocol.

Table 2-4 DTI-ATS protocol downstream message

Message group	Message	MST_MSG_TYPE field encoding	Message length in bits
Connection and disconnection.	DTI_ATS_CONDIS_REQ	0x0	32
Translation request.	DTI_ATS_TRANS_REQ	0x2	160
Invalidation and synchronization.	DTI_ATS_INV_ACK	0xC	8
	DTI_ATS_SYNC_ACK	0xD	8
Page request.	DTI_ATS_PAGE_REQ	0x8	128
IMPLEMENTATION DEFINED.	-	0xE	-
	-	0xF	-

DTI-ATS protocol upstream message

The following table shows the upstream messages of the DTI-ATS protocol.

Table 2-5 DTI-ATS protocol upstream messages

Message group	Message	SLV_MSG_TYPE field encoding	Message length in bits
Connection and disconnection.	DTI_ATS_CONDIS_ACK	0x0	32
Translation request.	DTI_ATS_TRANS_FAULT	0x1	32
	DTI_ATS_TRANS_RESP	0x2	160
Invalidation and synchronization.	DTI_ATS_INV_REQ	0xC	128
	DTI_ATS_SYNC_REQ	0xD	8
Page request.	DTI_ATS_PAGE_ACK	0x8	8
	DTI_ATS_PAGE_RESP	0x9	96
IMPLEMENTATION DEFINED	-	0xE	-
	-	0xF	-

IMPLEMENTATION DEFINED messages

Messages with bits [3:0] equal to 0xE or 0xF can be used for IMPLEMENTATION DEFINED purposes.

IMPLEMENTATION DEFINED messages must only be exchanged between components that are designed to expect them when in permitted channel states. See *Channel states* on page 2-28. The mechanism for discovering this, if required, is IMPLEMENTATION DEFINED.

2.1.3 Flow control

The DTI protocol uses tokens to provide flow control. The tokens are used to manage the number of messages of different types that can be outstanding at a point in time.

The DTI protocol uses the following types of tokens:

Translation tokens

Used in translation requests to limit the number of outstanding translation requests.

Invalidation tokens

Used in invalidation and synchronization messages to limit the number of outstanding invalidation requests.

Request messages consume tokens and response messages return them. If a response message is received over multiple cycles, then the token is only returned when the complete message has been received.

IDs are used to track some outstanding messages. A new request message cannot reuse an ID until a response message with that ID is received. If a response message is received over multiple cycles, then the ID can only be reused when the complete message has been received.

2.1.4 Reserved fields

Reserved fields in messages are described as either *Should Be Zero* (SBZ) or *Should be One* (SBO).

The recipient of a message with Reserved fields must ignore these fields. This specification recommends that the sender drive a Reserved field to 0 if it is described as SBZ, and 1 if it is described as SBO.

2.1.5 IMPLEMENTATION DEFINED fields

Some message fields are defined as being IMPLEMENTATION DEFINED. These fields can be used by implementations for any defined purpose.

These fields are treated as Reserved by components that do not require them.

2.2 Managing DTI connections

This section contains the following subsections:

2.2.1 Channel states

The four possible states of a DTI channel are:

DISCONNECTED

The TBU or PCIe RP might be powered down. A TCU must always be able to accept a Connect Request whenever a TBU or PCIe RP is powered up and able to send one. The method that is used to meet this requirement is outside the scope of this Specification.

REQ_CONNECT

The TBU or PCIe RP has issued a Connect Request. The TCU must provide a handshaking response to either establish or reject the connection.

CONNECTED

The channel is connected.

REQ_DISCONNECT

The TBU or PCIe RP has issued a Disconnect Request. The TCU issues a Disconnect Accept in response.

2.2.2 Handshaking

On power up, the channel is initially in the DISCONNECTED state. Figure 2-1 on page 2-29 shows how the channel state changes in response to connect and disconnect messages.

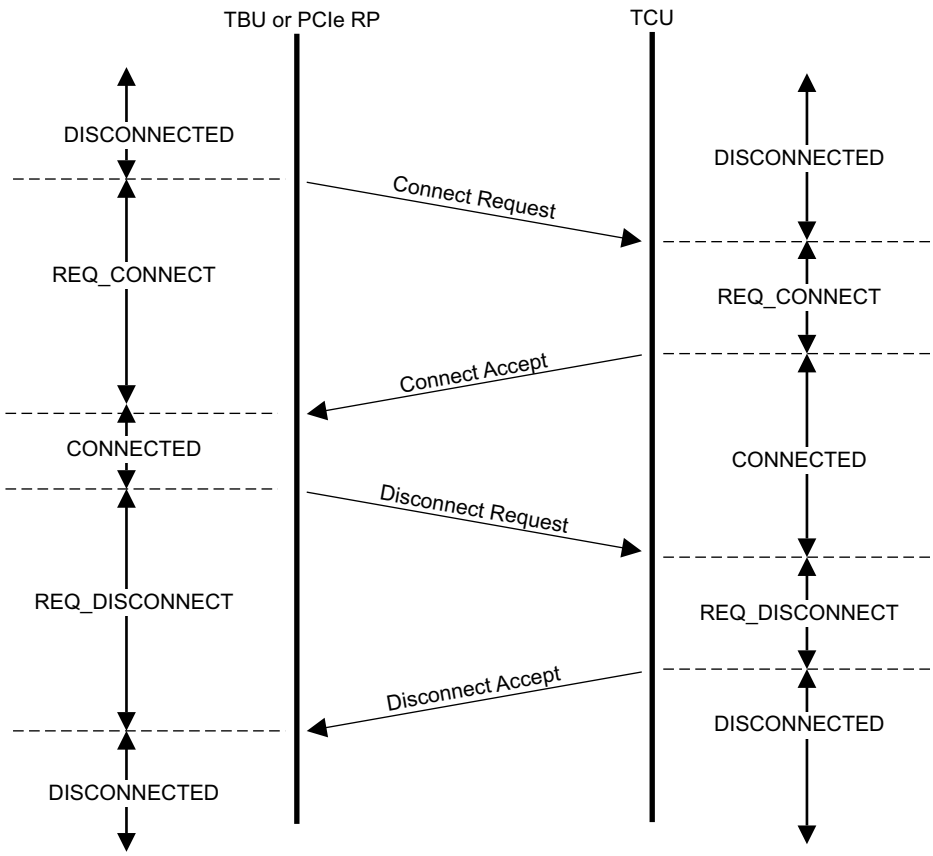


Figure 2-1 Handshake accept

Alternatively, a Connect Request might be denied, as shown in Figure 2-2 on page 2-30.

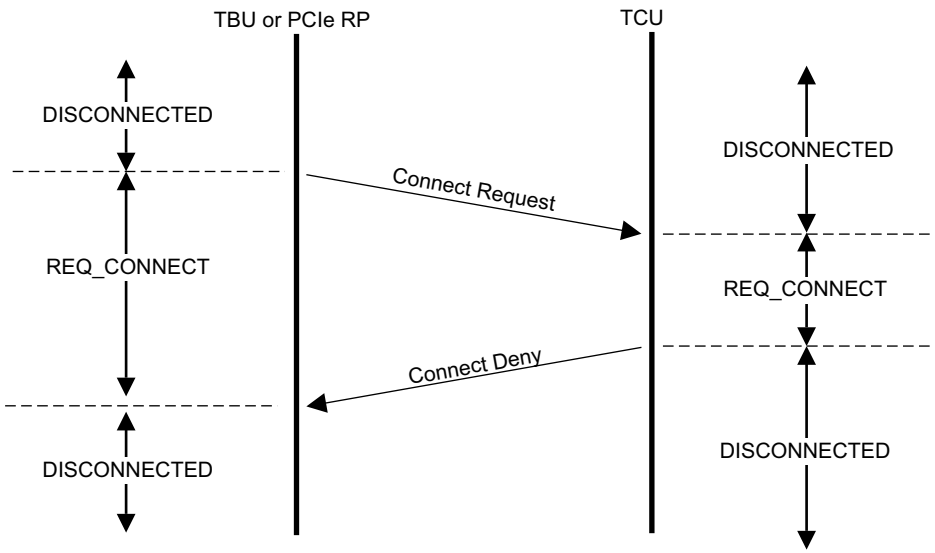


Figure 2-2 Handshake deny

A Connect Deny indicates a system failure, for example, due to a badly configured system. Subsequent attempts to connect are also likely to be denied until there is a system configuration change.

The following table describes the connection or disconnection messages that are permitted in each channel state.

Table 2-6 Connection or disconnection messages permitted in each channel state

Channel state	Downstream permitted messages	Upstream permitted messages
DISCONNECTED	Connect Request only	None
REQ_CONNECT	None	Connect Accept or Connect Deny
CONNECTED	Any, subject to the protocol rules	Any, subject to the protocol rules
REQ_DISCONNECT	None	Any, subject to the protocol rules

Channel behavior in the REQ_DISCONNECT state

When the channel is in the REQ_DISCONNECT state:

- Any outstanding invalidation or synchronization responses are not returned. All invalidation requests are considered to be completed when the TBU or PCIe RP enters DISCONNECTED state and invalidates its caches.
- Outstanding register access responses, DTI_TBU_REG_RDATA or DTI_TBU_REG_WACK, are not returned.
- Outstanding DTI_ATS_PAGE_RESPACK messages are not returned.
- The TBU or PCIe RP must continue to accept protocol appropriate requests from the TCU. No response is given to the requests and they can be ignored.

2.2.3 Initialization and disconnection

When the TBU enters the DISCONNECTED state, all state information is lost, including cache and register contents. The TBU must invalidate its caches before entering CONNECTED state. The TCU must reinitialize any necessary register contents after the connection handshake.

The DTI channel must not be disconnected while ATS is enabled in any PCIe Endpoint. DTI-ATS has no register messages.

2.2.4 Connecting multiple TBUs or PCIe RPs to a TCU

A DTI channel is a point-to-point link between a single TBU or PCIe RP and a single TCU. If a TCU is connected to multiple physical TBUs or PCIe RPs using a single interface, then each has its own DTI channel.

Therefore:

- If a TCU is required to send a message to multiple TBUs or PCIe RPs, then it must issue multiple messages.
- Each channel has its own flow control tokens.
- Outstanding message IDs, for example DTI_TBU_TRANS_REQ.TRANSLATION_ID, are specific to a channel. Multiple channels can have messages outstanding with the same ID at the same time.
- A DTI channel has a single connection state. It cannot be connected as both DTI-TBU and DTI-ATS at the same time.

Chapter 3
DTI-TBU Messages

This chapter describes the message groups of the DTI-TBU protocol.

It contains the following sections::

3.1 Connection and disconnection message group

The DTI-TBU protocol is designed to enable a single TCU to connect to multiple TBUs implementing different versions of the DTI-TBU. However, SMMUv3.2 requires support for some features that are not supported by DTI-TBUv1, and the SMMU architecture does not permit some TBUs to support features that other TBUs do not. Therefore, all TBUs connected to a TCU that implements SMMUv3.2 must support DTI-TBUv2.

This section contains the following subsections:

3.1.1 DTI_TBU_CONDIS_REQ

The DTI_TBU_CONDIS_REQ message is used to initiate a connection or disconnection handshake.

Description

Connection state change request

Source

TBU

Usage constraints

The TBU can only send a disconnect request when:

- The channel is in the CONNECTED state.
- There are no outstanding translation requests.
- The conditions for completing any future invalidation and synchronization are met. In practice, the result is that all downstream transactions must be complete.

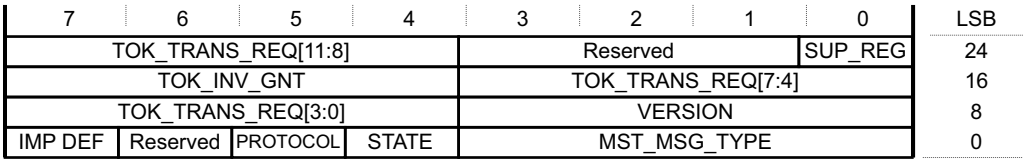
The TBU can only send a connect request when the channel is in the DISCONNECTED state.

Flow control result

None

Field descriptions

The DTI_TBU_CONDIS_REQ bit assignments are:



7	6	5	4	3	2	1	0	LSB
TOK_TRANS_REQ[11:8]				Reserved			SUP_REG	24
TOK_INV_GNT				TOK_TRANS_REQ[7:4]				16
TOK_TRANS_REQ[3:0]				VERSION				8
IMP DEF	Reserved	PROTOCOL	STATE	MST_MSG_TYPE				0

TOK_TRANS_REQ[7:4], bits [31:28]

TOK_TRANS_REQ[7:0] is bits [19:12].

The size of this field is dependent on the version of the DTI protocol being used.

DTI-TBUv1

TOK_TRANS_REQ[7:0] is bits [19:12].

Bits [31:28] are Reserved, SBZ.

DTI-TBUv2

TOK_TRANS_REQ[7:0] is bits [19:12].

TOK_TRANS_REQ[11:8] is bits [31:28].

The meaning of this field depends on the value of the STATE field.

STATE = 0

This field indicates the number of translation tokens returned.

The number of translation tokens returned is equal to the value of this field plus one.

This field must be the value of TOK_TRANS_GNT that was received in the DTI_TBU_CONDIS_ACK message that acknowledged the connection of the channel.

STATE = 1

This field indicates the number of translation tokens requested.

The number of translation tokens requested is equal to the value of this field plus one.

Bits [27:25]

Reserved, SBZ.

SUP_REG, bits [24]

This field indicates when register accesses are supported.

0 Register accesses are not supported.

1 Register accesses are supported.

When STATE is 1 and the value of this bit is 0, the TCU must not issue DTI_TBU register access messages on this channel.

When STATE is 0, this field is ignored.

TOK_INV_GNT, bits [23:20]

This field indicates the number of invalidation tokens granted.

The number of invalidation tokens granted is equal to the value of this field plus one.

This field is ignored when the STATE field has a value of 0.

TOK_TRANS_REQ[7:0], bits [19:12]

See TOK_TRANS_REQ[7:4], bits [31:28],

VERSION, bits [11:8]

This field identifies the requested protocol version.

`0b0000` DTI-TBUv1

`0b0001` DTI-TBUv2

All other encodings are Reserved.

A TBU can request any protocol version it supports. A DTI-TBU TCU must process requests for all protocol versions, including those not yet defined. The DTI_TBU_CONDIS_ACK message indicates the protocol version to use.

IMPLEMENTATION DEFINED, bit [7]

IMPLEMENTATION DEFINED.

Bit [6]

Reserved, SBZ.

PROTOCOL, bit [5]

This bit identifies the protocol that is used by this TBU.

0 DTI-TBU

This bit must be 0.

STATE, bit [4]

This bit identifies the new channel state requested.

0 Disconnect request

1 Connect request

A Disconnect request can only be issued when the channel is in the CONNECTED state.

A Connect request can only be issued when the channel is in the DISCONNECTED state.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for downstream messages, see *DTI-TBU protocol downstream messages* on page 2-25.

0000 DTI_TBU_CONDIS_REQ

3.1.2 DTI_TBU_CONDIS_ACK

The DTI_TBU_CONDIS_ACK message is used to accept or deny a request as part of the connection or disconnection handshake process.

Description

A connection state change acknowledgement

Source

TCU

Usage constraints

The TBU must have previously issued an unacknowledged DTI_TBU_CONDIS_REQ message.

Flow control result

None.

Field descriptions

The DTI_TBU_CONDIS_ACK bit assignments are:

7	6	5	4	3	2	1	0	LSB
TOK_TRANS_GNT[11:8]				Reserved			OAS[3]	24
OAS[2:0]			Reserved	TOK_TRANS_GNT[7:4]				16
TOK_TRANS_GNT[3:0]				VERSION				8
IMP DEF	Reserved		STATE	SLV_MSG_TYPE				0

TOK_TRANS_GNT[11:8], bits [31:28]

TOK_TRANS_GNT[7:0] is bits [19:12].

The size of this field is dependent on the version of the DTI protocol being used.

DTI-TBUv1

TOK_TRANS_REQ[7:0] is bits [19:12].

Bits [31:28] are Reserved, SBZ.

DTI-TBUv2

TOK_TRANS_REQ[7:0] is bits [19:12].

TOK_TRANS_REQ[11:8] is bits [31:28].

This field indicates the number of pre-allocated tokens for translation requests that have been granted.

The number of translation tokens granted is equal to the value of this field plus one.

The value of this field must not be greater than the value of the TOK_TRANS_REQ field in the DTI_TBU_CONDIS_REQ message.

When the value of STATE is 0, this field is ignored.

Bits [27:25]

Reserved, SBZ.

OAS, bits [24:21]

This indicates the output address size, which is the maximum address size permitted for translated addresses.

`0b0000` 32 bits (4GB)

`0b0001` 36 bits (64GB)

`0b0010` 40 bits (1TB)

`0b0011` 42 bits (4TB)

`0b0100` 44 bits (16TB)

`0b0101` 48 bits (256TB)

`0b0110` 52 bits (4PB)

All other values are Reserved.

Bit [20]

Reserved, SBZ.

TOK_TRANS_GNT[7:0], bits [19:12]

See TOK_TRANS_GNT[7:4], bits [31:28]

VERSION, bits [11:8]

The protocol version that is granted by the TCU.

`0b0000` DTI-TBUv1

`0b0001` DTI-TBUv2

The value of this field must not be greater than the value of the VERSION field in the DTI_TBU_CONDIS_REQ Connect Request message.

IMPLEMENTATION DEFINED, bit [7]

IMPLEMENTATION DEFINED.

Bits [6:5]

Reserved, SBZ.

STATE, bit [4]

Identifies the new state. The possible values of this bit are:

0 DISCONNECTED

1 CONNECTED

When the value of STATE in the unacknowledged DTI_TBU_CONDIS_REQ message is 0, the value of this bit must be 0.

When the value of STATE in the unacknowledged DTI_TBU_CONDIS_ REQ message is 1, this field can be 0 or 1.

For example, it can be 0 if there are no translation tokens available. This normally indicates a serious system configuration failure.

SLV_MSG_TYPE, bits [3:0]

Identifies the message type. The value of this field is taken from the list of encodings for upstream messages, see *DTI-TBU protocol upstream messages* on page 2-25.

`0b0000` DTI_TBU_CONDIS_ACK

3.2 Translation request message group

The DTI-TBU translation request messages enable the TBU to find the translation for a given transaction, or prefetch a translation. The TCU responds with either a successful translation or a fault.

This section contains the following subsections:

3.2.1 DTI_TBU_TRANS_REQ

The DTI_TBU_TRANS_REQ message is used to initiate a translation request.

Description

A translation request

Source

TBU

Usage constraints

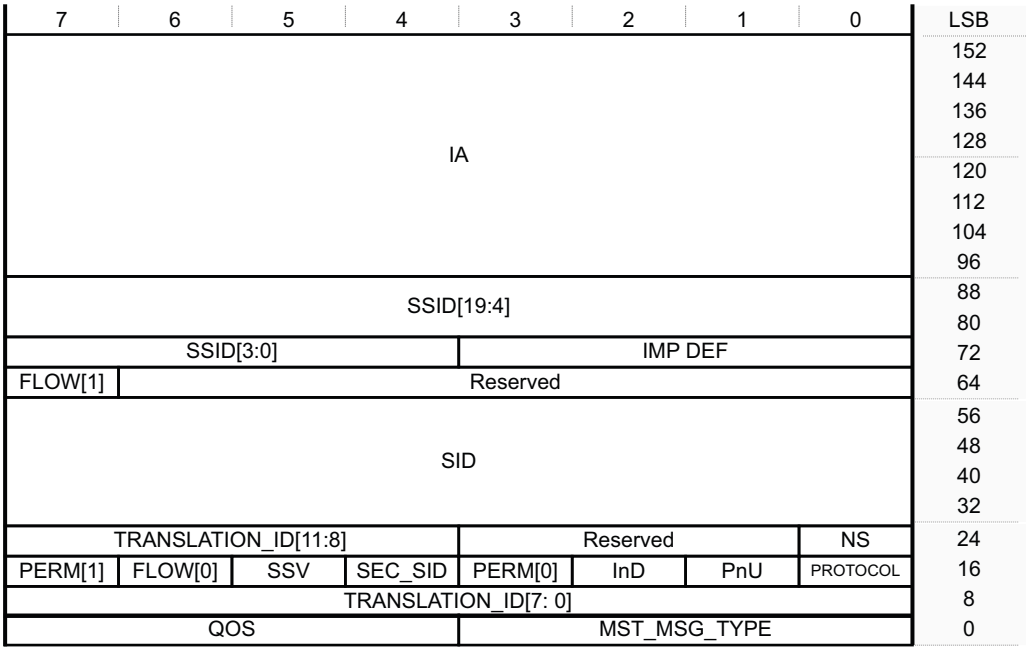
The TBU must have at least one translation token.

Flow control result

The TBU consumes a translation token.

Field descriptions

The DTI_TBU_TRANS_REQ bit assignments are:



IA, bits [159:96]

This field holds the input address, IA[63:0], to be used in the translation.

SSID, bits [95:76]

This field indicates the SubstreamID value that is used for the translation.

When the value of SSV is 0, this field is Reserved, SBZ.

IMPLEMENTATION DEFINED, bit [75:72]

IMPLEMENTATION DEFINED.

FLOW[1], bit [71]

FLOW[0] is bit [22].

This field indicates the translation flow required.

0b00 Stall

If enabled, the SMMU stall fault flow can be used for this request.

A translation request can only be stalled by the TCU if FLOW=Stall.

Selecting FLOW=Stall does not cause a stall to occur. A stall only occurs if software enables stall faulting for the translation context.

0b01 ATST

The transaction has been translated by ATS.

When this field has a value of 1, it indicates that this transaction was the result of a previous ATS translation request made using DTI-ATS.

DTI-DTIv2

- PnU field must be 0
- InD field must be 0

0b10 NoStall

DTI-TBUv1

Reserved.

DTI-TBUv2

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 upon software activity.

0b11 PRI

DTI-TBUv1

Reserved.

DTI-TBUv2

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.

PRI requests must be sent using a DTI-ATS connection. There is no mechanism to issue a PRI requests from a DTI-TBU connection.

Bits [70:64]

Reserved, SBZ.

SID, bits [63:32]

This field indicates the StreamID value that is used for the translation.

TRANSLATION_ID[11:8], bits [31:28]

TRANSLATION_ID[7:0] is bits [15:8].

This field gives the identification number of this translation.

The value of this field must not be in use by any translation request that has not yet received a DTI_TBU_TRANS_RESP or DTI_TBU_TRANS_FAULT response.

DTI-TBUv1

Any 8-bit translation ID in TRANSLATION_ID[7:0], bits [15:8], can be used if the maximum number of outstanding translation requests is not exceeded.

TRANSLATION_ID[11:8] is Reserved, SBZ.

DTI-TBUv2

Any 12-bit translation ID can be used, if the maximum number of outstanding translation requests is not exceeded.

Bits [27:25]

Reserved, SBZ.

NS, bit [24]

This bit indicates the security level of the transaction.

0 Secure

1 Non-secure

Must be 1 if SEC_SID = 0.

PERM[1], bit [23]

PERM[1] and PERM[0] indicate permissions a translation request requires to avoid causing a permission fault.

The encoding of PERM[1:0] is:

0b00 W: Write permission required.

0b01 R: Read permission required.

0b11 SPEC: Neither permission required. The translation request is speculative and cannot cause a permission fault.

0b10 RW: Read and write permission required.

Note

Between Edition 1 and Edition 2, the SPECULATIVE and RnW fields have been combined to create the PERM field. The only behavioral change is that the combined field supports a new RW encoding.

FLOW[0], bit [22]

See FLOW[1], bit [71].

SSV, bit [21]

This bit indicates whether a valid SSID field is associated with this translation.

0 The SSID field is not valid.

1 The SSID field is valid.

When the value of FLOW is ATST, this bit must be 0.

SEC_SID, bit [20]

This bit indicates whether the StreamID is Secure.

0 Non-secure StreamID

1 Secure StreamID

When the value of FLOW is ATST, this bit must be 0.

PERM[0], bit [19]

See PERM[1], bit [23].

InD, bit [18]

This bit indicates whether the transaction is an instruction access or data access.

0 Data access

1 Instruction access

When the value of PERM[1:0] is W, RW or SPEC, this bit must be 0.

DTI-TBUv2

When FLOW is ATST, this bit must be 0.

PnU, bit [17]

This bit indicates whether this transaction represents privileged or unprivileged access.

0 Unprivileged

1 Privileged

When the value of PERM[1:0] is SPEC, this bit must be 0.

DTI-TBUv2

When FLOW is ATST, this bit must be 0.

PROTOCOL, bit [16]

This bit indicates the protocol that is used for this message.

0 DTI-TBU

This bit must be 0.

TRANSLATION_ID[7:0], bits [15:8]

See TRANSLATION_ID[15:8], bits [31:28].

QOS, bits [7:4]

This field indicates the Quality of Service priority level.

Translation requests with a high QOS value are likely to be responded to before requests with a lower QOS value.

This field is a hint.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for downstream messages, see *DTI-TBU protocol downstream messages* on page 2-25

`0b0010` DTI_TBU_TRANS_REQ

3.2.2 DTI_TBU_TRANS_RESP

The DTI_TBU_TRANS_RESP message is used to respond to a successful translation request.

The TCU can only return this message when permission is granted for the transaction that is described in the translation request. If permission is not granted, a DTI_TBU_TRANS_FAULT response must be issued. For more information, see *Faulting expressions of the translation request message* on page 3-56.

Description

A DTI translation response

Source

TCU

Usage constraints

The TBU must have previously issued a translation request that has not yet generated either a translation response or a fault message.

Flow control result

The TCU returns a translation token to the TBU.

Field descriptions

The DTI_TBU_TRANS_RESP bit assignments are:

7	6	5	4	3	2	1	0	LSB
IMP DEF				CTXTATTR or PARTID[3:0]				152
PARTID[7:4]				OA[51:48]				144
OA[47:16]								136 128 120 112
OA[15:12]				PARTID[8]	PMG	SH		104
ATTR								96
S2HWATTR or HWATTR				S1HWATTR or Reserved				88
INVAL_RNG				TRANS_RNG				80
TRANSLATION_ID[11:8]				COMB_ALLOC	COMB_SH	MPAMNS	GLOBAL	72
TBI	NS	ALLOW_PX or ALLOW_NSX	ALLOW_PW	ALLOW_PR	ALLOW_UX	ALLOW_UW	ALLOW_UR	64
ASID or ATTR_OVR								56 48
VMID								40 32
ALLOCCFG				COMB	ASETor NSOVR	INSTCFG		24
PRIVCFG	DCP	DRE		STRW or BP_TYPE		BYPASS	CONT[3]	16
CONT[2:0]			DO_NOT_CACHE	TRANSLATION_ID[7:4]				8
TRANSLATION_ID[3:0]				SLV_MSG_TYPE				0

IMPLEMENTATION DEFINED, bits [159:156]

IMPLEMENTATION DEFINED.

CTXTATTR/PARTID[3:0], bits [155:152]

DTI-TBUv1

IMPLEMENTATION DEFINED attributes for the translation context.

DTI-TBUv2

MPAM PARTID[3:0]

PARTID[7:4], bits [151:148]

DTI-TBUv1

Reserved, SBZ.

DTI-TBUv2

MPAM PARTID[7:4].

OA, bits [147:108]

This field holds the output address, OA[51:12], of the translated address.

DTI-TBUv1

This address must be the first byte in a region of size that is given by the TRANS_RNG field. For example, if the value of TRANS_RNG is 2, then OA[15:12] must be zero.

When the value of BYPASS is 1, this field is Reserved, SBZ.

DTI-TBUv2

Bits within the range given by TRANS_RNG must match DTI_TBU_TRANS_REQ.IA.

For example, if the value of TRANS_RNG is 2, then OA[15:12] must match DTI_TBU_TRANS_REQ.IA[15:12].

When the value of BYPASS is 1, this field must equal the value of IA in the translation request.

The address in this field must be within the range indicated by the OAS field of the DTI_TBU_CONDIS_ACK message received during the connection sequence.

PARTID[8], bit [107]

DTI-TBUv1

Reserved, SBZ.

DTI-TBUv2

MPAM PARTID[8]

PMG, bit [106]

DTI-TBUv1

Reserved, SBZ.

DTI-TBUv2

MPAM PMG

SH, bits [105:104]

This field indicates the shareability of the translation.

`0b00` Non-shareable

`0b01` Reserved

`0b10` Outer-shareable

`0b11` Inner-shareable

Note

This value represents the Shareability attribute that is stored in the page tables. In some cases the resulting Shareability of the translation might be different from the value that is shown here. For more information, see *Consistency check on combination of translation attributes* on page 3-63.

When the value of BYPASS is 1, this field is Reserved, SBZ.

ATTR, bits [103:96]

This field indicates the translation attributes.

Bits [103:100] are encoded as:

`0b0000` Device memory. See encoding of bits [99:96] for the device memory type.

`0b00RW` When RW is not 00, this field is Normal Memory, Outer Write-through transient.

`0b0100` Normal Memory, Outer Non-Cacheable.

`0b01RW` When RW is not 00 this field is Normal Memory, Outer Write-back transient.

`0b10RW` Normal Memory, Outer Write-through non-transient.

`0b11RW` Normal Memory, Outer Write-back non-transient.

Where R is the Outer Read Allocate Policy and W is the Outer Write Allocate Policy.

The meaning of bits [99:96] depends on the value of bits [103:100]:

Table 3-1 ATTR encoding bits [103:100]

Bits [99:96]	When [103:100] is 0b0000	When [103:100] is not 0b0000
`0b0000`	Device-nGnRnE memory	Reserved
`0b00RW`, RW is not `0b00`	Reserved	Normal Memory, Inner Write-through transient
`0b0100`	Device-nGnRE memory	Normal memory, Inner Non-cacheable
`0b01RW`, RW is not `0b00`	Reserved	Normal Memory, Inner Write-back transient
`0b1000`	Device-nGRE memory	Normal Memory, Inner Write-through non-transient (RW=00)

Table 3-1 ATTR encoding bits [103:100] (continued)

Bits [99:96]	When [103:100] is 0b0000	When [103:100] is not 0b0000
0b10RW, RW is not 0b00	Reserved	Normal Memory, Inner Write-through non-transient
0b1100	Device-GRE memory	Normal Memory, Inner Write-back non-transient (RW=00)
0b11RW, RW is not 0b00	Reserved	Normal Memory, Inner Write-back non-transient

Where R is the Inner Read Allocate Policy and W is the Inner Write Allocate Policy.

The R and W bits have the following encoding:

0 Do not allocate

1 Allocate

When the value of BYPASS is 1, this field is Reserved, SBZ.

S2HWATTR/HWATTR, bits [95:92]

This field gives IMPLEMENTATION DEFINED hardware attributes from the page tables. These are otherwise known as Page-Based Hardware Attributes (PBHA).

Bits that are not enabled for use by hardware must be 0.

If a TCU does not support this feature, it can return 0 for this field.

DTI-TBUv1

This field S2HWATTR gives the IMPLEMENTATION DEFINED stage 2 hardware attributes.

The value of this field must be 0 if either of the following conditions are true:

- The value of BYPASS is 1.
- The value of BYPASS is 0 and at least one of the following is true:
  - The value of SEC_SID is 1.
  - The value of STRW is either EL2 or EL3.

DTI-TBUv2

HWATTR gives the IMPLEMENTATION DEFINED combined stage 1 and stage 2 hardware attributes.

S1HWATTR, bits [91:88]

DTI_DTI-TBUv1

This field gives the IMPLEMENTATION DEFINED stage 1 hardware attributes.

These attributes are provided in the stage 1 page tables for IMPLEMENTATION DEFINED purposes.

Bits that are not enabled for use by hardware use must be 0.

If a TCU does not support this feature, it can return 0 for this field.

The value of this field must be 0 if either of the following conditions are true:

- The value of BYPASS is 1.
- The value of BYPASS is 0 and value of STRW is EL1-S2.

DTI-TBUv2

Reserved, SBZ.

INVAL_RNG, bits [87:84]

This field indicates the range of addresses for invalidation.

`0b0000` 4KB

`0b0001` 16KB

`0b0010` 64KB

`0b0011` 2MB

`0b0100` 32MB

`0b0101` 512MB

`0b0110` 1GB

`0b1000` 4TB

All other values are Reserved.

The value of this field might be different from the value of the TRANS_RNG field in either of the following cases:

- When two stage translation is used and the range of the stage 1 translation is larger than the range of the stage 2 translation range. In this case, this field represents the stage 1 translation range and TRANS_RNG represents the stage 2 translation range.
- When the CONT bit is set in a page table entry. The CONT bit increases the address range of the translation but is not required to affect the address range that is used by invalidations.

If an invalidation request is received, this translation must be invalidated when both of the following conditions exist:

- The properties of this transaction match the invalidation request properties.
- The address to be invalidated falls inside the range that is specified by this field.

When the value of the BYPASS field is 1, this field is Reserved, SBZ.

DTI-TBUv1

The range given by this field must not be greater than the size indicated by the OAS field of the DTI_TBU_CONDIS_ACK message.

For example, if the OAS is 4GB, this field must indicate a range of 1GB or less.

This field must not indicate a size of 4TB unless the OAS field of the DTI_TBU_CONDIS_ACK message received during the connection sequence indicates a size of 52 bits.

DTI_DTI-TBUv2

This field is not restricted by the size indicated by the OAS field of the DTI_TBU_CONDIS_ACK message received during the connection sequence.

TRANS_RNG, bits [83:80]

The meaning of this field depends on the value of the BYPASS field.

BYPASS==0

This field indicates the aligned range of addresses that this translation is valid for:

`0b0000` 4 KB

`0b0001` 16 KB

`0b0010` 64 KB

`0b0011` 2 MB

`0b0100` 32 MB

`0b0101` 512 MB

`0b0110` 1GB

`0b0111` 16 GB

`0b1000` 4 TB

All other values are Reserved.

Previous editions of this specification listed an encoding for 128 TB. This encoding option was removed, since is not supported in the Arm architecture.

DTI-TBUv1

This field must not be greater than the size indicated by the OAS field of the DTI_TBU_CONDIS_ACK message received during the connection sequence.

For example, if the value of the OAS field is 4GB, this field must indicate a range of 1GB or less.

This field must not indicate a size of 4TB, unless the OAS field of the DTI_TBU_CONDIS_ACK message received during the connection sequence indicates a size of 52 bits.

DTI-TBUv2

This field is not restricted by the size indicated by the OAS field of the DTI_TBU_CONDIS_ACK message received during the connection sequence.

BYPASS==1

This field indicates the maximum output address size of the system:

`0b0000` 32 bits (4GB)

`0b0001` 36 bits (64GB)

`0b0010` 40 bits (1TB)

`0b0011` 42 bits (4TB)

`0b0100` 44 bits (16TB)

`0b0101` 48 bits (256TB)

`0b0110` 52 bits (4PB)

All other values are Reserved.

This information is also given in the OAS field of the DTI_TBU_CONDIS_ACK message, and uses the same encodings. When BYPASS=1, this field must match DTI_TBU_CONDIS_ACK.OAS.

If the TBU encounters a transaction with an IA outside of the range indicated in this field, then it cannot be translated with this translation. In this case, a new translation request must be made, so that software can be notified about the fault, if necessary.

The maximum output address size is a static property of the system. If this is not the first DTI_TBU_TRANS_RESP message when BYPASS is 1 since the link was connected, TRANS_RNG must have the same value as the previous DTI_TBU_TRANS_RESP message where BYPASS is 1.

This field must show a range large enough to contain the IA of the transaction. For example, if DTI_TBU_TRANS_REQ.IA=0x0000_0001_0000_0000 or greater, this field cannot show a range of 32 bits (4GB).

COMB_ALLOC, bit [75]

DTI-TBUv1

Reserved, SBZ.

DTI-TBUv2

This field indicates how the translation allocation hints should be handled:

0 The allocation hints in the ATTR field override the transaction attributes.

1 The allocation hints in the ATTR field are combined with the transaction attributes.

When BYPASS is 0 and STRW is EL1_S2, COMB_ALLOC must be 1

When BYPASS is 1, COMB_ALLOC is Reserved, SBZ.

For more information, see *Calculating transaction attributes* on page 3-56.

COMB_SH, bit [74]

DTI-TBUv1

Reserved, SBZ.

DTI-TBUv2

This field indicates how the translation Shareability should be handled:

0 The Shareability in the SH field overrides the transaction attributes.

1 The Shareability in the SH field is combined with the transaction attributes.

When BYPASS is 0 and STRW is EL1, EL2 or EL3, COMB_SH must be 0.

When BYPASS is 0 and STRW is EL1_S2, COMB_SH must be 1.

When BYPASS is 1, COMB_SH is Reserved, SBZ.

For more information, see *Calculating transaction attributes* on page 3-56.

MPAMNS, bit [73]

DTI-TBUv1

Reserved, SBZ.

DTI-TBUv2

MPAM MPAMNS value.

If DTI_TBU_TRANS_REQ.SEC_SID=0, MPAMNS must be 1.

GLOBAL, bit [72]

This bit indicates that this result is valid for any ASID.

0 Non-global

1 Global

This bit might be 1 for either of the following reasons:

- The stage 1 page table global attribute is set.
- Stage 1 translation is disabled or not supported.

When the value of STRW is EL3, this bit must be 1.

When the value of BYPASS is 1, this bit is Reserved, SBZ.

TBI, bit [71]

This bit indicates whether this translation applies to future transactions where the top byte of the input address is different.

0 Subsequent transactions can only use this translation if IA[63:56] matches.

1 Subsequent transactions can use this translation regardless of the value of IA[63:56].

When the value of BYPASS is 1, this bit is Reserved, SBZ.

NS, bit [70]

This bit indicates the security status to be used for downstream transactions.

0 Secure

1 Non-secure

When the value of FLOW is ATST in the translation request, this bit must be 1.

When the value of SEC_SID in the translation request is 0, this bit must be 1.

DTI-TBUv1

When the value of BYPASS is 1 and the value of NSOVR is 0, this bit is Reserved, SBO. In this case, the downstream security status matches the upstream security status.

DTI-TBUv2

NS always contains the NS bit of the translated transaction.

When DTI_TBU_TRANS_REQ.SEC_SID = 1 and BYPASS = 1:

- When NSCFG = Use incoming, NS must equal DTI_TBU_TRANS_REQ.NS.
- When NSCFG = Secure, NS must be 0.
- When NSCFG = Non-secure, NS must be 1.

ALLOW_PX, bit [69] when BYPASS=0

This bit indicates permissions for privileged instruction reads.

0 Not permitted

1 Permitted

ALLOW_NSX, bit [69] when BYPASS=1

This bit indicates permissions for Non-secure instruction reads.

0 Not permitted

1 Permitted

Data accesses and Secure instruction reads are always permitted when the value of BYPASS is 1.

This bit is related to the Secure Instruction Fetch (SIF) setting in the SMMU.

When the value of SEC_SID in the translation request message is 0, this field is Reserved, SBZ.

ALLOW_PW, bit [68]

This bit indicates permissions for privileged data write accesses.

0 Not permitted

1 Permitted

When BYPASS is 1, this field is Reserved, SBZ.

ALLOW_PR, bit [67]

This bit indicates permissions for privileged data read accesses.

0 Not permitted

1 Permitted

When BYPASS is 1, this field is Reserved, SBZ.

ALLOW_UX, bit [66]

This bit indicates permissions for unprivileged instruction reads.

0 Not permitted

1 Permitted

When the value of STRW is EL3, this bit must be equal to the value of ALLOW_PX.

When BYPASS is 1, this field is Reserved, SBZ.

ALLOW_UW, bit [65]

This bit indicates permissions for unprivileged data write accesses.

0 Not permitted

1 Permitted

When the value of STRW is EL3, this bit must be equal to the value of ALLOW_PW.

When BYPASS is 1, this field is Reserved, SBZ.

ALLOW_UR, bit [64]

This bit indicates permissions for unprivileged data read accesses.

0 Not permitted

1 Permitted

When the value of STRW is EL3, this bit must be equal to the value of ALLOW_PR.

When BYPASS is 1, this field is Reserved, SBZ.

ASID/ATTR_OVR, bits [63:48]

This field is ASID when the value of BYPASS is 0, and the value of STRW is not EL1-S2.

Note

When the ASID field is valid, stage 1 translation is enabled, which overrides the incoming attributes. Therefore the ATTR_OVR field is unnecessary when the ASID field is valid.

This field is ATTR_OVR when either of the following conditions are met:

- The value of BYPASS is 1.
- The value of BYPASS is 0, and the value of STRW is EL1-S2.

ASID

This field holds the ASID to be used for stage 1 translation.

When the value of STRW is EL3, this field must be 0.

ATTR_OVR

This field is used to override the incoming attributes.

When the value of FLOW is ATST in the DTI_TBU_TRANS_REQ message, this field must be `0x0020`. The effect of this encoding is to cause the incoming attributes to be used, as stage 1 translation has already been performed.

This field might be combined with the ATTR and SH field to give different values for the attributes of this translation. For more information about this and the subfields of this field, see *Calculating transaction attributes* on page 3-56

When the value of MTCFG is 0, the MemAttr component of this field is ignored.

VMID, bits [47:32]

This field indicates the VMID value that is used for the translation.

DTI-TBUv1

This field must be 0 when BYPASS is 0 the value of SEC_SID in the translation request is 1.

When BYPASS is 0 and the value of STRW is either EL2 or EL3, this field must be 0.

When BYPASS is 1, this field is Reserved, SBZ.

ALLOCCFG, bits [31:28]

This field indicates the override for the allocation hints of incoming transactions.

For the encoding and the effects of this field *Calculating transaction attributes* on page 3-56.

COMB_MT, bit [27]

DTI-TBUv1

Reserved, SBZ.

DTI-TBUv2

This field indicates how the translation memory type and Cacheability should be handled:

0 The memory type and Cacheability in the ATTR field override the transaction attributes.

1 The memory type and Cacheability in the ATTR field are combined with the transaction attributes.

When BYPASS is 1, COMB_MT is Reserved, SBZ.

When BYPASS is 0 and STRW is EL1, EL2 or EL3, COMB_MT must be 0.

For more information, see *Calculating transaction attributes* on page 3-56.

ASET, bit [26] when BYPASS = 0

This bit indicates the shareability of the ASID set.

0 Shared set

1 Non-shared set

Note

This field is still valid when the ASID value is not valid.

NSOVR, bit [26] when BYPASS = 1

This bit indicates the Non-secure bit override.

0 Use the upstream NS value.

1 Override using the value of the NS bit in this message.

When the value of SEC_SID is 0, this value of this field must be 1.

DTI-TBUv2

The NSOVR bit can be ignored by DTI-TBUv2 devices, since its value can be derived from other fields. The following rules are added for DTI-TBUv2 when DTI_TBU_TRANS_REQ.SEC_SID = 1 and BYPASS = 1:

- When NSCFG = Use incoming, NSOVR must be 0.
- When NSCFG = Secure or Non-secure, NSOVR must be 1.

INSTCFG, bits [25:24]

This field is used to override the incoming InD values for the transaction.

`0b00` Use incoming

`0b01` Reserved

`0b10` Data

`0b11` Instruction

This field only applies to incoming reads. The overridden value is used for the permission check and downstream transaction.

PRIVCFG, bits [23:22]

This field is used to override the incoming PnU values for the transaction.

`0b00` Use incoming

`0b01` Reserved

`0b10` Unprivileged

`0b11` Privileged

The overridden value is used for the permission check and downstream transaction.

DCP, bit [21]

This bit indicates whether directed cache prefetch hints are permitted.

0 Not permitted

1 Permitted

A directed cache prefetch hint is an operation that changes the cache allocation in a part of the cache hierarchy that is not on the direct path to memory. For example, the AMBA 5 WriteUniquePtlStash, WriteUniqueFullStash, StashOnceShared, and StashOnceUnique transactions all perform a directed cache prefetch hint operation.

A directed cache prefetch without write data is permitted if the value of this bit is 1, and any of read, write or execute permissions are given by the appropriate fields in this message at the appropriate privilege level. A directed cache prefetch with write data is permitted if the value of this bit is 1, and write permission is given by the appropriate fields in this message at the appropriate privilege level.

If directed cache prefetch hints are not permitted, directed cache prefetch hints are stripped from the transaction being translated. A directed cache prefetch with write data is converted into an ordinary write, and a directed cache prefetch without write data is terminated with a response indicating successful completion of the transaction. There is no communication with the TCU to indicate that this conversion has occurred.

When the value of BYPASS is 1, this field is Reserved, SBZ, and directed cache prefetches are permitted.

DRE, bit [20]

This bit indicates whether destructive reads are permitted.

0 Not permitted

1 Permitted

A destructive read is permitted if the value of this bit is 1, and read and write permission is given by the appropriate fields in this message at the appropriate privilege level.

Note

As there is no concept of an instruction write, destructive instruction reads are never permitted.

If a destructive read is not permitted, and reads are permitted, then the read must be converted into a non-destructive read. For example, a MakeInvalid transaction must be converted into a CleanInvalid transaction and a ReadOnceMakeInvalid transaction must be converted into a ReadOnceCleanInvalid transaction. There is no communication with the TCU to indicate that this conversion has occurred.

When the value of BYPASS is 1, this field is Reserved, SBZ, and destructive reads are permitted.

STRW, bits [19:18] when BYPASS=0

These bits indicate the SMMU StreamWorld, which is the Exception level that is used by the translation context.

0b00 EL1

0b01 EL1-S2

0b10 EL2

0b11 EL3

The permitted encodings of this field depend on the values of the SEC_SID and FLOW fields in the translation request:

- When the value of SEC_SID is 0, this field is not permitted to be EL3
- When the value of SEC_SID is 1:

  DTI-TBUv1

  This field must be EL1 or EL3.

  DTI-TBUv2

  This field is permitted to be any value.
- When the value of FLOW is ATST, this field must be EL1-S2.
- When the value of SSV is 1, this field must not be EL1-S2.

BP_TYPE, bits [19:18] when BYPASS=1

These bits indicate the scope of this translation.

0b00 Reserved

0b01 GlobalBypass

0b10 StreamBypassNoSSV

0b11 Reserved

Table 3-2 shows the fields of the translation request that must match for this translation to apply to future transactions.

Table 3-2 Matching field values for future transactions

BP_TYPE	SEC_SID	FLOW == ATST	SID	SSV	SSID
GlobalBypass	Yes	Yes	No	No	No
StreamBypassNoSSV	Yes	Yes	Yes	Yes (always 0)	-

If SSV = 1 in the translation request. this field must not be StreamBypassNoSSV.

The GlobalBypass encoding might be used when either:

- A translation is requested when the value of SMMUEN in the SMMU is LOW for the corresponding security level.
- A translation is requested with FLOW set to ATST and with the ATSCHK bit of the SMMU set to clear.

BYPASS, bit [17]

This field indicates whether translation is bypassed.

0 Normal translation

1 Translation bypassed

When the value of this field is 1, the VA and the PA of the translation are the same.

This bit must be 0 if the value of IA in the translation request is greater than the range shown in the OAS field of the DTI_TBU_CONDIS_ACK message that was received during the connection sequence.

When DTI_TBU_TRANS_REQ.SEC_SID is 1 and BYPASS is 0:

CONT, bits [16:13]

This field indicates the number of contiguous StreamIDs that the result of this transaction applies to.

This field is encoded to give the span of the contiguous block as 2^{CONT} StreamIDs. The block must start at a StreamID for which the bits SID[CONT-1:0] are 0.

When this field is non-zero, SID[CONT-1:0] in the translation request can be ignored when determining whether this translation matches future transactions.

If the value of the BYPASS bit is 1 and the BP_TYPE is GlobalBypass, this field is Reserved, SBZ.

DO_NOT_CACHE, bit [12]

This bit indicates to the TBU when not to cache a translation.

0 The translation has not been invalidated before this message was sent.

1 The translation might have been invalidated before this message was sent. Any transactions using this translation must be completed before the next invalidation synchronization operation is completed.

Note

A TBU can use this field to simplify invalidation, by not caching any translations that have a value of 1 for this field.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for upstream messages, see *DTI-TBU protocol upstream messages* on page 2-25.

0b0010 DTI_TBU_TRANS_RESP

Determination of IPA space

In DTI-TBUv2, the TBU uses DTI_TBU_TRANS_REQ.NS and NSCFG to determine whether the translation is for a secure IPA or Non-secure IPA.

Table 3-3 shows the information required for future TLB lookups and invalidation operations and forms part of the TLB tag information.

Table 3-3 Determination of IPA space in DTI-TBUv2

DTI_TBU_TRANS_REQ.NS	NSCFG	IPA space
0	Use Incoming	Secure
1	Use Incoming	Non-secure
-	Secure	Secure
-	Non-secure	Non-secure

In DTI-TBUv2, TLB entries match based upon the value of NS after being overridden by TRANS_RESP.ATTR_OVR.NSCFG, if appropriate. For example, a TLB entry created from a translation with TRANS_RESP.BYPASS = 1 can match subsequent transactions with any value of NS, provided that TRANS_RESP.ATTR_OVR.NSCFG is applied to the incoming transaction NS value to determine the translated NS value.

3.2.3 DTI_TBU_TRANS_FAULT

The DTI_TBU_TRANS_FAULT message is used to provide a fault response to a translation request.

Description

A translation fault response.

Source

TCU

Usage constraints

The TBU must have previously issued a translation request that has not yet generated either a translation response or a fault message.

This message must be used in the case of a translation request that has failed a permission check.

Flow control result

The TCU returns a translation token to the TBU.

Field descriptions

The DTI_TBU_TRANS_FAULT bit assignments are:

7	6	5	4	3	2	1	0	LSB
TRANSLATION_ID[11:8]				Reserved				24
Reserved				FAULT_TYPE			CONT[3]	16
CONT[2:0]			DO_NOT_CACHE	TRANSLATION_ID[7,4]				8
TRANSLATION_ID[3:0]				SLV_MSG_TYPE				0

TRANSLATION_ID[11:8], bits [31:28]

TRANSLATION_ID[7:0] is bits [11:4].

This field gives the identification number for the translation.

This field must have a value corresponding to an outstanding translation request.

DTI-TBUv1

Bits [31:28] Reserved, SBZ.

Bits [27:20]

Reserved, SBZ.

FAULT_TYPE, bits [19:17]

This bit indicates to the TBU how to handle the fault.

0b000 NonAbort. The translation has failed and the transaction must be terminated, depending on the value of DTI_TBU_TRANS_REQ.PERM[1:0]:

R Return read data of 0.

RW Return read data of 0 and ignore write data.

W Ignore write data.

SPEC Notify the TBU that the speculative read was unsuccessful, for example by returning an abort.

0b001 Abort. The translation has failed and the transaction must be terminated with an abort.

FAULT_TYPE must not be Abort when DTI_TBU_TRANS_REQ.PERM[1:0]=SPEC.

0b010 StreamDisabled.

The translation has failed and the transaction must be terminated with an abort.

The TBU can abort subsequent transactions, if all the following are true::

- The value of DTI_TBU_TRANS_REQ.SEC_SID is the same for both transactions.
- The value of DTI_TBU_TRANS_REQ.SID is the same for both transactions, when masked with DTI_TBU_TRANS_FAULT.CONT.
- Either
  - DTI_TBU_TRANS_REQ.FLOW is ATST for both transactions.
  - DTI_TBU_TRANS_REQ.FLOW is not ATST for either transaction.
- DO_NOT_CACHE is not 1.

0b011 GlobalDisabled

The translation has failed and the transaction must be terminated with an abort.

The TBU can abort subsequent transactions, if all the following are true:

- The value of DTI_TBU_TRANS_REQ.SEC_SID is the same for both transactions,
- DTI_TBU_TRANS_REQ.FLOW is not ATST for either transaction.
- DO_NOT_CACHE is not 1.

FAULT_TYPE must not be GlobalDisabled when DTI_TBU_TRANS_REQ.FLOW=ATST.

0b100 TranslationPRI

DTI-TBUv1

Not legal, bit [19] SBZ.

DTI-TBUv2

This response is only permitted when DTI_TBU_TRANS_REQ.FLOW=PRI. A translation-related fault has occurred, which might be resolved by a PRI request.

0b101 TranslationStall

DTI-TBUv1

Not legal, bit [19] SBZ.

DTI-TBUv2

The purpose of this response is to simplify deadlock handling when a DTI_TBU_SYNC_REQ message is received.

This response is only permitted when DTI_TBU_TRANS_REQ.FLOW=Stall. A translation fault has occurred, which has resulted in the transaction being stalled.

This does not complete the translation. The translation token is not returned and the translation request is still outstanding.

A TranslationStall response must not occur more than once for the same translation request.

CONT, bits [16:13]

This field indicates the number of contiguous StreamIDs that the result of this transaction applies to.

This field is encoded to give the span of the contiguous block as 2^{CONT} StreamIDs. When this field is non-zero, SID[CONT-1:0] in the translation request can be ignored when determining whether this translation matches future transactions.

When the value of FAULT_TYPE is not StreamDisabled, this field is Reserved, SBZ.

DO_NOT_CACHE, bit [12]

This bit indicates to the TBU when not to cache a fault response.

0 Can be cached

1 Must not be cached

When the value of FAULT_TYPE is not StreamDisabled or not GlobalDisabled, the value of this field must be 1.

TRANSLATION_ID[7:0], bits [11:4]

See TRANSLATION_ID[11:8], bits [31:28],

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for upstream messages, see *DTI-TBU protocol upstream messages* on page 2-25

0b0001 DTI_TBU_TRANS_FAULT.

3.2.4 Additional rules on permitted translation responses

Rules when IA out of range

The following rules limit the legal translation responses when the IA is out of range:

- If the TCU receives a translation request with DTI_TBU_TRANS_REQ.IA[63:56] != 0x00 and DTI_TBU_TRANS_REQ.IA[63:56] != 0xFF, it must complete the translation with either:
  - A DTI_TBU_TRANS_FAULT message
  - A DTI_TBU_TRANS_RESP message with BYPASS = 0 and TBI = 1.
- If the TCU receives a translation request with DTI_TBU_TRANS_IA[55:52] != 0x0 and DTI_TBU_TRANS_IA[55:52] != 0xF, the TCU must complete the translation with a DTI_TBU_TRANS_FAULT message.

A DTI_TBU_TRANS_FAULT message with TYPE = TranslationStall does not complete the transaction and therefore is not affected by the rules above.

For example, if the TCU receives a translation request with DTI_TBU_TRANS_REQ.IA[55:52] != 0x0:

- The TCU is permitted to return a DTI_TBU_TRANS_FAULT message with TYPE = TranslationStall, followed by a DTI_TBU_TRANS_FAULT message with TYPE = Abort.
- The TCU is not permitted to return a DTI_TBU_TRANS_FAULT message with TYPE = TranslationStall, followed by a DTI_TBU_TRANS_RESP message.

These rules were not specified by previous versions of the DTI specification, but do not change the behavior of DTI-TBUv1 systems because the SMMUv3 architecture requires this behavior.

Faulting expressions of the translation request message

The TCU can only return a DTI_TBU_TRANS_RESP message when permission is granted for the transaction that is described in the translation request.

The context for computing whether or not the permissions are legal is as follows:

```
bit effective_InD = ((RESP.INSTCFG == “Use incoming”) && REQ.InD) || (RESP.INSTCFG == “Instruction”);
bit effective_PnU = ((RESP.PRIVCFG == “Use incoming”) && REQ.PnU) || (RESP.PRIVCFG == “Privileged”);
bit effective_NS = DTI_TBU_CONDIS_ACK.VERSION == 1 ? RESP.NS : (RESP.NSOVR ? RESP.NS : REQ.NS);
req_R = ((REQ.PERM[1:0] == “R”) && !effective_InD) || (REQ.PERM[1:0] == “RW”)
req_W = ((REQ.PERM[1:0] == “W”) || (REQ.PERM[1:0] == “RW”))
req_X = (REQ.PERM[1:0] == “R”) && effective_InD
```

Within this context, it is a protocol error for either of the following expressions to be true:

```
!RESP.BYPASS && (
(!RESP.ALLOW_UR && req_R && !effective_PnU) ||
(!RESP.ALLOW_UW && req_W && !effective_PnU) ||
(!RESP.ALLOW_UX && req_X && !effective_PnU) ||
(!RESP.ALLOW_PR && req_R && effective_PnU)  ||
(!RESP.ALLOW_PW && req_W && effective_PnU)  ||
(!RESP.ALLOW_PX && req_X && effective_PnU))

RESP.BYPASS && REQ.SEC_SID && !RESP.ALLOW_NSX && req_X && effective_NS
```

3.2.5 Calculating transaction attributes

This section describes how the translated attributes of a transaction are calculated.

The set of possible transaction attributes is the same as those described in the *Arm Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*. The transaction attributes are composed of:

- Memory type
- Shareability
- Allocation hints

Fields used to calculate the attributes

To calculate the translated transaction attributes, the attributes of the untranslated transaction are used with the following fields of the translation response:

- BYPASS
- STRW
- ATTR
- SH
- ATTR_OVR
- ALLOCCFG

Note

The ATTR_OVR field is not always present, because it uses the same bits as the ASID field.

The ATTR_OVR field is composed of subfields that are shown in Table 3-4.

Table 3-4 ATTR_OVR subfields

Field bits	Field name
[3:0]	MemAttr
[4]	MTCFG
[6:5]	SHCFG
[8:7]	**DTI-TBUv1:** Reserved, SBZ **DTI-TBUv2:** NSCFG
[15:9]	Reserved, SBZ

Steps used to calculate the attributes

The TBU computes a translated transaction's attributes using the following process:

1. If the untranslated transaction does not have allocation hints, then they are treated as read-allocate, write-allocate, non-transient.
2. If ATTR_OVR is valid and MTCFG is set, then the memory type is replaced by the values in the ATTR_OVR.MemAttr field. For more information, see *The MemAttr and MTCFG fields* on page 3-58.
3. The allocation hints are modified based on the value of ALLOCCFG. For more information, see *The ALLOCCFG field* on page 3-60.
4. The shareability domain is modified based on the value of SHCFG. For more information see *The SHCFG field* on page 3-60.
5. The attributes are combined with the attributes in the ATTR and SH fields. For more information, see *Combining the translation response attributes* on page 3-61C.
6. A consistency check is applied to eliminate illegal attribute combinations. For more information, see *Consistency check on combination of translation attributes* on page 3-63.

The precise algorithm is:

```
MemoryAttributes MemoryAttributesOverride(MemoryAttributes attr_in, DTI_TBU_TRANS_RESP resp, bit
is_cmo_trans)

    MemoryAttributes attr_out;

    attr_out = attr_in;

    // The is_cmo_trans bit is set if the transaction is a Cache Maintenance
    // Operation (CMO) as defined in SMMUv3. This does not include
    // transactions which combine a CMO with a read or write transaction.
    if (is_cmo_trans) then
        attr_out = ApplyCMOAttributes(attr_out);
    attr_out = ConsistencyCheck(attr_out);

    if (resp.BYPASS == '1' || resp.STRW == EL1_S2) then
        if (resp.ATTR_OVR.MTCFG == '1') then
            attr_out = ModifyMemoryType(attr_out, resp.ATTR_OVR.MemAttr);
        ModifyAllocHints(attr_out, resp.ALLOCCFG);
        ModifyShareability(attr_out, resp.ATTR_OVR.SHCFG);
        attr_out = ConsistencyCheck(attr_out);
    else
        ModifyAllocHints(attr_out, resp.ALLOCCFG);

    attr_out = CombineAttributes(attr_out, resp);
    attr_out = ConsistencyCheck(attr_out);

    if (is_cmo_trans) then
        attr_out = ApplyCMOAttributes(attr_out);

    return attr_out;


MemoryAttributes ApplyCMOAttributes(MemoryAttributes current_attr)

    current_attr.type = MemType_Normal;
    current_attr.inner.attrs = MemAttr_WB;
    current_attr.inner.ReadAllocate = ‘1’;
    current_attr.inner.WriteAllocate = ‘1’;
    current_attr.inner.Transient = ‘0’;
    current_attr.outer.attrs = MemAttr_WB;
    current_attr.outer.ReadAllocate = ‘1’;
    current_attr.outer.WriteAllocate = ‘1’;
    current_attr.outer.Transient = ‘0’;

    return current_attr;
```

The MemAttr and MTCFG fields

If the value of MTCFG is 1, then the MemAttr field provides the memory type override for incoming transactions. Table 3-5 shows the encoding of this field.

Table 3-5 Encoding of the MemAttr field

Field encoding	Memory type	Inner cacheability	Outer cacheability
0b0000	Device-nGnRnE	-	-
0b0001	Device-nGnRE	-	-
0b0010	Device-nGRE	-	-
0b0011	Device-GRE	-	-

Table 3-5 Encoding of the MemAttr field (continued)

Field encoding	Memory type	Inner cacheability	Outer cacheability
0b0100	Reserved	Reserved	Reserved
0b0101	Normal	Non-cacheable	Non-cacheable
0b0110	Normal	Write-Through Cacheable	Non-cacheable
0b0111	Normal	Write-Back Cacheable	Non-cacheable
0b1000	Reserved	Reserved	Reserved
0b1001	Normal	Non-cacheable	Write-Through Cacheable
0b1010	Normal	Write-Through Cacheable	Write-Through Cacheable
0b1011	Normal	Write-Back Cacheable	Write-Through Cacheable
0b1100	Reserved	Reserved	Reserved
0b1101	Normal	Non-cacheable	Write-Back Cacheable
0b1110	Normal	Write-Through Cacheable	Write-Back Cacheable
0b1111	Normal	Write-Back Cacheable	Write-Back Cacheable

The MemAttr field is used to modify transaction memory type as follows:

```
MemoryAttributes ModifyMemoryType(MemoryAttributes current_attr, bits(4) mem_attr)
         MemoryAttributes memattr_attributes = DecodeMemAttr(mem_attr);

         // Override type
         current_attr.type = memattr_attributes.type;

         // Override cacheability
         current_attr.inner.attrs = memattr_attributes.inner.attrs;
         current_attr.outer.attrs = memattr_attributes.outer.attrs;

         // And leave allocation hints untouched
         return current_attr;
```

The ALLOCCFG field

The ALLOCCFG field overrides the allocation hints according to the following algorithm:

```
MemoryAttributes ModifyAllocHints(MemoryAttributes current_attr, bits(4) alloccfg)

         // Don't override allocation hints
         if alloccfg<3> == '0' then
                  return current_attr;

         // ALLOCCFG is packed as:
         bit T = alloccfg<0>; // Transient
         bit WA = alloccfg<1>; // Write allocate
         bit RA = alloccfg<2>; // Read allocate

         current_attr.inner.Transient = T;
         current_attr.inner.ReadAllocate = RA;
         current_attr.inner.WriteAllocate = WA;
         current_attr.outer.Transient = T;
         current_attr.outer.ReadAllocate = RA;
         current_attr.outer.WriteAllocate = WA;

         return current_attr;
```

The SHCFG field

The SHCFG field overrides the shareability of the translation.

0b00	Non-shareable
0b01	Use incoming shareability attribute
0b10	Outer shareable
0b11	Inner shareable

See `ModifyShareability()` in *Memory attributes* on page A-122 for an example implementation.

The NSCFG field

NSCFG indicates when the NS bit of the incoming transaction is overridden before translation. The encodings of NSCFG are:

0b00	Use incoming
0b01	Reserved
0b10	Secure
0b11	Non-secure

When DTI_TBU_TRANS_REQ.SEC_SID is 0, NSCFG must be "Use incoming".

Combining the translation response attributes

The memory attributes of an incoming transaction and a translation response are combined according to the following algorithms:

DTI-TBUv1

```
MemoryAttributes CombineAttributes(MemoryAttributes attr_txn,DTI_TBU_TRANS_RESP resp)

        MemoryAttributes attr_resp = DecodeAttr(resp.ATTR);

        if ((resp.BYPASS == '0') && (resp.STRW IN {EL1, EL2, EL3})) then
                // The ATTR and SH fields replace the incoming memory type and
                // Shareability. The memory type and Shareability of the untranslated
                // transaction are ignored.
                attr_txn.type = attr_resp.type;
                attr_txn.inner.attrs = attr_resp.inner.attrs;
                attr_txn.outer.attrs = attr_resp.outer.attrs;
                attr_txn.SH = resp.SH;

                // The allocation hints computed for the transaction so far are combined
                // with the allocation hints from the ATTR field.
                attr_txn = CombineAllocHints(attr_txn, attr_resp);

        elsif ((resp.BYPASS == '0') && (resp.STRW == EL1_S2)) then
                // The memory type and shareability attributes computed for the
                // transaction so far are combined with ATTR and SH fields.
                attr_txn = CombineMemoryType(attr_txn, attr_resp);
                attr_txn.SH = CombineShareability(attr_txn.SH, resp.SH);

                // The allocation hints computed for the transaction so far are combined
                // with the allocation hints from the ATTR field.
                attr_txn = CombineAllocHints(attr_txn, attr_resp);

        elsif (resp.BYPASS == '1') then
                // The memory type, Shareability and allocation hints computed so far
                // are used directly.
                attr_txn = attr_txn;

        return attr_txn;
```

DTI-TBUv2

```
MemoryAttributes CombineAttributes(MemoryAttributes attr_txn, DTI_TBU_TRANS_RESP resp)
        MemoryAttributes attr_resp = DecodeAttr(resp.ATTR);

        if (resp.BYPASS == '0') then
                if (resp.COMB_MT == '0') then
                        attr_txn = ReplaceMemoryType(attr_txn, attr_resp);
                elsif (resp.COMB_MT == '1') then
                        attr_txn = CombineMemoryType(attr_txn, attr_resp);

                if (resp.COMB_ALLOC == '0') then
                        attr_txn = ReplaceAllocHints(attr_txn, attr_resp);
                elsif (resp.COMB_ALLOC == '1') then
                        attr_txn = CombineAllocHints(attr_txn, attr_resp);

                if (resp.COMB_SH == '0') then
                        attr_txn.SH = resp.SH;
                elsif (resp.COMT_SH == '1') then
                        attr_txn.SH = CombineShareability(attr_txn.SH, resp.SH);

        return attr_txn;
```

When memory type, shareability and allocation hints are combined, the result is the strongest of each, as shown in Table 3-6.

Table 3-6 Combining the translation response attributes

Weakest → **Strongest**

Normal Write-Back	Normal Write-Through	Normal Non-cacheable	Device-GRE	Device-nGRE	Device-nGnRE	Device-nGnRnE
Non-shareable			Inner-shareable			Outer-shareable
Read-allocate						Read no-allocate
Write-allocate						Write no-allocate
Non-transient						Transient

See *Memory attributes* on page A-122 for the pseudocode implementation of this table.

Consistency check on combination of translation attributes

Between each step, the following additional conversions are performed to ensure that the attributes are consistent:

```
MemoryAttributes ConsistencyCheck(MemoryAttributes current_attr)

    if (current_attr.type != MemType_Normal ||
        (current_attr.inner.attrs == MemAttr_NC &&
         current_attr.outer.attrs == MemAttr_NC)) then
        current_attr.SH = OuterShareable;

    if (current_attr.type != MemType_Normal ||
        (current_attr.type == MemType_Normal &&
         current_attr.inner.attrs == MemAttr_NC)) then
        current_attr.inner.ReadAllocate = '1';
        current_attr.inner.WriteAllocate = '1';
        current_attr.inner.Transient = '0';

    if (current_attr.type != MemType_Normal ||
        (current_attr.type == MemType_Normal &&
         current_attr.outer.attrs == MemAttr_NC)) then
        current_attr.outer.ReadAllocate = '1';
        current_attr.outer.WriteAllocate = '1';
        current_attr.outer.Transient = '0';

    if (current.attr.inner.ReadAllocate == '0' && current.attr.inner.WriteAllocate == '0') then
        current.attr.inner.Transient == '0';

    if (current.attr.outer.ReadAllocate == '0' && current.attr.outer.WriteAllocate == '0') then
        current.attr.outer.Transient == '0';

    return current_attr;
```

In addition to these architectural attribute consistency rules, an implementation might include interconnect-specific consistency rules.

3.2.6 Speculative transactions and translations

A translation that is marked as speculative can be used for the following:

- Translating a speculative transaction.
- Prefetching a translation for a non-speculative transaction.

As a speculative translation request never results in a fault that is visible to software, it is permitted to be used for the prefetching of translations. A successful speculative translation request that is marked as cacheable can be used for future non-speculative transactions.

Note

A translation is permitted to be cached when the value of the DO_NOT_CACHE bit in the translation response message is 0.

When a speculative translation is not successful or it is non-cacheable, no translation is cached, and future non-speculative transactions will generate a new non-speculative translation request.

A speculative read transaction is permitted to use the cached translations of previous non-speculative translation requests, but is not permitted to cause a non-speculative translation request. When a speculative read transaction cannot be translated with cached translations that pass their permission check, then the TBU must either terminate the transaction with an abort, or request a new speculative translation.

Speculative write transactions are not supported.

Note

A speculative translation request does not have a specific transaction that is associated with it. As such, the PnU and InD fields in DTI_TBU_TRANS_REQ of the speculative translation request are not used and no permission check is performed as part of the translation. If a speculative translation is requested as a result of a speculative read transaction, the TBU must ensure that the transaction that caused it passes the permission check.

A speculative read transaction is never terminated as read 0, write ignored, even though the DTI_TBU_TRANS_FAULT.FAULT_TYPE field is always NonAbort for a speculative translation. A faulting speculative read transaction is always terminated with an abort.

3.3 Invalidation and synchronization message group

Invalidation operations are used by the TCU to indicate to the TBU that certain information must no longer be cached.

For more information about the caching model used by the DTI-TBU Protocol, see Chapter 4 *DTI-TBU Caching Model*

This section contains the following subsections:

3.3.1 DTI_TBU_INV_REQ

The DTI_TBU_INV_REQ message is used to request the invalidation of data that is stored in a cache.

Description

An invalidation request

Source

TCU

Usage constraints

The TCU must have at least one invalidation token.

Flow control result

The TCU consumes an invalidation token.

Field descriptions

The DTI_TBU_INV_REQ bit assignments are:

7	6	5	4	3	2	1	0	LSB
VA or IPA[63:16]								120 112 104 96
								88 80
VA or IPA[15:12]				Reserved				72
Reserved		INC_ASET1	RANGE					64
ASID or SID[31:16]								56 48
VMID or SID[15:0]								40 32
SSID[19:14]						SCALE[4:3] or SSID[13:12]		24
SCALE[2:0] or SSID[11:9]			NUM[4:0] or SSID[8:4]					16
TG or SSID[3:2]		TTL or SSID[1:0]		OPERATION[7:4]				8
OPERATION[3:0]				SLV_MSG_TYPE				0

VA/IPA, bits [127:76]

This field indicates the VA or IPA to be invalidated.

The encoding of the OPERATION field might cause this field to be invalid. When this field is invalid, it is Reserved, SBZ.

Bits [75:70]

Reserved, SBZ.

INC_ASET1, bit [69]

This bit indicates whether the ASET value of a translation affects its invalidation.

0 Translations with an ASET value of 0 are invalidated, only the shared set is invalidated.

1 The value of ASET has no effect, the shared and non-shared sets are invalidated.

Note

It is intended that this bit is 0 if the invalidation originates from a shared invalidate of the appropriate type. Some TLB invalidation operations always set this bit. This bit is always set for TLB invalidations originating from an explicit invalidate command to the SMMU.

This field is valid for all TLB invalidate operations. For all other invalidate operations, this field is ignored and is Reserved, SBZ.

This field must be 1 for the following TLB invalidate operations:

- TLBI_S_EL1_ALL
- TLBI_S_EL1_VAA
- TLBI_NS_EL1_ALL
- TLBI_NS_EL1_S1_VMID
- TLBI_NS_EL1_S12_VMID
- TLBI_NS_EL1_VAA
- TLBI_NS_EL1_S2_IPA
- TLBI_NS_EL2_ALL
- TLBI_NS_EL2_VAA
- TLBI_S_EL3_ALL
- TLBI_S_EL1_S1_VMID
- TLBI_S_EL1_S12_VMID
- TLBI_S_EL1_S2_S_IPA
- TLBI_S_EL1_S2_NS_IPA
- TLBI_S_EL2_ALL
- TLBI_S_EL2_VAA

RANGE, bits [68:64]

This field indicates the range of SIDs or VMIDs for invalidation.

The range to be invalidated is 2^{RANGE} 4KB pages.

When the value of the OPERATION field identifies this message as a CFGI_SID invalidate operation, the bottom RANGE bits of the SID field are ignored in both this message and the translations being considered for invalidation.

When the value of the OPERATION field identifies this message as a translation invalidate operation and the VMID field is valid for the operation:

- The bottom RANGE bits of the VMID field are ignored in both this message and the translations being considered for invalidation.
- The value of this field must not be greater than four.

The encoding of the OPERATION field might cause this field to be invalid. When this field is invalid, it is Reserved, SBZ.

ASID, bits [63:48], when OPERATION is a TLB invalidate operation.

This field indicates the ASID value to invalidate.

The encoding of the OPERATION field might cause this field to be invalid. When this field is invalid, it is Reserved, SBZ.

VMID, bits [47:32], when OPERATION is a TLB invalidate operation.

This field indicates the VMID value to invalidate.

The encoding of the OPERATION field might cause this field to be invalid. When this field is invalid, it is Reserved, SBZ.

When DTI_TBU_TRANS_REQ.SEC_SID is 1 and BYPASS is 0:

DTI-TBUv1

VMID must be 0

DTI-TBUv2

VMID can be non-zero

SID, bits [63:32], when OPERATION is a configuration invalidate operation.

This field indicates the StreamID to invalidate.

The encoding of the OPERATION field might cause this field to be invalid. When this field is invalid, it is Reserved, SBZ.

SSID, bits [31:12], when OPERATION is a configuration invalidate operation.

This field indicates the SubstreamID to invalidate.

The encoding of the OPERATION field might cause this field to be invalid. When this field is invalid, it is Reserved, SBZ.

SCALE, bits [25:21], when OPERATION is a TLB invalidate operation.

This field relates to Range invalidate operations. For more information, see *DTI-TBU invalidation operations* on page 3-71.

NUM, bits [20:16], when OPERATION is a TLB invalidate operation.

This field relates to Range invalidate operations. For more information, see *DTI-TBU invalidation operations* on page 3-71.

TG, bits 15:14], when OPERATION is a TLB invalidate operation.

This field relates to Range invalidate operations. For more information, see *DTI-TBU invalidation operations* on page 3-71.

TTL, bits [13:12], when OPERATION is a TLB invalidate operation.

This field relates to Range invalidate operations. For more information, see *DTI-TBU invalidation operations* on page 3-71.

OPERATION, bits [11:4]

This field identifies the type of invalidation operation being performed.

When a TBU receives a message with an unrecognized OPERATION field value, this specification recommends that the TBU acknowledges the invalidation without performing any operation. For the encoding of this field and information on the effects of the invalidate operations, see *DTI-TBU invalidation operations* on page 3-71.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for upstream messages, see *DTI-TBU protocol upstream messages* on page 2-25.

`0b0100` DTI_TBU_INV_REQ

3.3.2 DTI_TBU_INV_ACK

The DTI_TBU_INV_ACK message is used to acknowledge an invalidation request.

Description

An invalidation acknowledgement

Source

TBU

Usage constraints

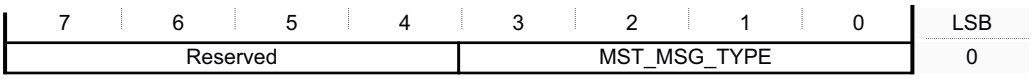
The TCU must have previously issued an invalidation request that has not yet been acknowledged.

Flow control result

The TBU returns an invalidation token to the TCU.

Field descriptions

The DTI_TBU_INV_ACK bit assignments are:



Bits [7:4]

Reserved, SBZ.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for downstream messages, see *DTI-ATS protocol downstream messages* on page 2-26.

0b0100 DTI_TBU_INV_ACK.

3.3.3 DTI_TBU_SYNC_REQ

The DTI_TBU_SYNC_REQ message is used to request synchronization of the TBU and TCU.

Description

A synchronization request

Source

TCU

Usage constraints

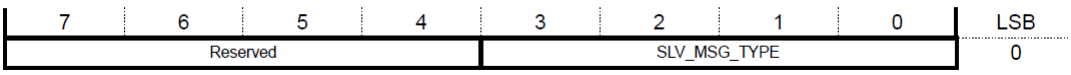
There must be no currently unacknowledged synchronization requests.

Flow control result

None

Field descriptions

The DTI_TBU_SYNC_REQ bit assignments are:



Bits [7:4]

Reserved, SBZ.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for upstream messages, see *DTI-TBU protocol upstream messages* on page 2-25.

0b0101 DTI_TBU_SYNC_REQ

3.3.4 DTI_TBU_SYNC_ACK

The DTI_TBU_SYNC_ACK message is used to acknowledge a synchronization request.

Description

A synchronization acknowledge

Source

TBU

Usage constraints

There must currently be an unacknowledged synchronization request.

Flow control result

None

Field descriptions

The DTI_TBU_SYNC_ACK bit assignments are:

7	6	5	4	3	2	1	0	LSB
Reserved				MST_MSG_TYPE				0

Bits [7:4]

Reserved, SBZ.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for downstream messages, see *DTI-TBU protocol downstream messages* on page 2-25.

`0b0101` DTI_TBU_SYNC_ACK

3.3.5 The DTI-TBU invalidation sequence

The invalidation sequence describes how individual invalidate messages interact with translation messages.

For all translations that are affected by the invalidation, the order in which they arrive at the TBU determines how they are handled. Figure 3-1 shows the invalidation phases in which an affected DTI_TBU_TRANS_RESP can arrive.

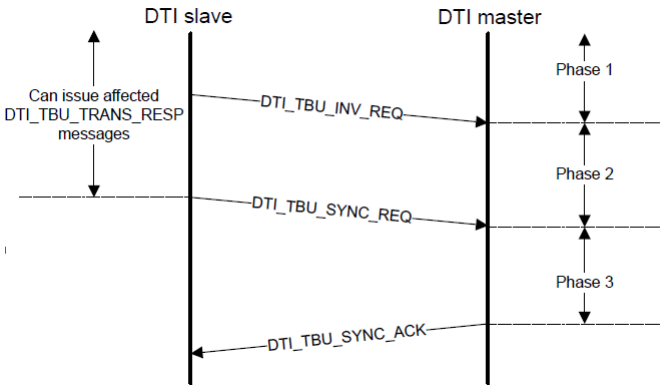


Figure 3-1 Phases of the invalidation sequence

The invalidation phases of the invalidation sequence are delimited by the following events:

1. A DTI_TBU_INV_REQ message
2. The following DTI_TBU_SYNC_REQ
3. The following DTI_TBU_SYNC_ACK

Note

Each DTI_TBU_INV_REQ message is followed by a DTI_TBU_INV_ACK message. The DTI_TBU_INV_ACK message is only used for flow control, it does not affect the invalidation sequence or indicate completion of the invalidate operation.

When a DTI_TBU_SYNC_REQ message is received, the TBU must ensure both:

- Translations within the scope of previous invalidations have been invalidated.
- Transactions that use them have completed downstream.

When both are ensured, the TBU can return a DTI_TBU_SYNC_ACK message. The actions that must be taken depend upon in what phase of the invalidation sequence, the affected DTI_TBU_TRANS_RESP messages arrived. The following table describes the phases and required actions.

Table 3-7 Phases and actions of an invalidation sequence

Sequence phase	Actions
Before the corresponding DTI_TBU_INV_REQ.	The TBU must identify which translations must be invalidated and which transactions must be completed before returning the DTI_TBU_SYNC_ACK message. These translations might or might not be marked as DO_NOT_CACHE.
After the corresponding DTI_TBU_INV_REQ but before the DTI_TBU_SYNC_REQ.	If the translation is based upon invalidated data then it will be marked as DO_NOT_CACHE. The TBU must invalidate translations marked as DO_NOT_CACHE and complete transactions using those translations before returning a DTI_TBU_SYNC_ACK.
After the DTI_TBU_SYNC_REQ.	These translations are out of scope of the current invalidation synchronization operation and play no part in the timing of the DTI_TBU_SYNC_ACK. The TCU delays issuing the DTI_TBU_SYNC_REQ if necessary to ensure this.

Overlapping invalidations

New DTI_TBU_INV_REQ messages can be sent after the DTI_TBU_SYNC_REQ has been sent, even if this is before the expected DTI_TBU_SYNC_ACK response is received. In all cases, an invalidation is only included in a synchronization if it is sent before the DTI_TBU_SYNC_REQ message.

A DTI_TBU_SYNC_REQ message can be sent after a DTI_TBU_INV_REQ is sent but before a DTI_TBU_INV_ACK is received. In this case, the invalidation is within scope of the synchronization operation. The DTI_TBU_INV_ACK message is solely for the purposes of returning invalidation tokens and does not affect synchronization operations.

Deadlock avoidance in the invalidation sequence

In order to avoid deadlocks, the following rules must be followed:

- The rules for DTI-TBUv1 and DTI-TBUv2 are different for the following case:

  DTI-TBUv1

  A TBU must not wait for an outstanding translation to complete before returning a DTI_TBU_SYNC_ACK message. Any outstanding translations must be discarded on receipt of a DTI_TBU_SYNC_REQ.

  DTI-TBUv2

  A TBU must not wait for an outstanding translation that has returned a fault with FAULT_TYPE TranslationStall to complete before returning a DTI_TBU_SYNC_ACK message. A TBU can wait for completion of an outstanding transaction that has not returned a fault with FAULT_TYPE TranslationStall. If the transaction returns a TranslationStall after the DTI_TBU_SYNC_REQ is received, it must be able to return a DTI_TBU_SYNC_ACK without waiting for the completion of that translation.

  Example 3-1 shows a case where failure to obey this rule will create a deadlock.

- The DTI_TBU_INV_REQ and DTI_TBU_INV_ACK messages must not wait for an outstanding DTI_TBU_SYNC_ACK message to be returned. Invalidation operations must be able to proceed without waiting for downstream transactions to complete, this is because those transactions might not be able to complete until the invalidation has been accepted.

Example 3-1 Deadlock caused by incorrect invalidation behavior in the TBU

Consider the following sequence:

1. Transaction A is received and a translation request is issued.
2. Transaction B is received, which must be ordered behind transaction A according to the bus protocol, and a translation request is issued.
3. The translation request for transaction A results in a stalling fault in the TCU, which cannot progress further until system software instructs the TCU to either retry or abort the translation. No response can be returned to the TBU until this occurs.
4. A translation response is received for transaction B, which is marked as DO_NOT_CACHE.
5. A DTI_TBU_SYNC_REQ is received.

In this case, the DTI_TBU_SYNC_ACK cannot be returned until the transaction B completes. This cannot occur until transaction A is issued, which cannot occur until the translation is received for transaction A, which would break the above requirement. Instead, the TBU should discard the translation for transaction B so that the DTI_TBU_SYNC_ACK can be returned, and re-request the translation for transaction B.

3.3.6 DTI-TBU invalidation operations

This section describes the DTI-TBU cache invalidation operations.

Types of invalidation operation

Table 3-8 specifies the OPERATION field encodings for DTI-TBUv1. It describes how the type of invalidation being performed affects the scope of the DTI_TBU_INV_REQ message for DTI-TBUv1. Other encodings of the OPERATION field are Reserved.

Table 3-8 DTI-TBUv1 list of invalidation operations

Code	Invalidation operation	StreamWorld affected	SEC_SID affected	Valid fields
0x80	TLBI_S_EL1_ALL	EL1	Secure	INC_ASET1
0x81	TLBI_S_EL1_VAA	EL1	Secure	VA, INC_ASET1
0x88	TLBI_S_EL1_ASID	EL1	Secure	ASID, INC_ASET1
0x89	TLBI_S_EL1_VA	EL1	Secure	ASID, VA, INC_ASET1
0xA0	TLBI_NS_EL1_ALL	EL1, EL1-S2	Non-secure	INC_ASET1
0xB2	TLBI_NS_EL1_S1_VMID	EL1	Non-secure	VMID, RANGE, INC_ASET1
0xB0	TLBI_NS_EL1_S12_VMID	EL1, EL1-S2	Non-secure	VMID, RANGE, INC_ASET1
0xB1	TLBI_NS_EL1_VAA	EL1	Non-secure	VMID, VA, RANGE, INC_ASET1
0xB8	TLBI_NS_EL1_ASID	EL1	Non-secure	VMID, ASID, RANGE, INC_ASET1
0xB9	TLBI_NS_EL1_VA	EL1	Non-secure	VMID, ASID, VA, RANGE, INC_ASET1
0xB5	TLBI_NS_EL1_S2_IPA	EL1-S2	Non-secure	VMID, IPA, RANGE, INC_ASET1
0xE0	TLBI_NS_EL2_ALL	EL2	Non-secure	INC_ASET1
0xE1	TLBI_NS_EL2_VAA	EL2	Non-secure	VA, INC_ASET1
0xE8	TLBI_NS_EL2_ASID	EL2	Non-secure	ASID, INC_ASET1
0xE9	TLBI_NS_EL2_VA	EL2	Non-secure	ASID, VA, INC_ASET1
0x40	TLBI_S_EL3_ALL	EL3	Secure	INC_ASET1
0x41	TLBI_S_EL3_VA	EL3	Secure	VA, INC_ASET1
0x00	CFGI_S_ALL	-	Secure	-
0x10	CFGI_S_SID	-	Secure	SID, RANGE
0x18	CFGI_S_SID_SSID	-	Secure	SID, SSID
0x20	CFGI_NS_ALL	-	Non-secure	-
0x30	CFGI_NS_SID	-	Non-secure	SID, RANGE
0x38	CFGI_NS_SID_SSID	-	Non-secure	SID, SSID
0x06	INV_ALL	All	All	-

Table 3-9 specifies the OPERATION field encodings for DTI-TBUv2. It describes how the type of invalidation being performed affects the scope of the DTI_TBU_INV_REQ message for DTI-TBUv2. Other encodings of the OPERATION field are Reserved.

Table 3-9 DTI-TBUv2 list of invalidation operations

Code	Invalidation operation	StreamWorld affected	SEC_SID affected	Valid fields
0x80	TLBI_S_EL1_ALL	EL1, EL1-S2	Secure	INC_ASET1
0x81	TLBI_S_EL1_VAA	EL1	Secure	VMID, VA, RANGE, INC_ASET1, SCALE, NUM, TG, TTL
0x82	TLBI_S_EL1_S1_VMID	EL1	Secure	VMID, RANGE, INC_ASET1
0x85	TLBI_S_EL1_S2_NS_IPA	EL1-S2	Secure	VMID, IPA, RANGE, INC_ASET1, SCALE, NUM, TG, TTL
0x88	TLBI_S_EL1_ASID	EL1	Secure	VMID, ASID, RANGE, INC_ASET1
0x89	TLBI_S_EL1_VA	EL1	Secure	VMID, ASID, VA, RANGE, INC_ASET1, SCALE, NUM, TG, TTL
0x90	TLBI_S_EL1_S12_VMID	EL1, EL1-S2	Secure	VMID, RANGE, INC_ASET
0x95	TLBI_S_EL1_S2_S_IPA	EL1-S2	Secure	VMID, IPA, RANGE, INC_ASET1, SCALE, NUM, TG, TTL
0xA0	TLBI_NS_EL1_ALL	EL1, EL1-S2	Non-secure	INC_ASET1
0xB2	TLBI_NS_EL1_S1_VMID	EL1	Non-secure	VMID, RANGE, INC_ASET1
0xB0	TLBI_NS_EL1_S12_VMID	EL1, EL1-S2	Non-secure	VMID, RANGE, INC_ASET1
0xB1	TLBI_NS_EL1_VAA	EL1	Non-secure	VMID, VA, RANGE, INC_ASET1, SCALE, NUM, TG, TTL
0xB8	TLBI_NS_EL1_ASID	EL1	Non-secure	VMID, ASID, RANGE, INC_ASET1
0xB9	TLBI_NS_EL1_VA	EL1	Non-secure	VMID, ASID, VA, RANGE, INC_ASET1, SCALE, NUM, TG, TTL
0xB5	TLBI_NS_EL1_S2_IPA	EL1-S2	Non-secure	VMID, IPA, RANGE, INC_ASET1, SCALE, NUM, TG, TTL
0xC0	TLBI_S_EL2_ALL	EL2	Secure	INC_ASET1
0xC9	TLBI_S_EL2_VA	EL2	Secure	ASID, VA, INC_ASET1, SCALE, NUM, TG, TTL
0xC1	TLBI_S_EL2_VAA	EL2	Secure	VA, INC_ASET1, SCALE, NUM, TG, TTL
0xC8	TLBI_S_EL2_ASID	EL2	Secure	ASID, INC_ASET1
0xE0	TLBI_NS_EL2_ALL	EL2	Non-secure	INC_ASET1
0xE1	TLBI_NS_EL2_VAA	EL2	Non-secure	VA, INC_ASET1, SCALE, NUM, TG, TTL
0xE8	TLBI_NS_EL2_ASID	EL2	Non-secure	ASID, INC_ASET1
0xE9	TLBI_NS_EL2_VA	EL2	Non-secure	ASID, VA, INC_ASET1, SCALE, NUM, TG, TTL
0x40	TLBI_S_EL3_ALL	EL3	Secure	INC_ASET1

Table 3-9 DTI-TBUv2 list of invalidation operations (continued)

Code	Invalidation operation	StreamWorld affected	SEC_SID affected	Valid fields
0x41	TLBI_S_EL3_VA	EL3	Secure	VA, INC_ASET1, SCALE, NUM, TG, TTL
0x00	CFGI_S_ALL	-	Secure	-
0x10	CFGI_S_SID	-	Secure	SID, RANGE
0x18	CFGI_S_SID_SSID	-	Secure	SID, SSID
0x20	CFGI_NS_ALL	-	Non-secure	-
0x30	CFGI_NS_SID	-	Non-secure	SID, RANGE
0x38	CFGI_NS_SID_SSID	-	Non-secure	SID, SSID
0x06	INV_ALL	All	All	-

If the value of the GLOBAL bit in the translation response is 1, the ASID field in that translation is ignored during invalidate operations. Invalidate operations that include an ASID are treated as follows:

- Invalidate operations, including a VA and ASID, invalidate the translation regardless of the ASID being invalidated.
- Invalidate operations including an ASID, but without a VA, do not invalidate the translation.

The following invalidation operations will invalidate GlobalBypass and GlobalDisable translations of the appropriate security level:

- CFGI_NS_ALL
- CFGI_S_ALL
- INV_ALL

Note

Invalidation operations can be issued without a corresponding SMMUv3 invalidate command. A TCU issues CFGI_NS_ALL and CFGI_S_ALL invalidation and sync operations to invalidate GlobalBypass and GlobalDisable translations as part of the process for changing certain SMMUv3 control registers.

The INV_ALL operation invalidates all caches, including Secure and Non-secure TLB and configuration caches as well as GlobalBypass and GlobalDisable translations.

Range Invalidate operations

DTI-TBUv2 supports Range Invalidation operations. These operations do not involve any RANGE fields specified in messages.

The range of addresses in scope of the invalidation operation is given by:

$$Range = ((NUM+1) * 2^{SCALE}) * Translation\_Granule\_Size$$

Table 3-10 shows the Translation_Granule_Size mapping:

Table 3-10 Translation_Granule_Size mapping

TG	Translation_Granule_Size
0b01	4 KB
0b10	16 KB
0b11	64 KB

The set of addresses A to be invalidated is given by:

```
Address <= A < Address + Range
```

An invalidation affects a translation if any address to be invalidated is within the range of the translation, as defined by INVAL_RNG in the translation response.

When TG == 0b00:

- The range is a single address
- The SCALE and NUM fields, are Reserved, SBZ

An invalidation might be limited to translations with specific values of INVAL_RNG in the translation response. Table 3-11 indicates encodings of INVAL_RNG that are within scope of an invalidation, dependent upon the TG and TTL fields:

Table 3-11 DTI-TBUv2 encodings of INVAL_RNG

TG	TTL	INVAL_RNG affected
0b00	0b00	All
0b01	0b00	4 KB, 2 MB, 1 GB
0b01	0b01	1 GB
0b01	0b10	2 MB
0b01	0b11	4 KB
0b10	0b00	16 KB, 32 MB
0b10	0b10	32 MB
0b10	0b11	16 KB
0b11	0b00	64 KB, 512 MB, 4 TB
0b11	0b01	4 TB
0b11	0b10	512 MB
0b11	0b11	64 KB

All other combinations of TG and TTL are Reserved:

- The combination TG == 0b00, TTL != 0b00 is legal in SMMUv3.2 invalidation commands but not legal in DTI, and must be mapped to TG == 0b00, TTL == 0b00 in DTI.
- The combination TG == 0b10, TTL == 0b01 is legal in SMMUv3.2 invalidation commands and Armv8.4 range invalidate operations but not legal in DTI, and must be mapped to TG == 0b10, TTL == 0b00 in DTI.

A TCU must return INVAL_RNG values that ensure correct invalidation by a TBU implementing the above rules. That means that INVAL_RNG must correctly identify the translation granule and level of the translation at the first encountered stage of translation and its value must not depend on the Contiguous bit in the leaf page table entry.

The SCALE, NUM, TG and TTL fields are valid for all invalidation operations where the VA/IPA field is valid.

When the fields of the invalidation operation match any of the following, no invalidation is required to occur:

- TG == `0b01` && TTL == `0b01` && Address[29:12] != 0
- TG == `0b01` && TTL == `0b10` && Address[20:12] != 0
- TG == `0b10` && TTL == `0b10` && Address[24:14] != 0
- TG == `0b10` && Address[13:12] != 0
- TG == `0b11` && TTL == `0b01` && Address[41:16] != 0
- TG == `0b11` && TTL == 0b10 && Address[28:16] != 0
- TG == `0b11` && Address[15:12] != 0

The following combination of field values is illegal: TG != `0b00` && TTL == 0b00 && NUM == 0 && SCALE == 0. A single address without TTL or range information should instead be encoded with TG == 0b00.

Configuration invalidate operations

Configuration invalidate operations invalidate configuration cache information. They do not need to invalidate TLB information unless the TLB and configuration information is held in a combined cache.

Table 3-12 shows the SMMUv3 commands that map that to DTI configuration invalidate operations.

Table 3-12 Mappings of SMMUv3 commands onto DTI invalidate operations

SMMUv3 command	DTI invalidate operation
CMD_CFGI_ALL	CFGI_S_ALL, CFGI_NS_ALL
CMD_CFGI_STE	CFGI_S_SID, CFGI_NS_SID
CMD_CFGI_STE_RANGE	CFGI_S_SID, CFGI_NS_SID
CMD_CFGI_CD_ALL	CFGI_S_SID, CFGI_NS_SID
CMD_CFGI_CD	CFGI_S_SID_SSID, CFGI_NS_SID_SSID

For any translation that has 0 as the value of DTI_TBU_TRANS_REQ.SSV, the value of DTI_TBU_TRANS_REQ.SSID is treated as being 0 for the purpose of the CFGI_S_SID_SSID and CFG_NS_SID_SSID operations.

3.4 Register access message group

The TBU provides IMPLEMENTATION DEFINED registers, which can be accessed using these messages. These registers provide information and control for the features of the TBU.

The DTI protocol supports 32-bit register accesses only. If 64-bit registers are implemented, they must be updated using multiple 32-bit accesses.

A TBU can implement up to 128KB of register space in both Secure and Non-secure states. The upper 64KB page is intended to be used to hold Page 1 of an SMMUv3 Performance Monitor Counter Group register file. The lower 64KB page is intended for all other registers.

This section contains the following subsections:

3.4.1 DTI_TBU_REG_WRITE

The DTI_TBU_REG_WRITE message is used to request a write to a register.

Description

A register write request

Source

TCU

Usage constraints

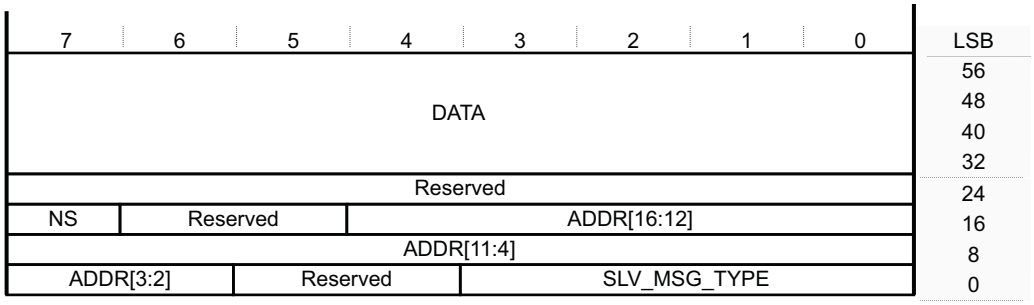
- The TCU must have no outstanding register reads or writes.
- DTI_TBU_CONDIS_ACK.SUP_REG was 1 during the connect sequence.

Flow control result

None

Field descriptions

The DTI_TBU_REG_WRITE bit assignments are:



DATA, bits [63:32]

This field holds the data to be written.

Bits [31:24]

Reserved, SBZ.

NS, bit [23]

This bit indicates the Security level of the register access.

0 Secure

1 Non-secure

Bits [22:21]

Reserved, SBZ.

ADDR, bits [20:6]

This field indicates the address of the register to be written to. Writes to unimplemented registers must be ignored.

Bits [5:4]

Reserved, SBZ.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for upstream messages, see *DTI-TBU protocol upstream messages* on page 2-25.

0b0110 DTI_TBU_REG_WRITE

3.4.2 DTI_TBU_REG_WACK

The DTI_TBU_REG_WACK message is used to acknowledge a register write request. Receipt of this message indicates a write has taken effect.

Description

A register write acknowledgement

Source

TBU

Usage constraints

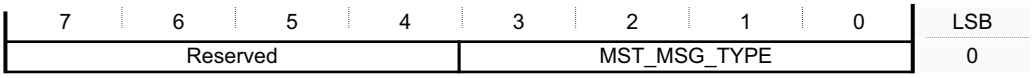
The TCU must have previously issued a register write request that has not yet been acknowledged.

Flow control result

None

Field descriptions

The DTI_TBU_REG_WACK bit assignments are:



Bits [7:4]

Reserved, SBZ.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for downstream messages, see *DTI-TBU protocol downstream messages* on page 2-25.

0b0110 DTI_TBU_REG_WACK

3.4.3 DTI_TBU_REG_READ

The DTI_TBU_REG_READ message is used to request a read from a register.

Description

A register read request

Source

TCU

Usage constraints

- The TCU must have no outstanding reads or writes.
- DTI_TBU_CONDIS_ACK.SUP_REG was 1 during the connect sequence.

Flow control result

None

Field descriptions

The DTI_TBU_REG_READ bit assignments are:

7	6	5	4	3	2	1	0	LSB
Reserved								24
NS	Reserved			ADDR[16:12]				16
ADDR[11:4]								8
ADDR[3:2]		Reserved			SLV_MSG_TYPE			0

Bits [31:24]

Reserved, SBZ.

NS, bit [23]

This bit indicates the Security level of the register access.

0 Secure

1 Non-secure

Bits [22:21]

Reserved, SBZ.

ADDR, bits [20:6]

This field indicates the address of the register to be written to. Reads from unimplemented registers must return 0 and have no other effect.

Bits [5:4]

Reserved, SBZ.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for upstream messages, see *DTI-TBU protocol upstream messages* on page 2-25.

0b0111 DTI_TBU_REG_READ

3.4.4 DTI_TBU_REG_RDATA

The DTI_TBU_REG_RDATA message is used to return the data from a register read request.

Description

A register read response

Source

TBU

Usage constraints

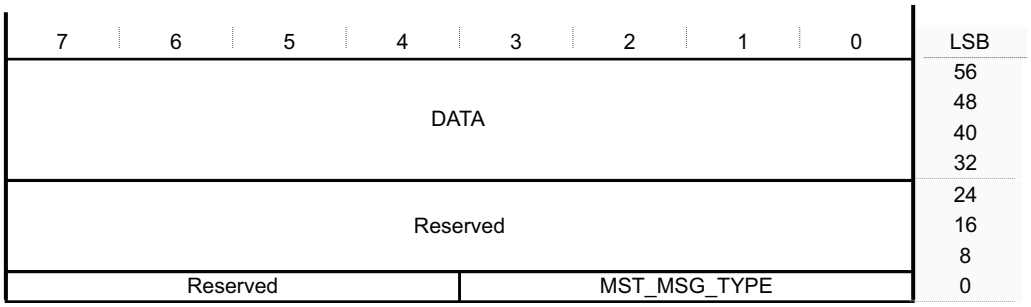
The TCU must have previously issued a register read request that has not yet received a response.

Flow control result

None

Field descriptions

The DTI_TBU_REG_RACK bit assignments are:



DATA, bits [63:32]

This field holds the read data.

Bits [31:4]

Reserved, SBZ.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for downstream messages, see *DTI-TBU protocol downstream messages* on page 2-25.

`0b0111` DTI_TBU_REG_RDATA

3.4.5 Deadlock avoidance in register accesses

A TBU must be able to respond to register access messages without requiring the completion of downstream transactions, or the progress of other DTI transactions.

Chapter 4
DTI-TBU Caching Model

This chapter describes the caching model for the DTI-TBU protocol.

It contains the following sections:

4.1 Caching model

The TBU implements a cache model where translation response information is cached depending upon its intended function. Architecturally, a TBU must implement the following caches, which are looked up in the following order:

- A global entry cache, for when translation is globally disabled.
- A configuration cache.
- A TLB.

Any implementation is permitted that is compatible with this cache model.

An implementation might implement a single cache that combines the lookup of two or more of these caches. Such an implementation is permitted if the invalidation operations still function in the order that is described here.

Each cache contains fields for the following:

Tag This is compared against future transactions or invalidations.

Scope This controls how much of the tag must match.

Data This is used to translate a transaction.

4.2 Lookup process

A lookup into the caches progresses as shown in Figure 4-1.

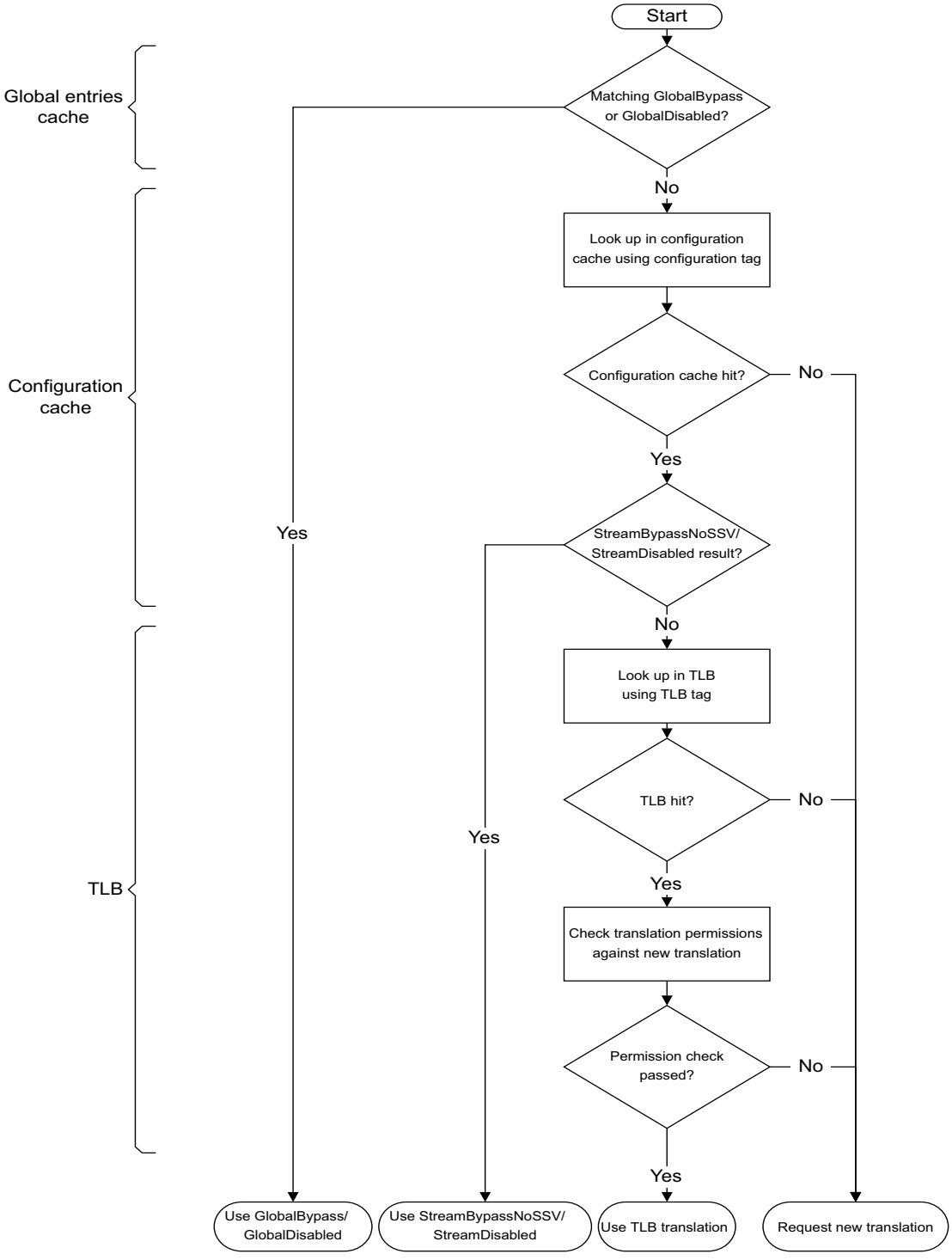


Figure 4-1 Lookup process

When there is a TLB hit on a cache lookup, the TBU must ensure that the stored translation matches the permission requirements of the new transaction. If the permission check fails, then the cached translation is not a match for the transaction. In this case, the TBU must request a new translation. The TCU might return a successful translation, or might return a translation fault for the transaction.

It is possible for multiple translations to match a transaction. In this case, a TBU can use any matching translation that has not been invalidated. The TBU is not required to use the most recent matching translation.

4.3 Global entry cache

The global entry cache can contain up to three entries:

- A GlobalBypass or GlobalDisabled entry for Secure transactions.
- A GlobalBypass or GlobalDisabled entry for Non-secure transactions that were not ATS translated.
- A GlobalBypass or GlobalDisabled entry for Non-secure transactions that were ATS translated.

The message fields that comprise the entry tag field combine to index these three entry types. The tag, scope, and data fields of a GlobalBypass cache entry are as follows:

Tag fields

- DTI_TBU_TRANS_REQ.SEC_SID
- DTI_TBU_TRANS_REQ.ATS

Scope fields

- DTI_TBU_TRANS_RESP.TRANS_RNG

Data fields

- DTI_TBU_TRANS_RESP.NSOVR
- DTI_TBU_TRANS_RESP.ALLOCCFG
- DTI_TBU_TRANS_RESP.NS
- DTI_TBU_TRANS_RESP.PRIVCFG
- DTI_TBU_TRANS_RESP.INSTCFG
- DTI_TBU_TRANS_RESP.ATTR_OVR
- DTI_TBU_TRANS_RESP.CTXTATTR
- DTI_TBU_TRANS_RESP.PARTID
- DTI_TBU_TRANS_RESP.PMG
- DTI_TBU_TRANS_RESP.MPAMNS

The tag, scope, and data fields of a GlobalDisable cache entry are as follows:

Tag fields

- DTI_TBU_TRANS_REQ.SEC_SID
- DTI_TBU_TRANS_REQ.ATS

Scope fields

None

Data fields

None

If a GlobalDisabled entry tag matches a transaction, then the transaction is always aborted.

4.4 Configuration cache

The configuration cache performs the following functions:

- Maps the incoming translation context fields to the TLB tags used by the page tables.
- Stores translation information affecting all transactions that are translated using a given context.
- Contains StreamDisabled entries for when translation is disabled for some streams.

The following tables show which DTI-TBU message fields are used to fill the Tag, Scope, and Data fields of entries in the configuration cache.

Tag fields

- DTI_TBU_TRANS_REQ.SEC_SID
- DTI_TBU_TRANS_REQ.ATST
- DTI_TBU_TRANS_REQ.SID
- DTI_TBU_TRANS_REQ.SSV
- DTI_TBU_TRANS_REQ.SSID

Scope fields

- DTI_TBU_TRANS_RESP.CONT
- DTI_TBU_TRANS_RESP.ALLOW_NSX

Data fields

- DTI_TBU_TRANS_RESP.BYPASS
- DTI_TBU_TRANS_RESP.STRW/BP_TYPE
- DTI_TBU_TRANS_RESP.DRE
- DTI_TBU_TRANS_RESP.DCP
- DTI_TBU_TRANS_RESP.NS
- DTI_TBU_TRANS_RESP.PRIVCFG
- DTI_TBU_TRANS_RESP.INSTCFG
- DTI_TBU_TRANS_RESP.ALLOCCFG
- DTI_TBU_TRANS_RESP.ASET/NSOVR
- DTI_TBU_TRANS_RESP.VMID
- DTI_TBU_TRANS_RESP.ASID/ATTR_OVR
- DTI_TBU_TRANS_RESP.CTXTATTR
- DTI_TBU_TRANS_RESP.PARTID
- DTI_TBU_TRANS_RESP.PMG
- DTI_TBU_TRANS_RESP.MPAMNS

The DTI_TBU_TRANS_RESP.BYPASS field indicates when the entry is a StreamBypassNoSSV entry.

The tag, scope, and data fields of a StreamDisabled cache entry are as follows:

Tag fields

- DTI_TBU_TRANS_REQ.SEC_SID
- DTI_TBU_TRANS_REQ.ATS
- DTI_TBU_TRANS_REQ.SID
- DTI_TBU_TRANS_REQ.SSV
- DTI_TBU_TRANS_REQ.SSID

Scope fields

- DTI_TBU_TRANS_FAULT.CONT

Data fields

None

4.5 TLB

The TLB uses information from the configuration cache to look up a saved translation for an instruction.

Translation failures reported using a DTI_TBU_TRANS_FAULT message are never stored in a TLB.

The following tables shows which DTI-TBU message fields are used to fill the Tag, Scope, and Data fields of entries in the TLB.

Tag fields

- DTI_TBU_TRANS_REQ.ATST
- DTI_TBU_TRANS_REQ.SEC_SID
- DTI_TBU_TRANS_REQ.IA
- DTI_TBU_TRANS_RESP.STRW
- DTI_TBU_TRANS_RESP.ASET
- DTI_TBU_TRANS_RESP.VMID
- DTI_TBU_TRANS_RESP.ASID

Scope fields

- DTI_TBU_TRANS_RESP.TBI
- DTI_TBU_TRANS_RESP.GLOBAL
- DTI_TBU_TRANS_RESP.TRANS_RNG
- DTI_TBU_TRANS_RESP.INVAL_RNG
- DTI_TBU_TRANS_RESP.ALLOW_UR
- DTI_TBU_TRANS_RESP.ALLOW_UW
- DTI_TBU_TRANS_RESP.ALLOW_UX
- DTI_TBU_TRANS_RESP.ALLOW_PR
- DTI_TBU_TRANS_RESP.ALLOW_PW
- DTI_TBU_TRANS_RESP.ALLOW_PX

Data fields

- DTI_TBU_TRANS_RESP.NS
- DTI_TBU_TRANS_RESP.OA
- DTI_TBU_TRANS_RESP.ATTR
- DTI_TBU_TRANS_RESP.SH
- DTI_TBU_TRANS_RESP.S1HWATTR
- DTI_TBU_TRANS_RESP.S2HWATTR
- DTI_TBU_TRANS_RESP.HWATTR
- DTI_TBU_TRANS_RESP.COMB

Chapter 5
DTI-ATS Messages

This chapter describes the message groups of the DTI-ATS protocol.

It contains the following sections:

5.1 Connection and disconnection message group

The DTI-ATS protocol is designed to enable a TCU to simultaneously support DTI-ATSv1 and DTI-ATSv2 connections from different PCIe RPs.

This section contains the following subsections:

5.1.1 DTI_ATS_CONDIS_REQ

The DTI_ATS_CONDIS_REQ message is used to initiate a connection or disconnection handshake.

Description

Connection state change request

Source

PCIe RP

Usage constraints

The PCIe RP can only send a disconnect request when:

- The channel is in the CONNECTED state.
- There are no outstanding translation requests.
- There are no outstanding page requests.
- The conditions for completing any future invalidation and sync are already met. In practice, the result is that all downstream transactions must be complete and all ATCs must be disabled and invalidated.

The PCIe RPs can only send a connect request when:

- The channel is in the DISCONNECTED state.

Flow control result

None

Field descriptions

The DTI_ATS_CONDIS_REQ bit assignments are:

7	6	5	4	3	2	1	0	LSB
Reserved							NO_TRANS	24
TOK_INV_GNT				TOK_TRANS_REQ[7:4]				16
TOK_TRANS_REQ[3:0]				VERSION				8
IMP DEF	Reserved		PROTOCOL	STATE	MST_MSG_TYPE			0

Bits [31:25]

Reserved, SBZ.

NO_TRANS, bit 24

When this bit is 1:

- The number of translation tokens requested is zero.
- The number of invalidation tokens granted is zero.
- None of the following messages are permitted to be sent:
  - DTI_ATS_TRANS_*
  - DTI_ATS_INV_*
  - DTI_ATS_SYNC_*

When STATE is 1, the value of this field must match the value of NO_TRANS in the previous connect request with STATE == 0.

TOK_INV_GNT, bits [23:20]

The meaning of this field depends on the value of the NO_TRANS field.

When NO_TRANS == 0:

This field indicates the number of invalidation tokens granted.

The number of invalidation tokens granted is equal to the value of this field plus one.

This field is ignored when the STATE field has a value of 0.

When NO_TRANS == 1:

Reserved, SBZ.

TOK_TRANS_REQ, bits [19:12]

The meaning of this field depends on the values of the STATE and NO_TRANS fields.

When NO_TRANS == 0 and STATE == 0:

This field indicates the number of translation tokens returned.

The number of translation tokens returned is equal to the value of this field plus one.

This field must be the value of the TOK_TRANS_GNT field that was received in the DTI_ATS_CONDIS_ACK message that acknowledged the connection of the channel.

TOK_TRANS is equal to the encoded value of this field plus one.

When NO_TRANS == 0 and STATE == 1:

This field indicates the number of translation tokens that are requested.

The number of translation tokens requested is equal to the value of this field plus one.

When NO_TRANS == 1:

Reserved, SBZ.

VERSION, bits [11:8]

This field indicates the requested protocol version.

`0b0000` DTI-ATSv1

`0b0001` DTI-ATSv2

All other encodings are reserved.

A PCIe RP can request any protocol version it supports. A TCU must accept requests for later protocol versions, including those not yet defined. The DTI_ATS_CONDIS_ACK message indicates the protocol version to use.

Bits [7:6]

Reserved, SBZ.

PROTOCOL, bit [5]

This bit indicates the protocol that is used by this PCIe RP.

1 DTI-ATS. This bit must be 1

STATE, bit [4]

This bit indicates the new channel state requested.

0 Disconnect request

1 Connect request

A Disconnect request can only be issued when the channel is in the CONNECTED state.

A Connect request can only be issued when the channel is in the DISCONNECTED state.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for downstream messages, see *DTI-ATS protocol downstream messages* on page 2-26.

`0b0000` DTI_ATS_CONDIS_REQ

5.1.2 DTI_ATS_CONDIS_ACK

The DTI_ATS_CONDIS_ACK message is used to accept or deny a request as part of the connect or disconnect handshake process.

Description

A connection state change acknowledgement

Source

TCU

Usage constraints

The PCIe RP must have previously issued an unacknowledged DTI_ATS_CONDIS_REQ message.

Flow control result

None

Field descriptions

The DTI_ATS_CONDIS_ACK bit assignments are:

7	6	5	4	3	2	1	0	LSB
Reserved							OAS[3]/Reserved	24
OAS[2:0]/Reserved			SUP_PRI	TOK_TRANS_GNT[7:4]				16
TOK_TRANS_GNT[3:0]				VERSION				8
Reserved			STATE	SLV_MSG_TYPE				0

Bits [31:21]

Reserved, SBZ.

OAS, bits [24:21]

DTI-ATSv1

Indicates the output address size, which is the maximum address size permitted for translated addresses.

`0b0000` 32 bits (4GB)
`0b0001` 36 bits (64GB)
`0b0010` 40 bits (1TB)
`0b0011` 42 bits (4TB)
`0b0100` 44 bits (16TB)
`0b0101` 48 bits (256TB)
`0b0110` 52 bits (4PB)

All other values are Reserved.

DTI-ATSv2

Reserved.

SUP_PRI, Bit [20]

Indicates that the PCIe ATS PRI messages are supported.

If the value of this bit is 0, then DTI_ATS_PAGE_REQ messages must not be issued.

When the value of STATE is 0, this bit is ignored.

DTI-ATSv2

If this bit is 0, then DTI_ATS_PAGE_RESP messages must not be issued.

TOK_TRANS_GNT, bits [19:12]

The meaning of this field depends on the value of DTI_ATS_CONDIS_REQ.NO_TRANS:

When DTI_ATS_CONDIS_REQ.NO_TRANS == 0

Indicates the number of pre-allocated tokens for translation requests that have been granted.

The number of translation tokens granted is equal to the encoded value of this field plus one.

The value of this field must not be greater than the value of the TOK_TRANS_REQ field in the DTI_ATS_CONDIS_REQ message that initiated the connection.

When the value of STATE is 0, this field is ignored.

When DTI_ATS_CONDIS_REQ.NO_TRANS == 1

Reserved, SBZ.

VERSION, bits [11:8]

This bit indicates the protocol version that the TCU has granted.

0b0000 DTI-ATSv1

0b0001 DTI-ATSv2

All other encodings are reserved.

The value of this field must not be greater than the value of the VERSION field in the DTI_ATS_CONDIS_REQ message.

Bits [7:5]

Reserved, SBZ.

STATE, bit [4]

This bit indicates the new DTI connection state. The possible values of this bit are:

0 DISCONNECTED

1 CONNECTED

When the value of STATE in the unacknowledged DTI_ATS_CONDIS_REQ message is 0, the value of this bit must be 0.

When the value of STATE in the unacknowledged DTI_ATS_CONDIS_ REQ message is 1, this field can be 0 or 1. For example, it can be 0 if there are no translation tokens available. This normally indicates a serious system configuration failure.

SLV_MSG_TYPE, bits [3:0]

Identifies the message type. The value of this field is taken from the list of encodings for upstream messages, see *DTI-ATS protocol upstream message* on page 2-26.

0b0000 DTI_ATS_CONDIS_ACK

5.2 Translation request message group

This section contains the following subsections:

5.2.1 DTI_ATS_TRANS_REQ

The DTI_ATS_TRANS_REQ message is used to initiate a translation request.

Description

A translation request

Source

PCIe RP

Usage constraints

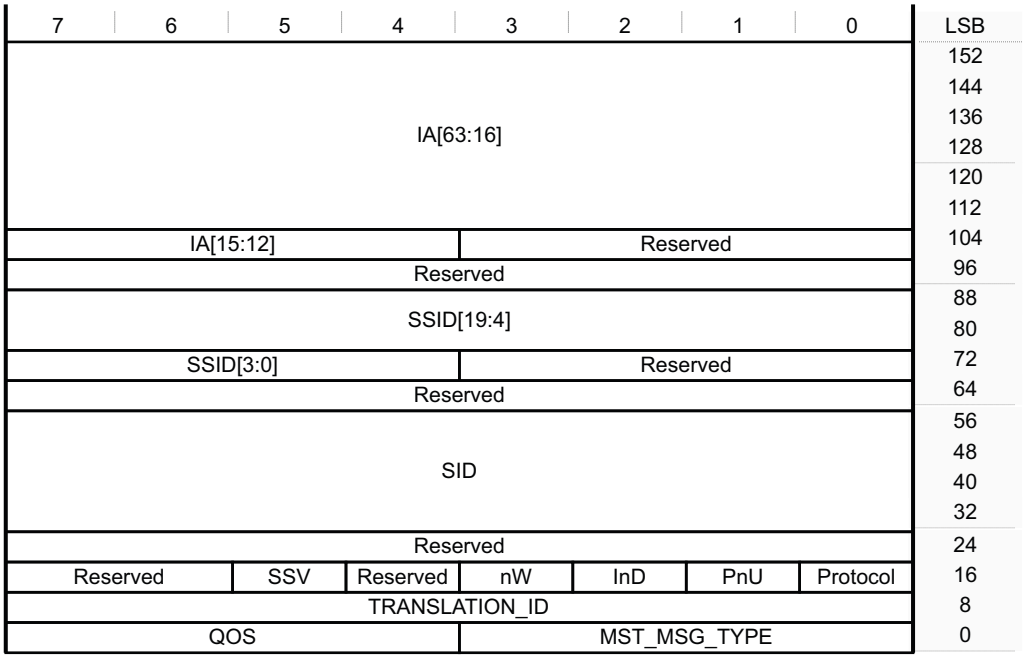
The PCIe RP must have at least one translation token.

Flow control result

The PCIe RP sends a translation token to the TCU.

Field descriptions

The DTI_ATS_TRANS_REQ bit assignments are:



IA, bits [159:108]

This field holds the input address, IA[63:12], to be used in the translation.

Bits [107:96]

Reserved, SBZ.

SSID, bits [95:76]

This field indicates the SubstreamID value that is used for the translation.

When the value of SSV is 0, this field is Reserved, SBZ.

Bits [75:64]

Reserved, SBZ.

SID, bits [63:32]

This field indicates the StreamID value that is used for the translation.

Bits [31:22]

Reserved, SBZ.

SSV, bit [21]

This bit indicates whether a valid SubstreamID is associated with this translation.

0 SSID not valid

1 SSID valid

Bit [20]

Reserved, SBZ.

nW, bit [19]

This bit indicates whether write access is requested.

0 Read and write access

1 Read-only access

When HTTU is enabled, a value of 0 in this field marks the page table entry as Dirty.

InD, bit [18]

This bit indicates whether execute (instruction) access is requested.

0 The translation will only be used for data accesses.

1 The translation might be used for instruction and data accesses.

When the value of SSV is 0, this bit must be 0.

PnU, bit [17]

This bit indicates whether this translation represents privileged or unprivileged access.

0 Unprivileged

1 Privileged

When the value of SSV is 0, this bit must be 0.

PROTOCOL, bit [16]

This bit indicates the protocol that is used for this message.

1 DTI-ATS

This bit must be 1

TRANSLATION_ID, bits [15:8]

This field gives the identification number for the translation.

The value of this field must not be in use by any translation request that has not yet received a DTI_ATS_TRANS_RESP or DTI_ATS_TRANS_FAULT response.

Any 8-bit translation ID can be used, provided that the maximum number of outstanding translation requests is not exceeded.

QOS, bits [7:4]

This field indicates the Quality of Service priority level. Translation requests with a high QOS value are likely to be responded to before requests with a lower QOS value.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for downstream messages, see *DTI-ATS protocol downstream messages* on page 2-26

0b0010 DTI_ATS_TRANS_REQ.

5.2.2 DTI_ATS_TRANS_RESP

The DTI_ATS_TRANS_RESP message is used respond to a translation request.

Description

A DTI translation response

Source

TCU

Usage constraints

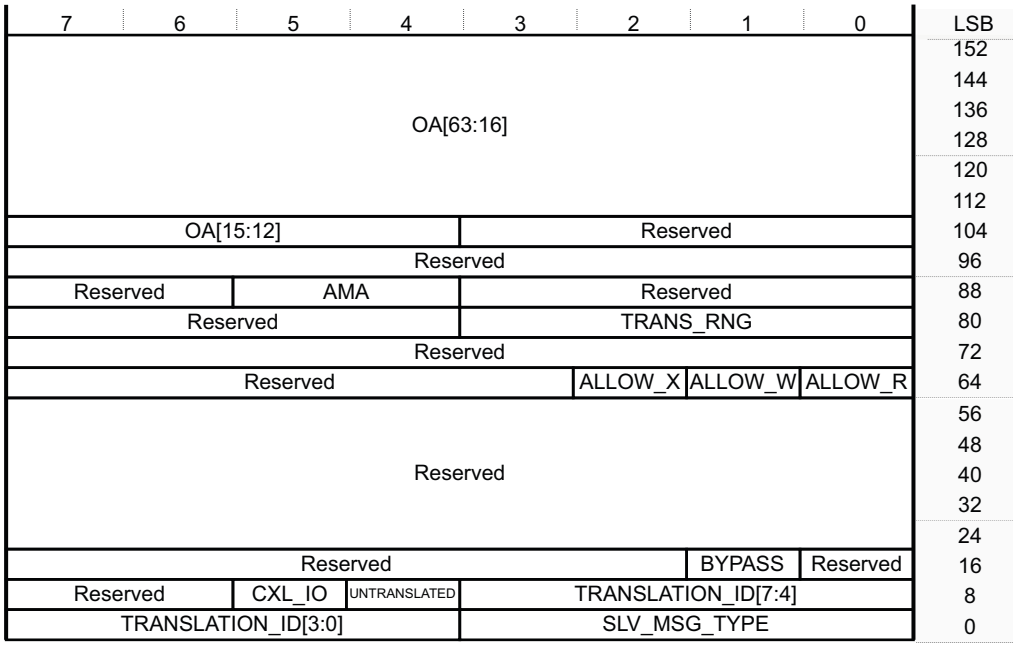
The PCIe RP must have previously issued a translation request that has not yet generated either a response or a fault message.

Flow control result

The TCU returns a translation token to the PCIe RP.

Field descriptions

The DTI_ATS_TRANS_RESP bit assignments are:



OA, bits [159:108]

This field holds the output address, OA[63:12], of the translated address.

DTI-ATSv1

The address in this field must be within the larger of the following address sizes:

- The size indicated by the OAS field of the DTI_ATS_CONDIS_ACK message received during the connection sequence.
- 40 bits.

This address must be to the first byte in a region of the size that is given by TRANS_RNG. For example, if the value of TRANS_RNG is 2, then OA[15:12] must be zero.

When BYPASS is 1, this field must be zero.

DTI-ATSv2

Bits within the range given by the TRANS_RNG field must match DTI_ATS_TRANS_REQ.IA.

For example, if the value of TRANS_RNG is 2, then OA[15:12] must equal DTI_ATS_TRANS_REQ.IA[15:12].

When the value of BYPASS is 1, this field must equal the value of IA in the translation request

When the value of UNTRANSLATED is 1, this field is Reserved, SBZ.

Bits [107:95]

Reserved, SBZ.

AMA, bits [94:92]

DTI-ATSv1

Reserved, SBZ.

DTI-ATSv2

This field indicates the translation attributes in a form that is designed for use by the PCIe ATS Memory Attributes field.

0b000	Normal-WB-RA-WA
0b001	Normal-WB-nRA-WA
0b010	Normal-WB-RA-nWA
0b011	Normal-WB-nRA-nWA
0b100	Device-nRnE
0b101	Device-nRE
0b110	Device-RE
0b111	Normal-NC

Bits [91:84]

Reserved, SBZ.

TRANS_RNG, bits [83:80]

The meaning of this field depends on the value of the DTI_ATS_TRANS_RESP.BYPASS field:

BYPASS=0

This field indicates the aligned range of addresses translation is valid for.

0b0000	4KB
0b0001	16KB
0b0010	64KB
0b0011	2MB
0b0100	32MB
0b0101	512MB
0b0110	1GB
00111b	16GB
0b1000	4TB
0b1001	128TB

All other values are Reserved.

BYPASS=1

DTI-ATSv1

This field indicates the maximum output address size of the system.

0b000 32 bits (4GB)

0b0001 36 bits (64GB)

0b0010 40 bits (1TB)

0b0011 42 bits (4TB)

0b0100 44 bits (16TB)

0b0101 48 bits (256TB)

0b0110 52 bits (4PB)

All other values are reserved.

This information is also given in the OAS field of the DTI_ATS_CONDIS_ACK message, and uses the same encodings. When BYPASS=1, this field must match DTI_ATS_CONDIS_ACK.OAS.

This value is a static property of the system, every transaction in which the value of the BYPASS field is 1 must return the same value for this field.

DTI-ATSv2

Reserved, SBZ.

Bits [79:67]

Reserved, SBZ.

ALLOW_X, bit [66]

This bit indicates permissions for instruction reads.

0 Not permitted

1 Permitted

When the value of ALLOW_R is 0, this bit must be 0.

When the value of InD in the DTI_ATS_TRANS_REQ translation request message was 0, this bit must be 0.

ALLOW_W, bit [65]

This bit indicates permissions for data write accesses.

0 Not permitted

1 Permitted

ALLOW_R, bit [64]

This bit indicates permissions for data read accesses.

0 Not permitted

1 Permitted

If the value of ALLOW_W is 0, the value of this field must be 1.

Bits [63:18]

Reserved, SBZ.

BYPASS, bit [17]

This field indicates that translation for this StreamID is bypassed.

0 Normal translation

1 Translation bypassed

When the value of this field is 1, the VA and the PA of the translation are the same.

This bit must be 0 if the value of IA in the translation request is greater than the range shown in the OAS field of the DTI_ATS_CONDIS_ACK message that was received during the connection sequence.

Bits [16:14]

Reserved, SBZ.

CXL_IO, bit [13]

DTI-ATSv1

Reserved, SBZ.

DTI-ATSv2

Used by root ports implementing CXL:

0 The translation response can be used by CXL.cache or CXL.io transactions.

1 The translation response cannot be used by CXL.cache transactions, and must only be used by CXL.io translated transactions.

When an ATS translation request is made with the Source_CXL bit set to 1, the CXL.io bit in the ATS response is the value of this field.

UNTRANSLATED, bit [12]

Indicates whether ATS translations should be used for this page.

0 The U bit in the PCIe ATS Translation Completion Data message must be 0.

1 The U bit in the PCIe ATS Translation Completion Data message must be 1.

This bit might be set when the TCU is not able to provide an ATS translation for the page.

For example, because of the memory attributes of the translated page.

When the value of this bit is 1, the PCIe Endpoint must access the page using untranslated transactions.

The ALLOW_R, ALLOW_W, and ALLOW_X values are unaffected by the value of this bit.

TRANSLATION_ID, bits [11:4]

This field gives the identification number for the translation.

This field must have a value corresponding to an outstanding translation request.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for upstream messages, see *DTI-ATS protocol upstream message* on page 2-26.

0b0010 DTI_ATS_TRANS_RESP

Mapping to PCIe Translation Completion Data Entry

When a DTI_ATS_TRANS_RESP message is received, the PCIe Translation Completion Data fields should be driven as follows:

Table 5-1 PCIe Translation Completion Data field mapping

Field	Value
Translated Address, S	Depends on the OA, TRANS_RNG and BYPASS fields of DTI_ATS_TRANS_RESP.
N	0b0
Global	0b0[a]
Exe	DTI_ATS_TRANS_RESP.ALLOW_X
Priv	DTI_ATS_TRANS_REQ.PnU

Table 5-1 PCIe Translation Completion Data field mapping (continued)

Field	Value
U	DTI_ATS_TRANS_RESP.UNTRANSLATED
R	DTI_ATS_TRANS_RESP.ALLOW_R
W	DTI_ATS_TRANS_RESP.ALLOW_W
CXL.io	If Source_CXL set in translation request: DTI_ATS_TRANS_RESP.CXL_IO Else: `0b0`

a. Previous versions of this specification included a GLOBAL field in DTI_ATS_TRANS_RESP. This was in error, since the SMMUv3 architecture requires the Global field in a Translation Completion to be 0.

5.2.3 DTI_ATS_TRANS_FAULT

The DTI_ATS_TRANS_FAULT message is used to provide a fault response to a translation request.

Description

A translation fault response

Source

TCU

Usage constraints

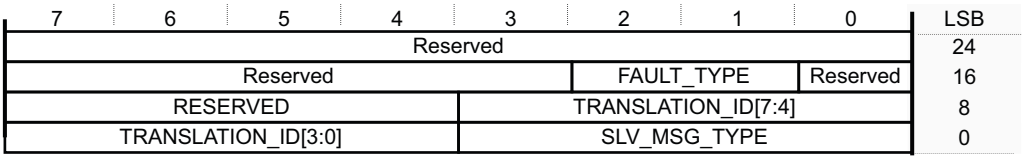
The PCIe RP must have previously issued a translation request that has not yet generated either a response or a fault message.

Flow control result

The TCU returns a translation token to the PCIe RP.

Field descriptions

The DTI_ATS_TRANS_FAULT bit assignments are:



Bits [31:19]

Reserved, SBZ.

FAULT_TYPE, bits [18:17]

This bit is used to tell the PCIe RP how to handle the fault.

`0b00` InvalidTranslation

`0b01` CompleterAbort

`0b10` UnsupportedRequest

`0b11` Reserved

When the value of this field is InvalidTranslation, this field indicates that ATS requests are permitted but that the translation resulted in a fault. The PCIe RP returns a Translation Completion message with the status value as Success and with the Read and Write bits clear.

When the value of this field is CompleterAbort, this field indicates that there was an error during the translation process. The PCIe RP returns a Translation Completion message with the status value as Completer Abort (CA).

When the value of this field is UnsupportedRequest, this field indicates that ATS is disabled for this or all StreamIDs. The PCIe RP returns a Translation Completion message with a status value as Unsupported Request (UR).

Bits [16:12]

Reserved, SBZ.

TRANSLATION_ID, bits [11:4]

This field gives the identification number for the translation.

This field must have a value corresponding to an outstanding translation request.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for upstream messages, see *DTI-ATS protocol upstream message* on page 2-26.

0b0001 DTI_ATS_TRANS_FAULT

5.2.4 The ATS translation sequence

A PCIe root complex must convert ATS translation requests from the PCIe world into DTI-ATS translation requests that the SMMU can respond to.

Figure 5-1 shows the steps required in a full ATS translation process that is supported by DTI.

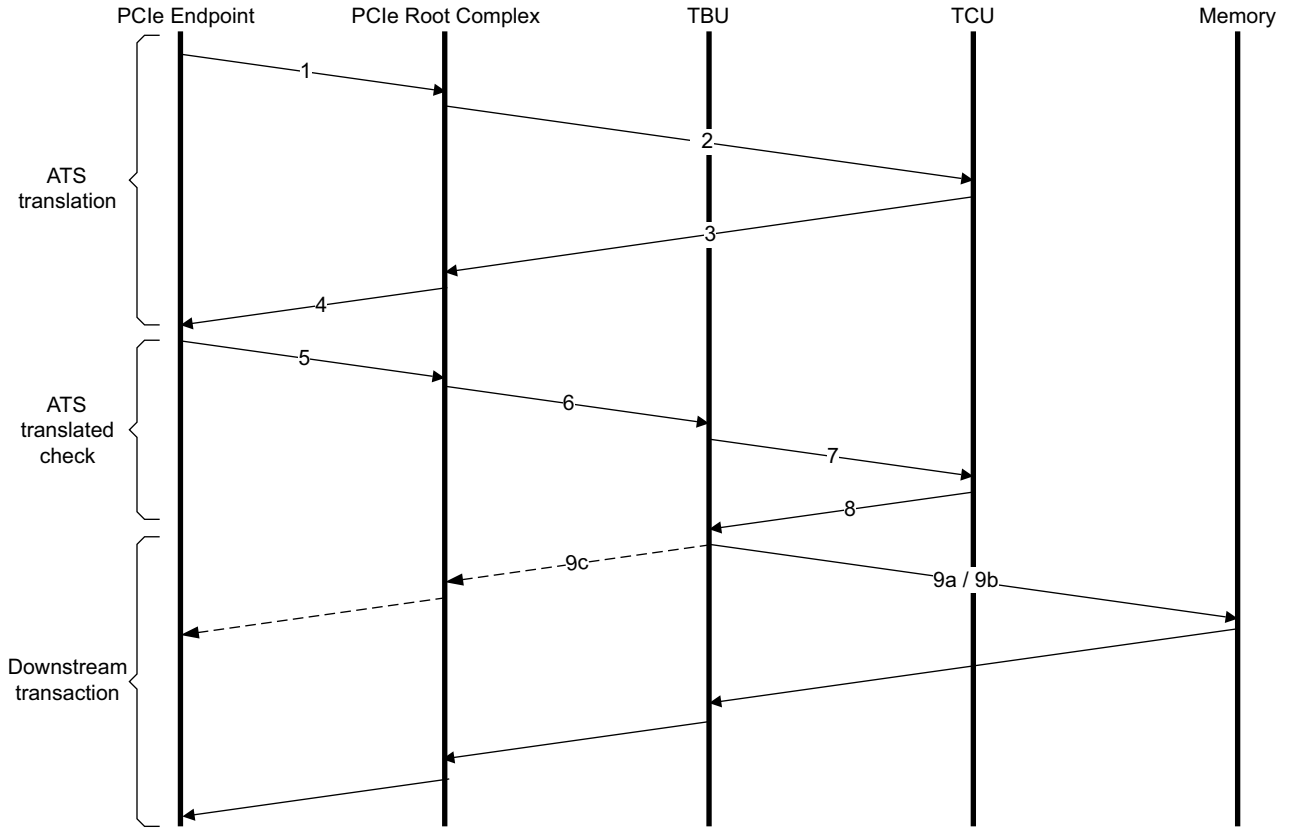


Figure 5-1 Example complete ATS translation sequence in DTI

The steps in Figure 5-1 are:

1. A PCIe Endpoint sends an ATS translation request to the Root Complex.
2. The Root Complex converts this to a DTI-ATS translation request and passes it to the TCU.
3. The TCU sends a DTI-ATS translation response to the Root Complex.
4. The Root Complex forwards the translation response to the Endpoint.
5. The Endpoint sends a translated transaction using the ATS translation.
6. The Root Complex sends this to a TBU, marked as ATS-translated.
7. The TBU, if it does not already have a suitable translation, sends a DTI-TBU translation request to the TCU.
8. The TCU sends a DTI-TBU translation response to the TBU.
9. The TBU handles the transaction, by either:
    a. Forwarding it downstream with the same address.
    b. Forwarding it downstream with additional stage 2 translation.
    c. Aborting the transaction if ATS is not supported for this stream.

The SMMU can be configured to:

- Prohibit ATS translation for individual streams. In this case, the TBU translation check prevents untrusted Endpoints from issuing physically addressed transactions into the system.
- Return stage 1 translation over ATS and perform stage 2 translation in the TBU. In this case, the TBU translation fetched in steps 7 and 8 perform stage 2 translation.
- Perform all translation using ATS. In this case, the TBU translation step is performed once to ensure that ATS is permitted for this stream, and can then be cached for all future transactions. This can be done per-stream or globally for all streams depending on the SMMU configuration.

Requests for multiple translations

Only one translation can be requested with each DTI_ATS_TRANS_REQ message. If a PCIe Root Complex receives an ATS translation request for multiple sequential pages then, it can either:

- Convert it into multiple individual DTI_ATS_TRANS_REQ messages and combine the responses.
- Convert it into a single DTI_ATS_TRANS_REQ message and respond with a single translation. This is legal behavior in PCIe ATS, in effect the Root Complex has denied the request to prefetch additional translations.

5.3 Invalidation and synchronization message group

This section describes the ATS invalidation and synchronization message group.

ATS Invalidation operations are passed to the PCIe Endpoints to invalidate their ATC.

Invalidation SYNC operations ensure that the invalidation and transactions associated with them are complete.

This section contains the following subsections:

5.3.1 DTI_ATS_INV_REQ

The DTI_ATS_INV_REQ message is used to request the invalidation of data that is stored in a cache.

Description

An invalidation request

Source

TCU

Usage constraints

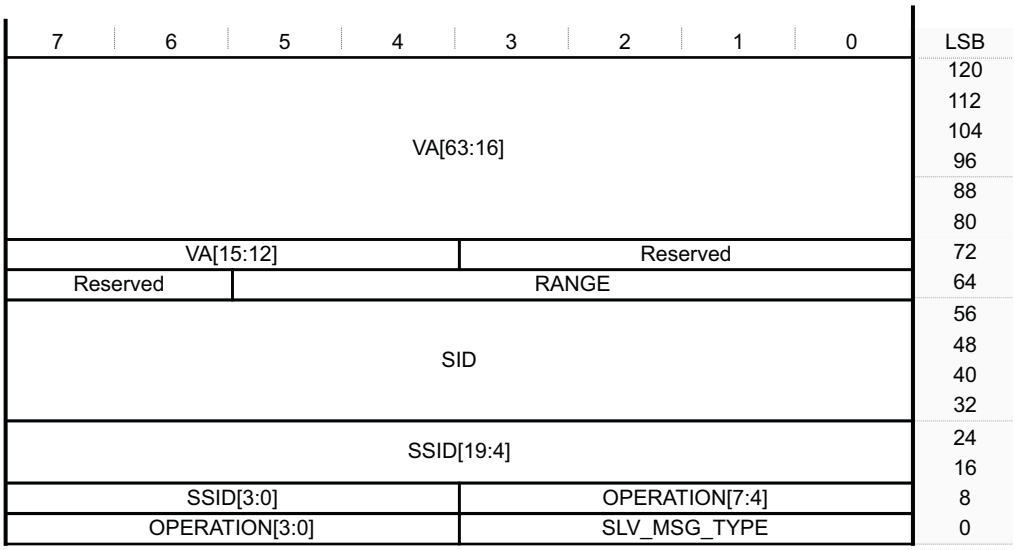
The TCU must have at least one invalidation token.

Flow control result

The TCU consumes an invalidation token.

Field descriptions

The DTI_ATS_INV_REQ bit assignments are:



VA, bits [127:76]

The Virtual Address or Intermediate Physical Address to be invalidated.

Bits [75:70]

Reserved, SBZ.

RANGE, bits [69:64]

This field identifies a range of Virtual Addresses for invalidation.

The range is calculated as 2^{RANGE} addresses, in multiples of 4KB pages. The bottom RANGE bits of the VA[63:12] field are ignored in this message, and the bottomRANGE bits of the IA[63:12] field are ignored in the translations being considered for invalidation. If RANGE is 52, all addresses are invalidated, and the VA field is ignored.

SID, bits [63:32]

This field indicates the StreamID to be invalidated.

The receiving PCIe RP must check to see if the value of this field is a StreamID that it uses. In the case that the StreamID is not used by this PCIe RP, the PCIe RP must acknowledge this message without performing an operation.

SSID, bits [31:12]

This field indicates the SubstreamID to be invalidated.

The encoding of the OPERATION field might cause this field to be invalid. When this field is invalid, it is reserved, SBZ.

OPERATION, bits [11:4]

This field identifies the type of invalidation operation being performed.

When a PCIe RP receives a message with an unrecognized OPERATION field value, this specification recommends that the PCIe RP acknowledges the invalidation without performing any operation.

The encoding of this field might cause other fields in this message to be invalid, for more information see *DTI-ATS invalidation operations* on page 5-109.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for upstream messages, see *DTI-ATS protocol upstream message* on page 2-26.

`0b1100` DTI_ATS_INV_REQ

5.3.2 DTI_ATS_INV_ACK

The DTI_ATS_INV_ACK message is used to acknowledge a cache invalidation request.

Description

A cache data invalidate acknowledgement

Source

PCIe RP

Usage constraints

The TCU must have previously issued an invalidation request that has not yet been acknowledged.

Flow control result

The PCIe RP returns an invalidation token to the TCU.

Field descriptions

The DTI_ATS_INV_ACK bit assignments are:

7	6	5	4	3	2	1	0	LSB
Reserved				MST_MSG_TYPE				0

Bits [7:4]

Reserved, SBZ.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for downstream messages, see *DTI-ATS protocol downstream messages* on page 2-26.

`0b1100` DTI_ATS_INV_ACK

5.3.3 DTI_ATS_SYNC_REQ

The DTI_ATS_SYNC_REQ message is used to request synchronization between the PCIe RP and TCU.

Description

A synchronization request

Source

TCU

Usage constraints

There must be no currently unacknowledged synchronization requests.

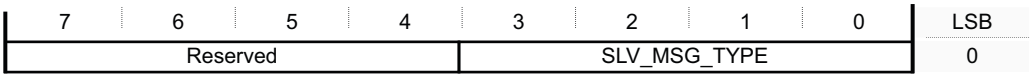
There must be no currently unacknowledged invalidation requests.

Flow control result

None.

Field descriptions

The DTI_ATS_SYNC_REQ bit assignments are:



Bits [7:4]

Reserved, SBZ.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for upstream messages, see *DTI-ATS protocol upstream message* on page 2-26.

`0b1101` DTI_ATS_SYNC_REQ

5.3.4 DTI_ATS_SYNC_ACK

The DTI_ATS_SYNC_ACK message is used to acknowledge a synchronization request.

Description

A synchronization acknowledge

Source

PCIe RP

Usage constraints

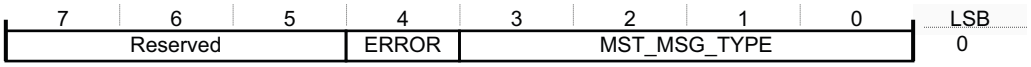
There must currently be an outstanding synchronization request.

Flow control result

None

Field descriptions

The DTI_ATS_SYNC_ACK bit assignments are:



Bits [7:5]

Reserved, SBZ.

ERROR, bit [4]

This bit indicates that a PCIe error has occurred.

0 Success

1 Error

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for downstream messages, see *DTI-ATS protocol downstream messages* on page 2-26.

0b1101 DTI_ATS_SYNC_ACK

5.3.5 The DTI-ATS invalidation sequence

ATS invalidation messages are used only to invalidate ATCs in a PCIe Endpoint. They are not used to invalidate TBU caches.

SMMUv3 requires that a TCU that intends to invalidate entries in an ATC must first invalidate the equivalent TBU entries. This results in an invalidation sequence shown in Figure 5-2 on page 5-108.

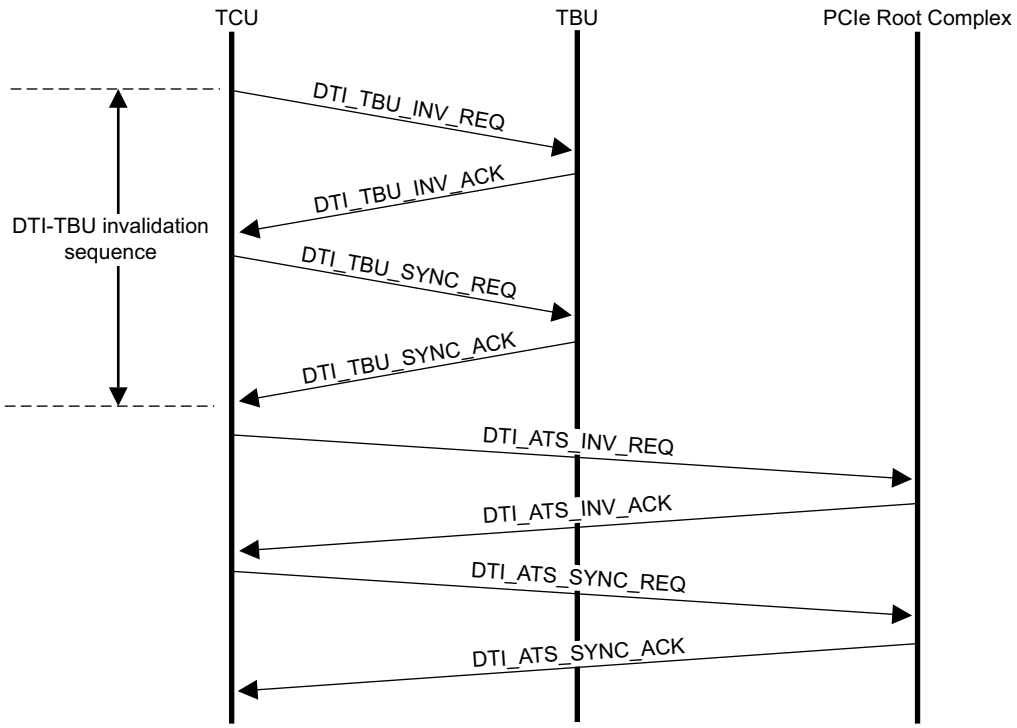


Figure 5-2 DTI-ATS invalidation sequence

The invalidation sequence in Figure 5-2 has the following steps:

1. The TCU issues a TLB invalidate operation to the TBU and waits for it to complete.
2. The TCU issues an invalidation synchronization operation to the TBU and waits for it to complete.
3. The TCU issues an ATS invalidation operation to the PCIe Root Complex and waits for it to complete.
4. The TCU issues an invalidation synchronization to the PCIe Root Complex and waits for it to complete.

The return of a DTI_ATS_SYNC_ACK message indicates that:

- Responses have been received from the appropriate Endpoints for DTI_ATS_INV_REQ messages that were received before the corresponding DTI_ATS_SYNC_REQ was received.
- No further accesses to memory are made using those translations, that is, transactions using those translations are complete.

Note

A DTI_ATS_SYNC_ACK message is likely to be dependent upon completion of outstanding translations in the downstream TBU. This does not cause deadlocks because SMMUv3 stalling faults are not permitted for PCIe RPs. This dependency is likely because DTI_ATS_SYNC_ACK is dependent on the Root Complex receiving invalidation completion messages from Endpoints, and those completion messages are ordered behind posted writes that might need translating.

Handling outstanding invalidations

PCIe requires that Endpoints support a minimum of 32 outstanding invalidation operations that must be accepted whether downstream transactions are able to make forward progress or not.

However, not all Endpoints can consume this number of invalidation operations without backpressure. And so, for performance reasons, the number of invalidate operations that should be outstanding in an Endpoint at one time might be less.

A PCIe RPs indicates in DTI_ATS_CONDIS_REQ.TOK_INV_GNT how many invalidation messages it can accept without giving backpressure on the DTI interface. It should buffer these locally so that the DTI interface is not stalled waiting for an Endpoint to progress an invalidation.

DTI-ATS invalidation tokens are only used for flow control of invalidation messages on the DTI channel. The Root Complex does not need to receive an Invalidation Completion message from an Endpoint before it returns a DTI_ATS_INV_ACK message on DTI-ATS. It can return a DTI_ATS_INV_ACK message as soon as it has successfully sent an Invalidation Request message to the Endpoint and is able to buffer a new DTI_ATS_INV_REQ message.

The Endpoint must return all Invalidation Completion messages before the Root Complex returns a DTI_ATS_SYNC_ACK message. If a new DTI_ATS_INV_REQ message is received after a DTI_ATS_SYNC_REQ, the Root Complex must do both of the following:

- Issue an Invalidation Request message to the Endpoint without waiting for the DTI_ATS_SYNC_ACK to be returned.
- Not wait for a corresponding Invalidation Completion message from the Endpoint for this invalidation before returning the currently outstanding DTI_ATS_SYNC_ACK message.

Ensuring downstream transaction completion

When an Endpoint returns an Invalidation Completion message, it guarantees that:

- All outstanding read requests that use the invalidated translations are complete.
- All posted write requests are pushed ahead of the Invalidation Completion message.

It does not guarantee that the posted write requests are complete, as memory writes in PCIe do not receive a response.

To ensure correct ordering, the Root Complex must ensure that posted writes intended for the AMBA system, that were received before the Invalidation Completion, have been issued downstream and are complete. A Root Complex can only return a DTI_ATS_SYNC_REQ message when this requirement has been met. The Root Complex is not required to ensure that reads are complete because this has already been ensured by the Endpoint.

5.3.6 DTI-ATS invalidation operations

This section gives information about the DTI-ATS cache invalidation operations.

Types of invalidation operation

The following table specifies the OPERATION field encodings and describes how the type of invalidation being performed affects the scope of the DTI_ATS_INV_REQ message. Other encodings of the OPERATION field are Reserved.

Table 5-2 List of invalidation operations

Field encoding	Invalidation operations	Substream Valid	Valid fields
0x31	ATCI_NOPASID	SSV = 0	SID, VA, RANGE
0x33	ATCI_PASID_GLOBAL	Global	SID, VA, RANGE
0x39	ATCI_PASID	SSV = 1	SID, SSID, VA, RANGE

Mapping DTI-ATS to SMMUv3 invalidate operations

DTI-ATS invalidation operations are generated as a result of commands in the SMMU command queue, the following table shows how these are mapped to DTI-ATS invalidate operations.

Table 5-3 Mapping DTI-ATS operation to SMMUv3 command

SMMUv3 Command	SSValid field value	Global field value	DTI-ATS Operation
CMD_ATC_INV	0	-	ATCI_NOPASID
CMD_ATC_INV	1	0	ATCI_PASID
CMD_ATC_INV	1	1	ATCI_PASID_GLOBAL

For more information, see the *Arm System MMUv3 (SMMUv3) Architecture Specification*.

5.4 Page request message group

The messages of this section enable a PCIe RPs to directly request software makes pages available. The messages of this group implement the PCIe ATS PRI.

The full details of the PCIe ATS PRI operations are not described here. For further information, see the PCIe Address Translation Service specification.

This section contains the following subsections:

5.4.1 DTI_ATS_PAGE_REQ

The DTI_ATS_PAGE_REQ message is used to request that a page is made available.

Description

A speculative page request

Source

PCIe RP

Usage constraints

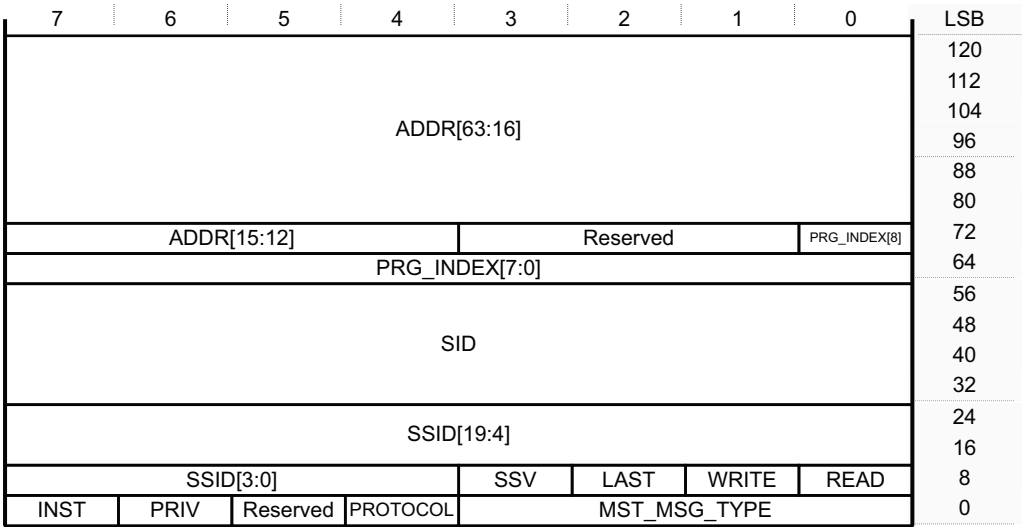
- There must be no current outstanding unacknowledged DTI_ATS_PAGE_REQ message.
- DTI_ATS_CONDIS_ACK.SUP_PRI was 1 during the connect sequence.

Flow control result

None

Field descriptions

The DTI_ATS_PAGE_REQ bit assignments are:



ADDR, bits [127:76]

This field holds the Page address[63:12] that is requested.

Bits [75:73]

Reserved, SBZ.

PRG_INDEX, bits [72:64]

This field identifies the Page Request group index.

SID, bits [63:32]

This field indicates the StreamID used for this transaction.

SSID, bits [31:12]

This field holds the SubstreamID used for this transaction.

If the value of SSV is 0, this field is reserved, SBZ.

SSV, bits [11]

This bit indicates whether a valid SubstreamID is associated with this transaction.

0 SSID not valid

1 SSID valid

LAST, bit [10]

This bit indicates whether this message is the last request in a page request group.

Note

The “Stop PASID” marker is indicated by SSV=1, LAST=1, READ=0, WRITE=0.

WRITE, bit [9]

This bit indicates whether write access is requested.

0 Write access is not requested

1 Write access is requested

A page request does not set the Dirty flag.

READ, bit [8]

This bit indicates whether read access is requested.

0 Read access is not requested

1 Read access is requested

INST, bit [7]

This bit indicates whether execute access is requested.

0 Execute access is not requested

1 Execute access is requested

If the value of READ is 0, the value of this bit must be 0.

PRIV, bit [6]

This bit indicates whether privileged access is requested.

0 Unprivileged

1 Privileged

Bit [5]

Reserved, SBZ.

PROTOCOL, bit [4]

This bit indicates the protocol that is used for this message.

1 DTI-ATS

This bit must be 1.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for downstream messages, see *DTI-ATS protocol downstream messages* on page 2-26.

0b1000 DTI_ATS_PAGE_REQ

5.4.2 DTI_ATS_PAGE_ACK

The DTI_ATS_PAGE_ACK message is used to acknowledge a page request.

Description

A page request acknowledgement

Source

TCU

Usage constraints

The PCIe RP must have previously issued a DTI_ATS_PAGE_REQ message that has not yet been acknowledged.

Flow control result

None

Field descriptions

The DTI_ATS_PAGE_ACK bit assignments are:

7	6	5	4	3	2	1	0	LSB
Reserved				SLV_MSG_TYPE				0

Bits [7:4]

Reserved, SBZ.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for upstream messages, see *DTI-ATS protocol upstream message* on page 2-26.

`0b1000` DTI_ATS_PAGE_ACK

5.4.3 DTI_ATS_PAGE_RESP

The DTI_ATS_PAGE_RESP message is used to respond to an ATS page request.

Description

An ATS page response

Source

TCU

Usage constraints

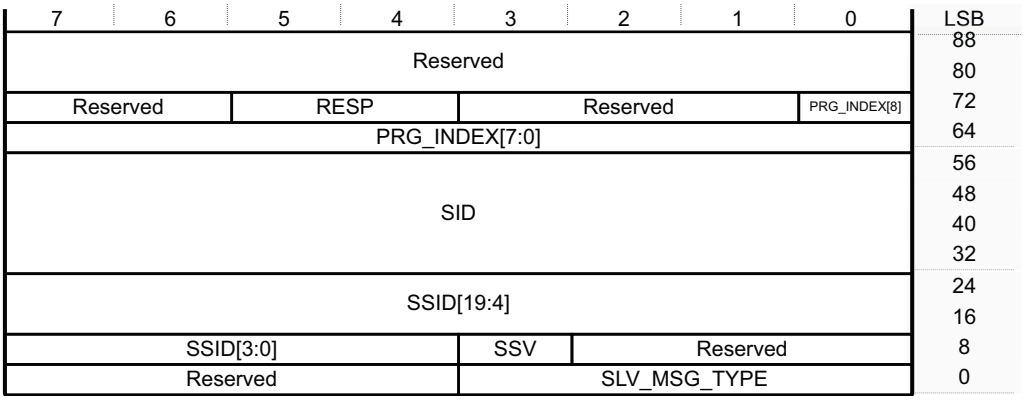
There must be no current outstanding unacknowledged DTI_ATS_PAGE_RESP message.

Flow control result

None

Field descriptions

The DTI_ATS_PAGE_RESP bit assignments are:



Bits [95:78]

Reserved, SBZ.

RESP, bits [77:76]

This field indicates the response code to the page request.

0b00 ResponseFailure

0b01 InvalidRequest

0b10 Success

0b11 Reserved

When the value of this field is ResponseFailure, a permanent error is indicated.

When the value of this field is InvalidRequest, the page-in was unsuccessful for at least one of the pages in the group.

When the value of this field is Success, the page-in was successful for all pages. This does not guarantee the success of a subsequent translation request to this page.

Bits [75:73]

Reserved, SBZ.

PRG_INDEX, bits [72:64]

This field holds the page request group index.

SID, bits [63:32]

This field holds the StreamID used for this page request.

The receiving TBU or PCIe RP must check to see if the value of this field is a StreamID that it uses. In the case that the StreamID is not used by this TBU or PCIe RP, the TBU or PCIe RP must ignore this message.

SSID, bits [31:12]

This field holds the SubstreamID used for this page request.

If the value of SSV is 0, this field is 0.

SSV, bits [11]

This bit indicates whether a valid SubstreamID is associated with this transaction.

0 SSID not valid

1 SSID valid

Bits [10:4]

Reserved, SBZ.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for upstream messages, see *DTI-ATS protocol upstream message* on page 2-26.

`0b1001` DTI_ATS_PAGE_RESP

5.4.4 DTI_ATS_PAGE_RESPACK

The DTI_ATS_PAGE_RESPACK message is used to acknowledge DTI_ATS_PAGE_RESP messages.

Description

Acknowledges DTI_ATS_PAGE_RESP messages

Source

PCIe RP

Usage constraints

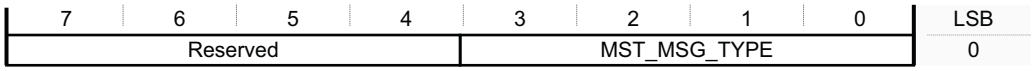
There must be at least one current outstanding unacknowledged DTI_ATS_PAGE_RESP message.

Flow control result

None

Field descriptions

The DTI_ATS_PAGE_RESPACK bit assignments are:



Bits [7:4]

Reserved, SBZ.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for downstream messages, see *DTI-ATS protocol downstream messages* on page 2-26.

`0b0100` DTI_ATS_PAGE_RESPACK

5.4.5 Generating the page response

If the DTI_ATS_PAGE_REQ was a PCIe PRI message, it is intended that it should result in a DTI_ATS_PAGE_RESP. However, the DTI_ATS_PAGE_RESP is generated by a software operation and cannot be guaranteed by the DTI protocol.

It is a software-level protocol error if a DTI_ATS_PAGE_RESP message with a StreamID used by the TBU or PCIe RP does not match an unanswered DTI_ATS_PAGE_REQ, when the value of LAST is 1, with the same PRG_INDEX value that is not a "Stop PASID" marker.

DTI_ATS_PAGE_RESP messages can be broadcast to all DTI_ATS TBU or PCIe RPs. As such, a DTI_ATS_PAGE_RESP message might be received with a StreamID that is not used by the TBU or PCIe RP and that does not match any of the StreamIDs from its unanswered DTI_ATS_PAGE_REQ messages.

Note

If a DTI_ATS_PAGE_RESP message is received with its RESP field as ResponseFailure, this requirement is suspended for the StreamID until the Page Request Interface can be re-enabled for that StreamID. For more information, see *PCI Express Address Translation Services Revision 1.1*.

Chapter 6
Transport Layer

This chapter describes the transport layer of the DTI protocol.

It contains the following sections:

6.1 Introduction

The DTI protocol can be conveyed over different transport layer mediums. This specification uses AXI4-Stream as an example transport medium.

The transport layer is responsible for:

- Indicating the source or destination of the message.
- Managing the link-level flow control.

The transport layer is not permitted to:

- Reorder the messages in the DTI protocol.
- Interleave messages in the DTI protocol.

6.2 AXI4-Stream transport protocol

This section defines the use of AXI4-Stream as a transport protocol.

This section contains the following subsections:

6.2.1 AXI4-Stream signals

An AXI4-Stream link for DTI consists of two AXI4-Stream interfaces, one for each direction.

The following table shows the mapping of AXI4-Stream signals for the DTI protocol.

Table 6-1 Mapping of AXI4-Stream to the DTI protocol

Signal	Usage	Notes
TVALID	Flow control	-
TREADY	Flow control	-
TDATA	Message data	Multi-cycle messages are permitted if the data is larger than the width of **TDATA**. A new message must always start on **TDATA[0]**.
TKEEP	Indicates valid bytes	Indicates which bytes contain valid data, with one bit for each byte of **TDATA**. Valid bytes must be packed towards the least-significant byte. The least significant byte must always be valid. All bytes must be valid if **TLAST** is LOW.
TSTRB	Not implemented	Uses default value of all bits equal to the corresponding bit of **TKEEP**.
TLAST	Last cycle of message	Each DTI message is transported as a number of AXI4-Stream transfers. This signal is used to indicate the last transfer of a message. Even if this interface is wide enough to carry all messages in a single cycle, this signal must be implemented.
TID	Originator node ID or not implemented	The meaning of this signal depends on the direction of the interface: • For a downstream interface, this signal indicates the source of the message. • For an upstream interface, this signal is not implemented. There is only one TCU in the network.
TDEST	Destination node ID or not implemented	The meaning of this signal depends on the direction of the interface: • For a downstream interface, this signal is not implemented. There is only one TCU in the network. • For an upstream interface, this signal indicates the desination of the message.
TUSER	Not implemented	The DTI protocol does not require this signal.

The signal names of the AXI4-Stream interface are given a suffix to indicate the direction of the interface they are using. The following table shows how the signals are suffixed.

Table 6-2 Suffixes appended to the AXI4-Stream signals

Direction	Suffix
Downstream (TBU or PCIe RP to TCU)	*_DTI_DN
Upstream (TCU to TBU or PCIe RP)	*_DTI_UP

For example, the downstream **TDATA** signal is **TDATA_DTI_DN**.

Components can add a further suffix to distinguish between multiple interfaces.

6.2.2 Interleaving

Message of the DTI protocol must not be interleaved when **TID** and **TDEST** are different. When an AXI4-Stream transfer is received with **TLAST** LOW, subsequent AXI4-Stream transfers must continue the same message with the same **TID** and **TDEST** until **TLAST** is HIGH. After **TLAST** is HIGH, a new message is permitted.

6.2.3 Usage of the TID and TDEST signals

In some cases a TBU or PCIe RP might not be aware of what value to use for the **TID** signal. This specification does not require the **TID** signal to be generated at the source. This specification recommends that:

- A TBU or PCIe RP interface does not implement the following:
  - TID_DTI_DN
  - TDEST_DTI_UP
- An interconnect that connects multiple DTI interfaces to a single TCU adds additional bits, as required, to the **TID** signal. The interconnect accepts messages from the TCU and redirects them to the appropriate component by IMPLEMENTATION DEFINED mapping of the **TID** signal.

This scheme can be extended to support hierarchical interconnects, with each layer of interconnect adding additional ID bits to the **TID** signal if necessary.

Appendix A
Pseudocode

This appendix provides example implementations of the requirements specified in this document.

The pseudocode language is as described in the *Arm Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

It contains the following section:

A.1 Memory attributes

This section details the decoding and processing of memory attributes in DTI.

This section contains the following subsections:

A.1.1 Memory attribute types

These types are used to describe propagating, modifying, combining and overriding memory attributes.

```
enumeration MemoryType {
         MemType_Normal,
         MemoryType_GRE,
         MemoryType_nGRE,
         MemoryType_nGnRE,
         MemoryType_nGnRnE
};

type MemAttrHints is (
         bits(2) attrs, // The possible encodings for each attributes field are as below
         bit ReadAllocate,
         bit WriteAllocate,
         bit Transient
)
constant bits(2) MemAttr_NC = '00'; // Non-cacheable
constant bits(2) MemAttr_WT = '10'; // Write-through
constant bits(2) MemAttr_WB = '11'; // Write-back

type MemoryAttributes is (
         MemoryType type,
         MemAttrHints inner, // Inner hints and attributes
         MemAttrHints outer, // Outer hints and attributes
         SH_e SH
)
```

A.1.2 Memory attribute decoding

These functions unpack encoded memory attributes from messages into their conceptual component properties.

MemAttrHintsDecode()

```
// MemAttrHintsDecode()
// ====================
// Converts the attribute fields for Normal memory as used in stage 2
// descriptors to orthogonal attributes and hints.
MemAttrHints MemAttrHintsDecode(bits(2) attr)

        MemAttrHints result;

        case attr of
                when '01' // Non-cacheable (no allocate)
                        result.attrs = MemAttr_NC;
                        result.ReadAllocate = '0';
                        result.WriteAllocate = '0';
                when '10' // Write-through
                        result.attrs = MemAttr_WT;
                        result.ReadAllocate = '1';
                        result.WriteAllocate = '1';
                when '11' // Write-back
                        result.attrs = MemAttr_WB;
                        result.ReadAllocate = '1';
                        result.WriteAllocate = '1';
                        result.Transient = '0';

        return result;
```

DecodeMemAttr()

```
// DecodeMemAttr()
// ===============
// Converts the MemAttr short-from field from stage 2 descriptors
// into the unpacked MemoryAttributes type.

MemoryAttributes DecodeMemAttr(bits(4) memattr)

        MemoryAttributes memattrs;
        if memattr<3:2> == '00' then // Device
                case memattr<1:0> of
                        when '00' memattrs.type = MemoryType_nGnRnE;
                        when '01' memattrs.type = MemoryType_nGnRE;
                        when '10' memattrs.type = MemoryType_nGRE;
                        when '11' memattrs.type = MemoryType_GRE;
                memattrs.inner = MemAttrHints UNKNOWN;
                memattrs.outer = MemAttrHints UNKNOWN;
                memattrs.SH = OuterShareable;

        elsif memattr<1:0> != '00' then // Normal
                memattrs.type = MemType_Normal;
                memattrs.outer = MemAttrHintsDecode(memattr<3:2>);
                memattrs.inner = MemAttrHintsDecode(memattr<1:0>);
                if (memattrs.inner.attrs == MemAttr_NC
                                && memattrs.outer.attrs == MemAttr_NC) then
                        memattrs.SH = OuterShareable;

        else
                // Unreachable
                assert(FALSE);

        return memattrs;
```

LongConvertAttrsHints()

```
// LongConvertAttrsHints()
// =======================
// Decodes the attribute fields for Normal memory as used in stage 1
// descriptors to orthogonal attributes and hints.
MemAttrHints LongConvertAttrsHints(bits(4) attrfield)
         MemAttrHints result;

         if attrfield<3:2> == '00' then // Write-through transient
                  result.attrs = MemAttr_WT;
                  result.ReadAllocate = attrfield<1>;
                  result.WriteAllocate = attrfield<0>;
                  result.Transient = '1';
         elsif attrfield<3:0> == '0100' then // Non-cacheable (no allocate)
                  result.attrs = MemAttr_NC;
                  result.ReadAllocate = '0';
                  result.WriteAllocate = '0';
                  result.Transient = '0';
         elsif attrfield<3:2> == '01' then // Write-back transient
                  result.attrs = MemAttr_WB;
                  result.ReadAllocate = attrfield<1>;
                  result.WriteAllocate = attrfield<0>;
                  result.Transient = '1';
         else // Write-through/Write-back non-transient
                  result.attrs = attrfield<3:2>;
                  result.ReadAllocate = attrfield<1>;
                  result.WriteAllocate = attrfield<0>;
                  result.Transient = '0';
         return result;
```

DecodeAttr()

```
// DecodeAttr()
// ============
// Converts the long-from ATTR field from stage 1 descriptors
// into the unpacked MemoryAttributes type.
MemoryAttributes DecodeAttr(bits(8) attrfield)
         MemoryAttributes memattrs;

         assert !(attrfield<7:4> != '0000' && attrfield<3:0> == '0000');
         assert !(attrfield<7:4> == '0000' && attrfield<3:0> != 'xx00');

         if attrfield<7:4> == '0000' then // Device
                  case attrfield<3:0> of
                           when '0000' memattrs.type = MemoryType_nGnRnE;
                           when '0100' memattrs.type = MemoryType_nGnRE;
                           when '1000' memattrs.type = MemoryType_nGRE;
                           when '1100' memattrs.type = MemoryType_GRE;
                  memattrs.inner = MemAttrHints UNKNOWN;
                  memattrs.outer = MemAttrHints UNKNOWN;
                  memattrs.SH = OuterShareable;

         elsif attrfield<3:0> != '0000' then // Normal
                  memattrs.type = MemType_Normal;
                  memattrs.outer = LongConvertAttrsHints(attrfield<7:4>);
                  memattrs.inner = LongConvertAttrsHints(attrfield<3:0>);

         return memattrs;
```

A.1.3 Memory attribute processing

This section details the procedures for combining memory type information.

DefaultMemAttrHints()

```
// DefaultMemAttrHints()
// =====================
// Populate MemoryAttribute sub-fields with default values that might be
// required later in combine/modify operations.
MemoryAttributes DefaultMemAttrHints(MemoryAttributes current_attr)

        if (current_attr.type != MemType_Normal
                        || current_attr.inner.attrs == MemAttr_NC) then
                current_attr.inner.ReadAllocate = '1';
                current_attr.inner.WriteAllocate = '1';
                current_attr.inner.Transient = '0';

        if (current_attr.type != MemType_Normal
                        || current_attr.outer.attrs == MemAttr_NC) then
                current_attr.outer.ReadAllocate = '1';
                current_attr.outer.WriteAllocate = '1';
                current_attr.outer.Transient = '0';

        return current_attr;
```

CombineMemoryType()

```
// CombineMemoryType()
// ===================
// Return the stronger of two memory types.

MemoryAttributes CombineMemoryType(MemoryAttributes attr_a, MemoryAttributes attr_b)

        if attr_a.type == MemoryType_nGnRnE || attr_b.type == MemoryType_nGnRnE then
                attr_a.type = MemoryType_nGnRnE;

        elsif attr_a.type == MemoryType_nGnRE || attr_b.type == MemoryType_nGnRE then
                attr_a.type = MemoryType_nGnRE;

        elsif attr_a.type == MemoryType_nGRE || attr_b.type == MemoryType_nGRE then
                attr_a.type = MemoryType_nGRE;

        elsif attr_a.type == MemoryType_GRE || attr_b.type == MemoryType_GRE then
                attr_a.type = MemoryType_GRE;

        else
                attr_a.type = MemType_Normal;
                attr_a.inner.attrs = (attr_a.inner.attrs AND attr_b.inner.attrs);
                attr_a.outer.attrs = (attr_a.outer.attrs AND attr_b.outer.attrs);

        return attr_a;
```

CombineShareability()

```
// CombineShareability()
// =====================
// Return the stronger of two shareability values.
SH_e CombineShareability(SH_e sh_a, SH_e sh_b)
        if sh_a == OuterShareable || sh_b == OuterShareable then
                return OuterShareable;
        elsif sh_a == InnerShareable || sh_b == InnerShareable then
                return InnerShareable;
        elsif sh_a == NonShareable || sh_b == NonShareable then
                return NonShareable;
```

CombineAllocHints()

```
// CombineAllocHints()
// ===================
// Return the stronger transient, read, and write allocation hints of
// two sets of memory attributes.

MemoryAttributes CombineAllocHints(MemoryAttributes attr_a, MemoryAttributes attr_b)

         // Combine the allocation hints. The strongest (encoded as 0) should take
         // precedence over the weakest (encoded as 1).
         attr_a.inner.WriteAllocate = (attr_a.inner.WriteAllocate AND attr_b.inner.
                  WriteAllocate);
         attr_a.inner.ReadAllocate = (attr_a.inner.ReadAllocate AND attr_b.inner.
                  ReadAllocate);
         attr_a.outer.WriteAllocate = (attr_a.outer.WriteAllocate AND attr_b.outer.
                  WriteAllocate);
         attr_a.outer.ReadAllocate = (attr_a.outer.ReadAllocate AND attr_b.outer.
                  ReadAllocate);

         // Combine the transient hints. The strongest (encoded as 1) should take
         // precedence over the weakest (encoded as 0).
         attr_a.inner.Transient = (attr_a.inner.Transient OR attr_b.inner.
                  Transient);
         attr_a.outer.Transient = (attr_a.outer.Transient OR attr_b.outer.
                  Transient);
         return attr_a;
```

ModifyShareability()

```
// ModifyShareability()
// ====================
// Override shareability using the SHCFG field.

MemoryAttributes ModifyShareability(MemoryAttributes current_attr, SHCFG_e shcfg)
         case shcfg of
                  when SHCFG_NonShareable
                           current_attr.SH = NonShareable;
                  when SHCFG_UseIncoming
                           current_attr.SH = current_attr.SH;
                  when SHCFG_OuterShareable
                           current_attr.SH = OuterShareable;
                  when SHCFG_InnerShareable
                           current_attr.SH = InnerShareable;
         return current_attr;
```

ReplaceMemoryType()

```
// ReplaceMemoryType()
// ===================
// Replace the memory type and Cacheability in the first parameter
// with that from the second parameter.

MemoryAttributes ReplaceMemoryType(MemoryAttributes current_attr, MemoryAttributes new_attr)
         current_attr.type = new_attr.type;
         current_attr.inner.attrs = new_attr.inner.attrs;
         current_attr.outer.attrs = new_attr.outer.attrs;

         return current_attr;
```

ReplaceAllocHints()

```
// ReplaceAllocHints()
// ===================
// Replace the allocation hints in the first parameter
// with that from the second parameter.

MemoryAttributes ReplaceAllocHints(MemoryAttributes current_attr, MemoryAttributes new_attr)
         current_attr.inner.ReadAllocate = new_attr.inner.ReadAllocate;
         current_attr.inner.WriteAllocate = new_attr.inner.WriteAllocate;
         current_attr.inner.Transient = new_attr.inner.Transient;
         current_attr.outer.ReadAllocate = new_attr.outer.ReadAllocate;
         current_attr.outer.WriteAllocate = new_attr.outer.WriteAllocate;
         current_attr.outer.Transient = new_attr.outer.Transient;

         return current_attr;
```