AMBA® DTI

Protocol Specification

arm

AMBA DTI
Protocol Specification

Copyright © 2016-2018, 2020-2024 Arm Limited or its affiliates. All rights reserved.

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
16 June 2021	E.b	Non-Confidential	Technical corrections
28 September 2023	F	Non-Confidential	**DTI-TBU protocol**: This issue is a standalone document for the DTI-TBUv3 protocol. Information on DTI-TBUv1 and DTI-TBUv2 is not included in this issue. For DTI-TBUv1 and DTI-TBUv2, see Arm Developer, https://developer.arm.com/documentation. **DTI-ATS protocol:** This issue describes DTI-ATSv1, DTI-ATSv2, and DTI-ATSv3 protocols.
28 June 2024	G	Non-Confidential	Support for DTI-TBUv4 and DTI-ATSv4, and other technical corrections.

Non-Confidential Proprietary Notice

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 or controlled, directly or indirectly, by You. A company shall be a Subsidiary only for the period during which such control exists.

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

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

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

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

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

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

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

ARM IHI 0088G
ID062824

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

PRE-21451 version 3

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

Chapter 3   **DTI-TBU Messages**

Chapter 4   **DTI-ATS Messages**

Preface

This preface introduces the *AMBA Distributed Translation Interface protocol specification*.

It contains the following sections:

About this specification

This specification describes the *AMBA Distributed Translation Interface (DTI) protocol.* It includes information on DTI messages, caching model, transport layer, and the pseudocode that describes various features of the DTI protocol.

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 specification 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-ATS Messages*

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

Chapter 5 *Transport Layer*

> This chapter describes the transport layer of the DTI protocol.

Chapter 6 *Pseudocode*

> This chapter provides example implementations of the requirements specified in this specification.

Appendix A *Revisions*

> Information about the technical changes between released issues of this specification.

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

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

Lowercase x At the second letter of a signal name denotes a collective term for both Read and Write.

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`. In both cases, the prefix and the associated value are written in a monospace font, for example, `0xFFFF0000`.

Additional reading

This section lists relevant publications from Arm. See Arm Developer, <https://developer.arm.com/documentation> for access to Arm documentation.

Arm publications

- *AMBA® LTI Protocol Specification* (ARM IHI 0089)
- *Arm® System Memory Management Unit Architecture Specification SMMU architecture version 3* (ARM IHI 0070)
- *AMBA® AXI-Stream Protocol Specification* (ARM IHI 0051)

Other publications

- *PCI Express Base Specification, Revision 6*, PCI-SIG
- *Compute Express Link Specification, Compute Express Link^{TM} Consortium, Inc., Revision 3*

Feedback on this specification

Arm welcomes feedback on its documentation.

If you have any comments or suggestions for additions and improvements, create a ticket at
https://support.developer.arm.com. As part of the ticket, please include:

*   The title, AMBA DTI Protocol Specification.
*   The number, ARM IHI 0088G.
*   The page number(s) that your comments apply.
*   A concise explanation of your comments.

─── **Note** ───

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of
any document when viewed with any other PDF reader.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive.
Arm strives to lead the industry and create change.

Previous issues of this document included terms that can be offensive. We have replaced these terms. If you find
offensive terms in this document, please contact terms@arm.com.

Chapter 1
Introduction

This chapter introduces the DTI protocol.

It contains the following sections:
* *About the DTI protocols* on page 1-18
* *DTI protocol specification terminology* on page 1-20

1.1 About the 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 provides translations for them. 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 referred to as the DTI protocol. The current versions of the DTI protocol are as follows:

DTI-TBUv1   Describes DTI-TBU version 1.

DTI-TBUv2   Describes DTI-TBU version 2.

DTI-TBUv3   Describes DTI-TBU version 3.

DTI-TBUv4   Describes DTI-TBU version 4.

DTI-ATSv1   Describes DTI-ATS version 1.

DTI-ATSv2   Describes DTI-ATS version 2.

DTI-ATSv3   Describes DTI-ATS version 3.

DTI-ATSv4   Describes DTI-ATS version 4.

———— **Note** ————

This specification does not describe DTI-TBUv1 and DTI-TBUv2. For information on these versions, see Arm Developer, https://developer.arm.com/documentation.

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 shows an example SMMU system that implements DTI.
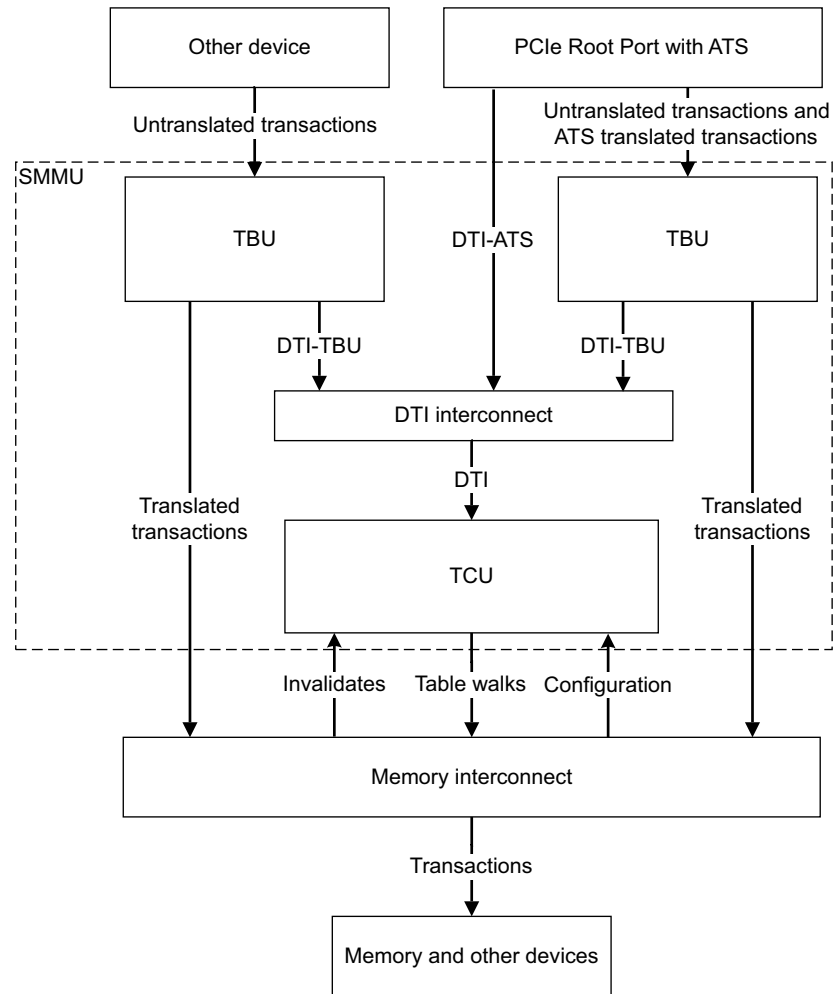
Figure 1-1 An example SMMU system

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 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.

DTI-ATSv2

> Describes characteristics of DTI-ATS version 2.

DTI-ATSv3

> Describes characteristics of DTI-ATS version 3.

DTI-ATSv4

> Describes characteristics of DTI-ATS version 4.

DTI-TBUv1

> Describes characteristics of DTI-TBU version 1.

DTI-TBUv2

> Describes characteristics of DTI-TBU version 2.

DTI-TBUv3

> Describes characteristics of DTI-TBU version 3.

DTI-TBUv4

> Describes characteristics of DTI-TBU version 4.

E2H

> EL2 Host mode. The Virtualization Host Extensions, introduced in the *Arm Architecture Reference Manual for A-profile architecture, 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 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.

ARM IHI 0088G
ID062824

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. Performs Granule Protection Checks (GPC).	Retrieves an ATS translation. Performs GPC.
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 message is 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

Table 2-2 shows the downstream messages of the DTI-TBU protocol.

Table 2-2 DTI-TBU protocol downstream messages

Message group	Message	M_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

Table 2-3 shows the upstream messages of the DTI-TBU protocol.

Table 2-3 DTI-TBU protocol upstream messages

Message group	Message	S_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
	DTI_TBU_TRANS_RESPEX	0x3	192
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

Table 2-4 shows the downstream messages of the DTI-ATS protocol.

Table 2-4 DTI-ATS protocol downstream message

Message group	Message	M_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_INV_COMP[a]	0xB	96
	DTI_ATS_SYNC_ACK	0xD	8
Page request	DTI_ATS_PAGE_REQ	0x8	128
	DTI_ATS_PAGE_RESPACK[b]	0x9	8
IMPLEMENTATION DEFINED	-	0xE	-
	-	0xF	-

a. DTI-ATSv3 or later

b. DTI-ATSv2 or later

DTI-ATS protocol upstream message

Table 2-5 shows the upstream messages of the DTI-ATS protocol.

Table 2-5 DTI-ATS protocol upstream messages

Message group	Message	S_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 messages to limit the number of outstanding invalidation requests.

Request messages consume tokens and response messages return them. See *Flow control result* section of respective message. 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 when the width of DTI interface is narrower than width of the response message, then the ID can only be reused when the complete message has been received. If a request message has multiple response messages associated with it, then the ID can only be reused when the final response message has been received. More details can be found in the *Flow control result* section of each message.

--- **Note** ---

The only response message that is not the final response message is DTI_TBU_TRANS_FAULT with FAULT_TYPE = TranslationStall.

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. It is recommended 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 Reserved encodings

When a field is not Reserved but it has Reserved encodings, it is a protocol error to use a Reserved encoding.

2.1.6 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.

2.1.7 Ignored fields

If a message has Ignored fields, the sender can drive any value to these fields, and the recipient must ignore these fields.

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. shows how the channel state changes in response to connect and disconnect messages.
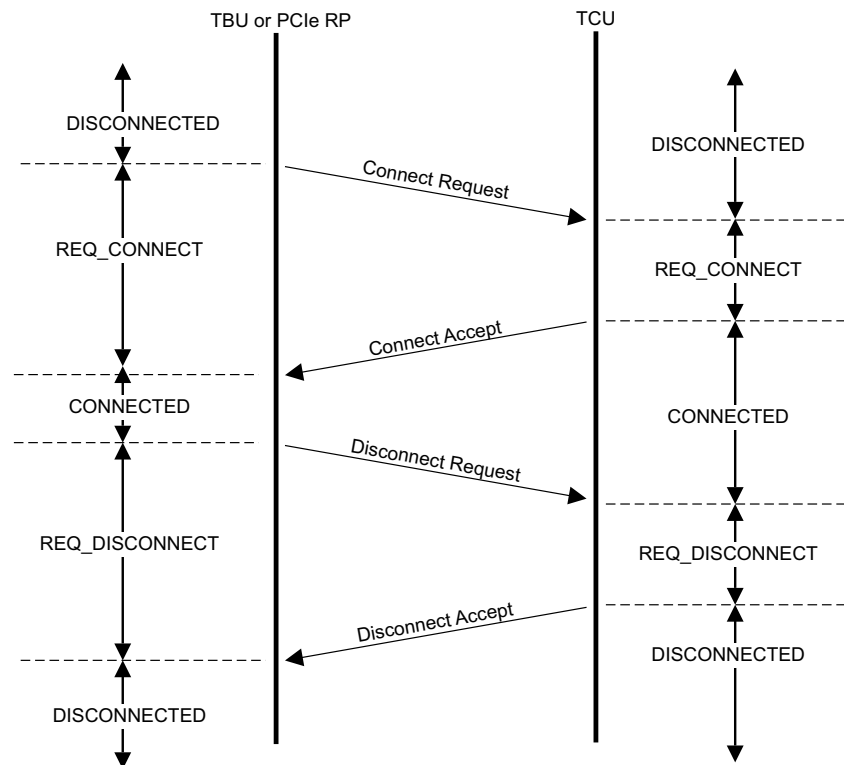
Figure 2-1 Handshake accept

Alternatively, a Connect Request might be denied, as shown in Figure 2-2.
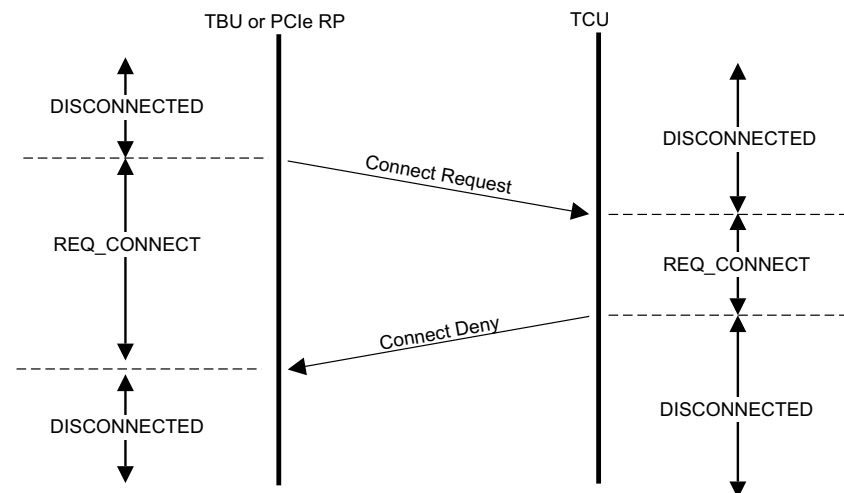
Figure 2-2 Handshake deny

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.

Table 2-6 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, it is expected that all TBUs connected to a TCU use the same version of DTI-TBU. This is because the SMMU architecture does not permit TBUs in the same implementation to have different feature sets.

- If using SMMUv3.2, it is required that all TBUs support DTI-TBUv2 or later.
- If using SMMU extensions for RME or RME Device Assignment, it is required that all TBUs support DTI-TBUv3 or later.

This section contains the following subsections:

- *DTI_TBU_CONDIS_REQ*
- *DTI_TBU_CONDIS_ACK* on page 3-34

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]				STAGES		SPD	SUP_REG	24
TOK_INV_GNT				TOK_TRANS_REQ[7:4]				16
TOK_TRANS_REQ[3:0]				VERSION				8
IMP DEF	Reserved	PROTOCOL	STATE	M_MSG_TYPE				0

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

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

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.

STAGES, bits [27:26]

This field indicates the security stages. STAGES[1:0] is encoded as follows:

0b00   M: SMMUv3 defined translation stages only

0b01   MG: SMMUv3 defined translation stages, plus Granule Protection Checks (GPC)

0b10   G: GPC only

0b11   Reserved

When STATE is 0, this field is ignored.

SPD, bit [25]

Same Power Domain (SPD). This extension is micro-architectural to make it easier to integrate power control.

0   The TBU and TCU are in different power domains.

1   The TBU and TCU are in the same power domain.

When STATE is 0, this field is ignored.

———— **Note** ————

This field is Reserved in versions prior to DTI-TBUv3 and will be ignored by TCUs that do not support DTI-TBUv3 and DTI-TBUv4.

————————————

SUP_REG, bit [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[11:8], bits [31:28].

VERSION, bits [11:8]

This field identifies the requested protocol version.

0b0000   DTI-TBUv1

0b0001   DTI-TBUv2

0b0010   DTI-TBUv3

0b0011   DTI-TBUv4

All other encodings are for future protocol versions and are currently not defined.

———— **Note** ————

This specification describes only DTI-TBUv3 and DTI-TBUv4. For information on DTI-TBUv1 and DTI-TBUv2, see Arm Developer, https://developer.arm.com/documentation.

————————————

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.

M_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.

0b0000   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 acknowledgment.

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	S_MSG_TYPE				0

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

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

This field indicates the number of preallocated tokens for translation requests that have been granted. The number of translation tokens granted is equal to the value of this field plus one.

When the value of STATE is 1, the value of this field must equal the value of the

TOK_TRANS_REQ field in the DTI_TBU_CONDIS_REQ message that initiated the connection.

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.

The TBU must ensure that the output address does not exceed the OAS.

Bit [20]

Reserved, SBZ

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

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

VERSION, bits [11:8]

The protocol version that is granted by the TCU.

0b0000	DTI-TBUv1
0b0001	DTI-TBUv2
0b0010	DTI-TBUv3
0b0011	DTI-TBUv4

All other encodings are Reserved.

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.

——— **Note** ———

This specification describes only DTI-TBUv3 and DTI-TBUv4. For information on DTI-TBUv1 and DTI-TBUv2, see Arm Developer, https://developer.arm.com/documentation.

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 where value 0 denies the connection and value 1 accepts the connection.

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

S_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.

———— **Note** ————

Unless the description indicates otherwise, behavior and reference to DTI_TBU_TRANS_RESP and DTI_TBU_TRANS_RESPEX messages are equivalent.

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:

7	6	5	4	3	2	1	0	LSB
								152
								144
								136
				IA				128
								120
								112
								104
								96
			SSID[19:4]					88
								80
	SSID[3:0]				IMP DEF			72
FLOW[1]	Reserved	MMUV	REQEX		Reserved			64
								56
								48
			SID					40
								32
	TRANSLATION_ID[11:8]			IDENT	SEC_SID[1]	NSE	NS	24
PERM[1]	FLOW[0]	SSV	SEC_SID[0]	PERM[0]	InD	PnU	PROTOCOL	16
			TRANSLATION_ID[7: 0]					8
	QOS				M_MSG_TYPE			0

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.

When MMUV is 0, SSID 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 FLOW = ATST, it indicates that this transaction was the result of a previous ATS translation request made using DTI-ATS.
>
> **0b10** NoStall
>
> If a translation fault occurs, then even if the SMMU has enabled stall faulting for this translation context, a fault response is returned without dependence on software activity.
>
> **0b11** PRI
>
> If a translation fault occurs, a fault response is returned indicating that a PRI request might resolve the fault. Architecturally, the request is treated as an ATS request and translation faults do not result in an event record. This option is for use by PCIe enumerated endpoints.
>
> PRI requests must be sent using a DTI-ATS connection. There is no mechanism to issue a PRI request from a DTI-TBU connection.

―――― **Note** ――――

If FLOW = PRI and PERM = SPEC, then translation faults are reported as NonAbort. For more information, see FAULT_TYPE field in *DTI_TBU_TRANS_FAULT* on page 3-57.

―――――――――――――

When MMUV is 0, FLOW is Reserved, SBZ.

Bit [70]

Reserved, SBZ

MMUV, bit [69]

When MMUV is 0, no SMMU stage 1 or 2 checking is performed, only the Realm Management Extension (RME) Granule Protection Check is performed.
When DTI_TBU_CONDIS_REQ.STAGES is G, MMUV must be 0.
When DTI_TBU_CONDIS_REQ.STAGES is M, MMUV must be 1.

―――― **Note** ――――

When MMUV = 0, a translation can be requested where REQ.IA exceeds DTI_TBU_CONDIS_ACK.OAS. The TCU must return a fault in this case, similar to the behavior with BYPASS responses.

―――――――――――――

Rules dependent on the value of fields that are Reserved when MMUV = 0, are valid only when MMUV = 1.

REQEX, bit [68]

This field controls whether the TCU can return a DTI_TBU_TRANS_RESPEX response.
When REQEX = 0, the translation response cannot be DTI_TBU_TRANS_RESPEX.
When REQEX = 1, the translation response can be DTI_TBU_TRANS_RESPEX.
The response is never required by DTI to be DTI_TBU_TRANS_RESPEX. It can always be DTI_TBU_TRANS_RESP or DTI_TBU_TRANS_FAULT.

Bits [67:64]

Reserved, SBZ

SID, bits [63:32]

This field indicates the StreamID value that is used for the translation.
When MMUV is 0, SID is Reserved, SBZ.

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 with FAULT_TYPE != TranslationStall response.
Any 12-bit translation ID can be used, if the maximum number of outstanding translation requests is not exceeded.

IDENT, bit [27]

This field indicates whether an identity translation is required.
When IDENT is 1, DTI_TBU_TRANS_RESP.OA must always be equal to IA.
The encodings of IDENT are as follows:

0   Identity translation is not required.

1   Identity translation is required.

When IDENT is 1, FLOW must be ATST.
When MMUV is 0, IDENT is Reserved, SBZ.

SEC_SID[1], bit [26]

This field indicates the Security states of StreamID. The encodings of SEC_SID[1:0] are as follows:

`0b00`	Non-secure
`0b01`	Secure
`0b10`	Realm
`0b11`	Reserved

When DTI_TBU_CONDIS_REQ.STAGES is M, SEC_SID must be Non-secure or Secure.

When MMUV is 0, SEC_SID is Reserved, SBZ.

If FLOW = ATST, SEC_SID must be Non-secure or Realm.

NSE, bit [25]

{NSE, NS} indicates the physical address space of the untranslated transaction. For more information, see NS, bit [24].

NS, bit [24]

{NSE, NS} indicates the physical address space of the untranslated transaction. The encodings of {NSE, NS} are as follows:

`0b00`	Secure
`0b01`	Non-secure
`0b10`	Root
`0b11`	Realm

When MMUV is 1 and SEC_SID is Non-secure, {NSE,NS} must be Non-secure.
When MMUV is 1 and SEC_SID is Secure, {NSE,NS} must be Non-secure or Secure.
When MMUV is 1 and SEC_SID is Realm, {NSE,NS} must be Non-secure or Realm.

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.

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.

When MMUV is 0, SSV is Reserved, SBZ.

SEC_SID[0], bit [20]

To see the encodings of SEC_SID[1:0], please refer to SEC_SID[1], bit [26].

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.

When MMUV is 0, InD is Reserved, SBZ.

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.

When MMUV is 0, PnU is Reserved, SBZ.

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[11:8], 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 the requests with a lower QOS value.

This field is a hint.

M_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 3-59.

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 with FAULT_TYPE != TranslationStall.

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				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
HWATTR				PARTID[9]	Reserved	MPAMNSE	NSE	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_MT	ASET	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]				S_MSG_TYPE				0

IMPLEMENTATION DEFINED, bits [159:156]

IMPLEMENTATION DEFINED

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

MPAM PARTID[3:0]

When DTI_TBU_TRANS_REQ.MMUV is 0, PARTID must be 0.

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

MPAM PARTID[7:4].

When DTI_TBU_TRANS_REQ.MMUV is 0, PARTID must be 0.

OA, bits [147:108]

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

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]

MPAM PARTID[8].

When DTI_TBU_TRANS_REQ.MMUV is 0, PARTID must be 0.

PMG, bit [106]

MPAM PMG.

When DTI_TBU_TRANS_REQ.MMUV is 0, PMG must be 0.

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 translation 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 3-64.

─────────────

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 [99:96]

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)
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.

HWATTR, bits [95:92]

This field gives IMPLEMENTATION DEFINED hardware attributes from the translation 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.

When DTI_TBU_TRANS_REQ.MMUV is 0, HWATTR is Reserved, SBZ.

Bit [91]

> **DTI-TBUv3**
>
> > Reserved, SBZ
>
> **DTI-TBUv4**
>
> > MPAM PARTID[9]
> > When DTI_TBU_TRANS_REQ.MMUV is 0, PARTID must be 0.

Bit [90]

> Reserved, SBZ

MPAMNSE, bit [89]

> {MPAMNSE, MPAMNS} indicates the PARTID space. For more information, see MPAMNS, bit [73].

NSE, bit [88]

> {NSE, NS} indicates the physical address space of the translated transaction. For more information, see NS, bit [70].

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 |
> | 0b1010 | 64GB |
> | 0b1011 | 512GB |
> | 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 translation 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.
>
> 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.

TRANS_RNG, bits [83:80]

> The meaning of TRANS_RNG does not depend on BYPASS.
>
> > This field indicates the aligned range of addresses that this translation is valid for:
> >
> > | | |
> > |---|---|
> > | 0b0000 | 4KB |
> > | 0b0001 | 16KB |

0b0010	64KB
0b0011	2MB
0b0100	32MB
0b0101	512MB
0b0110	1GB
0b0111	16GB
0b1010	64GB
0b1011	512GB
0b1000	4TB
0b1111	The full address range provided in DTI_TBU_CONDIS_ACK.OAS.

All other values are Reserved.

When BYPASS is 1 and BP_TYPE is DPTBypass, TRANS_RNG must not be 0b1111.

When BYPASS is 0, TRANS_RNG must not be 0b1111.
When DTI_TBU_CONDIS_REQ.STAGES is M and BYPASS is 1, then TRANS_RNG must be 0b1111.

A cache entry matches future transactions irrespective of input PAS if TRANS_RNG == 0b1111, and BP_TYPE == GlobalBypass.

A cache entry matches future transactions with the same input PAS as DTI_TBU_TRANS_REQ.{NSE,NS} if either of the following is true:

- TRANS_RNG != 0b1111
- BYPASS == 1 and BP_TYPE == StreamBypass

This field must not be greater than the size indicated by the OAS field of the DTI_TBU_CONDIS_ACK message received during the connecting sequence. For example, if the value of the OAS field is 4GB, this field must indicate a range of 1GB or less.

TRANSLATION_ID [11:8], bits [79:76]

This field gives the identification number for the translation. This field must have a value corresponding to an outstanding translation request.

COMB_ALLOC, bit [75]

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 3-60.

COMB_SH, bit [74]

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 3-60.

MPAMNS, bit [73]

{MPAMNSE, MPAMNS} indicates the PARTID space. The encodings of {MPAMNSE, MPAMNS} are as follows:

0b00   Secure

0b01   Non-secure

`0b10`	Root	
`0b11`	Realm	

When DTI_TBU_TRANS_REQ.MMUV is 0, {MPAMNSE,MPAMNS} must match DTI_TBU_TRANS_REQ.{NSE, NS}.

When DTI_TBU_TRANS_REQ.MMUV is 1 and DTI_TBU_TRANS_REQ.SEC_SID is Non-secure, {MPAMNSE,MPAMNS} must be Non-secure.

When DTI_TBU_TRANS_REQ.MMUV is 1 and DTI_TBU_TRANS_REQ.SEC_SID is Secure, {MPAMNSE,MPAMNS} must be Non-secure or Secure.

When DTI_TBU_TRANS_REQ.MMUV is 1 and DTI_TBU_TRANS_REQ.SEC_SID is Realm, {MPAMNSE,MPAMNS} must be Non-secure or Realm.

For more information, see MPAMNSE, bit [89].

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 translation 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 physical address space (PAS) to be used for downstream transactions. For more information, see NSE, bit [88].

The encodings of {NSE, NS} are as follows:

`0b00`	Secure
`0b01`	Non-secure
`0b10`	Root
`0b11`	Realm

When DTI_TBU_TRANS_REQ.MMUV is 1, DTI_TBU_TRANS_REQ.SEC_SID is Non-secure, {NSE, NS} must be Non-secure.

When DTI_TBU_TRANS_REQ.MMUV is 1, DTI_TBU_TRANS_REQ.SEC_SID is Secure, {NSE, NS} must be Non-secure or Secure.

When DTI_TBU_TRANS_REQ.MMUV is 1 and DTI_TBU_TRANS_REQ.SEC_SID is Realm, {NSE, NS} must be Non-secure or Realm.

When DTI_TBU_TRANS_REQ.MMUV is 1, DTI_TBU_TRANS_REQ.SEC_SID is Secure, and BYPASS is 1:

*   When NSCFG is Use Incoming, {NSE,NS} must equal DTI_TBU_TRANS_REQ.{NSE,NS}.
*   When NSCFG is Secure, {NSE,NS} must be Secure.
*   When NSCFG is Non-secure, {NSE,NS} must be Non-secure.

When DTI_TBU_TRANS_REQ.MMUV is 1 and DTI_TBU_TRANS_REQ.SEC_SID is Realm and BYPASS is 1 and BP_TYPE is StreamBypass:

*   When NSCFG = Use incoming, {NSE,NS} must equal DTI_TBU_TRANS_REQ.{NSE,NS}.

- When NSCFG = Realm, {NSE,NS} must be Realm.

- When NSCFG = Non-secure, {NSE,NS} must be Non-secure.

When DTI_TBU_TRANS_REQ.MMUV is 0, then {NSE,NS} must match DTI_TBU_TRANS_REQ.{NSE,NS}.

───── **Note** ─────

When DTI_TBU_TRANS_REQ.MMUV is 1 and DTI_TBU_TRANS_REQ.SEC_SID is Realm and BYPASS is 1 and BP_TYPE is DPTBypass, {NSE,NS} is not affected by NSCFG.

─────────────

ALLOW_PX, bit [69] when BYPASS = 0

This bit indicates permissions for privileged instruction reads.

0   Not permitted

1   Permitted

It is expected that TCU should drive this field to 1 if the permission is granted by the translation.

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 Non-secure or Realm, this field is Reserved, SBZ.

When DTI_TBU_TRANS_REQ.MMUV is 0, ALLOW_NSX is Reserved, SBZ.

It is expected that TCU should drive this field to 1 if the permission is granted by the translation.

ALLOW_PW, bit [68]

This bit indicates permissions for privileged data write accesses.

0   Not permitted

1   Permitted

DTI-TBUv3

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

DTI-TBUv4

- When BYPASS is 1 and BP_TYPE != DPTBypass, this field is Reserved, SBZ.

- When BYPASS is 1 and BP_TYPE = DPTBypass, ALLOW_UW must be equal to the value of ALLOW_PW.

It is expected that TCU should drive this field to 1 if the permission is granted by the translation.

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.

It is expected that TCU should drive this field to 1 if the permission is granted by the translation.

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.

It is expected that TCU should drive this field to 1 if the permission is granted by the translation.

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.

DTI-TBUv3

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

DTI-TBUv4

- When BYPASS is 1 and BP_TYPE != DPTBypass, this field is Reserved, SBZ.

- When BYPASS is 1 and BP_TYPE = DPTBypass, ALLOW_UW must be equal to the value of ALLOW_PW.

It is expected that TCU should drive this field to 1 if the permission is granted by the translation.

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.

It is expected that TCU should drive this field to 1 if the permission is granted by the translation.

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, ATTR_OVR.MTCFG must be 0 and ATTR_OVR.SHCFG must be 0b01. 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 3-60.

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

When DTI_TBU_TRANS_REQ.MMUV is 0, ATTR_OVR is Reserved, SBZ. The corresponding fields all behave according to the Use Incoming encoding.

VMID, bits [47:32]

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

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 3-60.

When DTI_TBU_TRANS_REQ.MMUV is 0, ALLOCCFG is Reserved, SBZ. The corresponding fields all behave according to the Use Incoming encoding.

COMB_MT, bit [27]

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 3-60.

ASET, bit [26]

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.

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

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.

When DTI_TBU_TRANS_REQ.MMUV is 0, INSTCFG is Reserved, SBZ. The corresponding fields all behave according to the Use Incoming encoding.

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.

When DTI_TBU_TRANS_REQ.MMUV is 0, PRIVCFG is Reserved, SBZ. The corresponding fields all behave according to the Use Incoming encoding.

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.

DTI-TBUv3

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

DTI-TBUv4

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

- BP_TYPE is GlobalBypass.

- BP_TYPE is StreamBypass and DTI_TBU_TRANS_REQ.FLOW != ATST.

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 or ReadOnce transaction. There is no communication with the TCU to indicate that this conversion has occurred.

DTI-TBUv3

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

DTI-TBUv4

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

- BP_TYPE is GlobalBypass.

- BP_TYPE is StreamBypass and DTI_TBU_TRANS_REQ.FLOW != ATST.

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 DTI_TBU_TRANS_REQ.SEC_SID and DTI_TBU_TRANS_REQ.FLOW fields in the translation request:

- When DTI_TBU_TRANS_REQ.SEC_SID is Non-secure or Realm, this field is not permitted to be EL3.

- When the value of DTI_TBU_TRANS_REQ.FLOW is ATST, this field must be EL1-S2.

- When the value of DTI_TBU_TRANS_REQ.SSV is 1, this field must not be EL1-S2.

- When DTI_TBU_CONDIS_REQ.STAGES is MG, STRW must not be EL3.

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

This field has the following encodings:

- **0b00**:
  — DTI-TBUv3: Reserved
  — DTI-TBUv4: DPTBypass
    — Only permitted if DTI_TBU_TRANS_REQ.SEC_SID == Realm and DTI_TBU_TRANS_REQ.FLOW == ATST.
    — The translation can be used for future transactions with the same values of DTI_TBU_TRANS_REQ.SEC_SID, DTI_TBU_TRANS_REQ.SID, and DTI_TBU_TRANS_REQ.FLOW == ATST.

- **0b01**: GlobalBypass:
  — Not permitted if DTI_TBU_TRANS_REQ.SEC_SID == Realm.
  — The translation can be used for future transactions with the same values of DTI_TBU_TRANS_REQ.SEC_SID and DTI_TBU_TRANS_REQ.FLOW == ATST.

- **0b10**: StreamBypass:
  — Not permitted when DTI_TBU_TRANS_REQ.SSV == 1. The translation can be used for future transactions with the same values of DTI_TBU_TRANS_REQ.SEC_SID, DTI_TBU_TRANS_REQ.SID, DTI_TBU_TRANS_REQ.SSV, and DTI_TBU_TRANS_REQ.FLOW == ATST.

- **0b11**: Reserved

All Bypass translations are subject to the address range defined in TRANS_RNG.

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	MMUV	SEC_SID	FLOW = ATST	SID	SSV	SSID
GlobalBypass with MMUV = 0	Yes	No	No	No	No	No
GlobalBypass with MMUV = 1	Yes	Yes	Yes	No	No	No
StreamBypass	Yes - always 1	Yes	Yes	Yes	Yes - always 0	No
DPTBypass	Yes - always 1	Yes - always Realm	Yes - always 1	Yes	No	No

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 for the corresponding security level.
- A translation is requested with MMUV set to LOW.

When DTI_TBU_TRANS_REQ.MMUV is 0, BP_TYPE must be GlobalBypass.

Summary of permitted translation response contexts

For each translation response, one of the following fields is valid that gives a context for the response:

- DTI_TBU_TRANS_RESP.STRW
- DTI_TBU_TRANS_RESP.BPTYPE
- DTI_TBU_TRANS_FAULT.FAULT_TYPE

Table 3-3, Table 3-4 on page 3-53, and Table 3-5 on page 3-53 describe which of the cacheable response contexts are permitted depending on the value of DTI_TBU_TRANS_REQ.SEC_SID and whether DTI_TBU_TRANS_REQ.FLOW == ATST.

When DTI_TBU_CONDIS_REQ.STAGES == MG && DTI_TBU_CONDIS_ACK.VERSION == DTI-TBUv3:

Table 3-3 Summary of DTI-TBUv3 permitted translation response contexts when DTI_TBU_CONDIS_REQ.STAGES = MG

SEC_SID	Non-secure	Secure	Realm	Non-secure	Realm
FLOW	Not ATST	Not ATST	Not ATST	ATST	ATST
EL1	Yes	Yes	Yes	-	-
EL1-S2	Yes	Yes	Yes	Yes	Yes
EL2	Yes	Yes	Yes	-	-
StreamBypass	Yes	Yes	Yes	Yes	Yes
StreamDisabled	Yes	Yes	Yes	Yes	Yes
GlobalBypass	Yes	Yes	-	Yes	-
GlobalDisabled	Yes	Yes	Yes	-	-

When DTI_TBU_CONDIS_REQ.STAGES == MG &&
DTI_TBU_CONDIS_ACK.VERSION == DTI-TBUv4:

**Table 3-4 Summary of DTI-TBUv4 permitted translation response contexts when
DTI_TBU_CONDIS_REQ.STAGES = MG**

SEC_SID	Non-secure	Secure	Realm	Non-secure	Realm
FLOW	Not ATST	Not ATST	Not ATST	ATST	ATST
EL1	Yes	Yes	Yes	-	-
EL1-S2	Yes	Yes	Yes	Yes	Yes
EL2	Yes	Yes	Yes	-	-
DPTBypass	-	-	-	-	Yes
StreamBypass	Yes	Yes	Yes	Yes	Yes
StreamDisabled	Yes	Yes	Yes	Yes	Yes
GlobalBypass	Yes	Yes	-	Yes	-
GlobalDisabled	Yes	Yes	Yes	-	-

When DTI_TBU_CONDIS_REQ.STAGES == M:

**Table 3-5 Summary of permitted translation response contexts when
DTI_TBU_CONDIS_REQ.STAGES = M**

SEC_SID	Non-secure	Secure	Non-secure
FLOW	Not ATST	Not ATST	ATST
EL1	Yes	Yes	-
EL1-S2	Yes	Yes	Yes
EL2	Yes	Yes	-
EL3	-	Yes	-
StreamBypass	Yes	Yes	Yes
StreamDisabled	Yes	Yes	Yes
GlobalBypass	Yes	Yes	Yes
GlobalDisabled	Yes	Yes	-

Device Permission Table support

The Device Permission Table (DPT) and associated behavior provides a mechanism to
enforce the association between granules of Physical Address Space and the memory
footprint of virtual machines. This means that the physical memory pages of a virtual
machine can be protected from accesses by a malicious virtual machine when both
virtual machines are required for DPT check. DTI-TBUv4 only supports DPT for Realm
state although the SMMUv3 architecture supports an independent DPT for each of
Non-secure and Realm states.

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.MMUV is 0, BYPASS must be 1.

When DTI_TBU_TRANS_REQ.IDENT is 1, BYPASS must be 1.

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 nonzero, 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.

———— **Note** ————

This field is not the same as the CONT bit in the translation table entry which affects the range of address this translation applies to which is reflected in TRANS_RNG. This field affects the range of StreamIDs this translation applies to. In another word, this field affects the number of translation contexts rather than the size of memory region.

————————————

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.

————————————

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

See TRANSLATION_ID [11:8], bits [79:76].

S_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

When DTI_TBU_TRANS_REQ.MMUV is 1 and DTI_TBU_TRANS_REQ.SEC_SID is Secure, the TBU uses DTI_TBU_TRANS_REQ.NS and DTI_TBU_TRANS_RESP.ATTR_OVR.NSCFG to determine whether the translation is for a Secure IPA or Non-secure IPA.

This information is used when determining the scope of invalidation operations. NSCFG is not used by cache lookups during translation.

Table 3-6 shows the information required for invalidation operations.

Table 3-6 Determination of IPA space in DTI-TBU

DTI_TBU_TRANS_REQ.{NSE,NS}	NSCFG	IPA space
Secure	Use Incoming	Secure
Non-secure	Use Incoming	Non-secure
-	Secure	Secure
-	Non-secure	Non-secure

If DTI_TBU_TRANS_REQ.SEC_SID is Realm, the IPA space is always Realm.
If DTI_TBU_TRANS_REQ.SEC_SID is Non-secure, the IPA space is always Non-secure.

3.2.3 DTI_TBU_TRANS_RESPEX

DTI_TBU_TRANS_RESPEX encapsulates DTI_TBU_TRANS_RESP message and adds new fields.

Description

A DTI translation response with extended message length.

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 with FAULT_TYPE != TranslationStall.

Flow control result

The TCU returns a translation token to the TBU.

Field descriptions

The DTI_TBU_TRANS_RESPEX bit assignments are:

7	6	5	4	3	2	1	0	LSB
			Reserved					184
		Reserved				PARTID[11:10]		176
			MECID					168
								160
								152
								144
								136
								128
								120
								112
			DTI_TBU_TRANS_RESP					104
								96
								88
								80
								72
								64
								56
								48
								40
								32
								24
								16
								8
	DTI_TBU_TRANS_RESP				S_MSG_TYPE			0

Bits [191:178]

> Reserved, SBZ

Bits [177:176]

> **DTI-TBUv3**
>
> > Reserved, SBZ
>
> **DTI-TBUv4**
>
> > PARTID [11:10].
> >
> > When DTI_TBU_TRANS_REQ.MMUV is 0, PARTID must be 0.
> >
> > If DTI_TBU_TRANS_REQ.REQEX = 1 and DTI_TBU_TRANS_RESP is returned, it is equivalent to a DTI_TBU_TRANS_RESPEX with all bits 0 in [177:176].

> ──── **Note** ────
>
> If an implementation uses more than 10 bits of PARTID, it must always set DTI_TBU_TRANS_REQ.REQEX and the TCU is recommended to always return DTI_TBU_TRANS_RESPEX.
> ────────

MECID, bits [175:160]

> *Memory Encryption Context Identifier* (MECID) to support Realm Management Extension (RME) and RME-Device Assignment (RME-DA).
>
> When DTI_TBU_TRANS_REQ.SEC_SID != Realm, MECID must be 0.
>
> When the translated output PAS is Non-secure, a MECID of 0 must be used for translated transaction regardless of the MECID value returned in the translation response.
>
> When DTI_TBU_TRANS_REQ.MMUV = 0, MECID must be 0.
>
> If DTI_TBU_TRANS_REQ.REQEX = 1 and DTI_TBU_TRANS_RESP is returned, it is equivalent to a DTI_TBU_TRANS_RESPEX with all bits 0 in [175:160].

Bits [159:4]

> Same as DTI_TBU_TRANS_RESP. See *DTI_TBU_TRANS_RESP* on page 3-41.

S_MSG_TYPE, bits [3:0]

> This field identifies the message type. The value of this field is 0x3.

———— **Note** ————

All protocol rules for DTI_TBU_TRANS_RESP apply to DTI_TBU_TRANS_RESPEX unless stated otherwise.

3.2.4   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 if FAULT_TYPE != TranslationStall.

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]				S_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.

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 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

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

0b101 TranslationStall

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 and DTI_TBU_TRANS_REQ.PERM != SPEC. 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.

0b110 Reserved

0b111 Reserved

When DTI_TBU_TRANS_REQ.MMUV is 0, FAULT_TYPE must be NonAbort if DTI_TBU_TRANS_REQ.PERM is SPEC, and Abort otherwise.

—— **Note** ——

A TBU implementation might have mechanisms to re-transmit the translation request for the same transaction after it has received a translation response. If it receives a DTI_TBU_TRANS_FAULT message with FAULT_TYPE != TranslationStall and DO_NOT_CACHE = 1, then the TBU is not expected to re-transmit the translation request again in order to avoid the possibility of multiple event reports for the same transaction.

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 nonzero, 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].

S_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

3.2.5   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 a TCU receives a translation request with DTI_TBU_TRANS_REQ.IA[55:52] != `0x0` and DTI_TBU_TRANS_REQ.IA[55:52] != `0xF`, then the TCU must complete the translation with a DTI_TBU_TRANS_FAULT message.

*   If a TCU receives a translation request with DTI_TBU_TRANS_REQ.IA[63:52] != `0x000` and DTI_TBU_TRANS_REQ.IA[63:52] != `0xFFF`, 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.

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.

Though these rules were not specified in the DTI-TBUv1 specification, they 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 (denoted as resp) when permission is granted for the transaction that is described in the translation request (denoted as req). It is a protocol error if PermissionCheck(req, resp) is False.

In addition to the requested permissions in the translation request, TCU is expected to return all permissions granted by the translation. This not only avoids unnecessary misses in TBU cache lookup by future transactions but also it might be relied upon by some transactions to function correctly, such as transactions related to cache maintenance or cache stash.

3.2.6 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 for A-profile architecture*. 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 3-7.

Table 3-7 ATTR_OVR subfields

Field bits	Field name
[3:0]	MemAttr
[4]	MTCFG
[6:5]	SHCFG
[8:7]	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 3-61.

3. The allocation hints are modified based on the value of ALLOCCFG. For more information, see *The ALLOCCFG field* on page 3-62.

4. The Shareability domain is modified based on the value of SHCFG. For more information, see *The SHCFG field* on page 3-63.

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 3-63.

6. A consistency check is applied to eliminate illegal attribute combinations. For more information, see *Consistency check on combination of translation attributes* on page 3-64.

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 Destructive Hint or 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 3-8 shows the encoding of this field:

Table 3-8 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 3-8 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-130 for an example implementation.

The NSCFG field

NSCFG indicates when the NS bit of the incoming transaction is overridden before translation.

When DTI_TBU_TRANS_REQ.SEC_SID == Secure, the encodings of NSCFG are:

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

When DTI_TBU_TRANS_REQ.SEC_SID == Realm, the encodings of NSCFG are:

`0b00`	Use incoming
`0b01`	Reserved
`0b10`	Realm
`0b11`	Non-secure

If DTI_TBU_TRANS_REQ.MMUV == 1 and DTI_TBU_TRANS_REQ.SEC_SID == Non-secure, DTI_TBU_TRANS_RESP.ATTR_OVR.NSCFG is Reserved, SBZ.

When DTI_TBU_TRANS_REQ.MMUV is 1 and DTI_TBU_TRANS_REQ.SEC_SID is Realm and BYPASS is 1 and BP_TYPE is DPTBypass, then NSCFG is Reserved, SBZ.

Combining the translation response attributes

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

```
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.COMB_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 3-9:

Table 3-9 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-130 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.

3.2.7 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.

3.2.8 Cache lookup process

When there is a 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.

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

See *MatchTranslation* on page A-136 and *MatchFault* on page A-137 for precise cache lookup matching functions.

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.

This section contains the following subsections:

- *DTI_TBU_INV_REQ*
- *DTI_TBU_INV_ACK* on page 3-69
- *DTI_TBU_SYNC_REQ* on page 3-69
- *DTI_TBU_SYNC_ACK* on page 3-70
- *DTI-TBU invalidation sequence* on page 3-70
- *DTI-TBU invalidation operations* on page 3-73

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
								120
								112
			ADDR[63:16]					104
								96
								88
								80
ADDR[15:12]				Reserved				72
SCALE[5]	OPERATION[8]	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] or SIZE[3:2]	TTL or SSID[1:0] or SIZE[1:0]		OPERATION[7:4]					8
OPERATION[3:0]			S_MSG_TYPE					0

ADDR, bits [127:76]

This field indicates the address to be invalidated. The address can be VA[63:12], IPA[51:12], or PA[51:12] where ADDR[63:52] must be 0 if it represents IPA or PA.

The encoding of the OPERATION field might cause this field to be invalid. When no OPERATION is using this field, it is Reserved, SBZ.

Bits [75:72]

Reserved, SBZ

SCALE[5], bit [71]

Extends the SCALE field to 6 bits to enable larger ranges to be specified for invalidation. See SCALE[4:0], bits [25:21].

OPERATION[8], bit [70]

See OPERATION[7:0], bits [11:4].

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 apart from TLBI_PA. 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
- TLBI_RL_EL1_ALL
- TLBI_RL_EL1_S1_VMID
- TLBI_RL_EL1_S12_VMID
- TLBI_RL_EL1_VAA
- TLBI_RL_EL1_S2_IPA
- TLBI_RL_EL2_ALL
- TLBI_RL_EL2_VAA

RANGE, bits [68:64]

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

When the value of the OPERATION field identifies this message as a CFGI_SID invalidate operation, the bottom RANGE number of 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 number of 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 no OPERATION is using this field, 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 no OPERATION is using this field, 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 no OPERATION is using this field, it is Reserved, SBZ.

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 no OPERATION is using this field, 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 no OPERATION is using this field, it is Reserved, SBZ.

SCALE[4:0], bits [25:21], when OPERATION is a TLB invalidate operation.

This field relates to Range invalidate operations. The encoding of the OPERATION field might cause this to be invalid. When no OPERATION is using this field, it is Reserved, SBZ. For more information, see *DTI-TBU invalidation operations* on page 3-73.

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

This field relates to Range invalidate operations. The encoding of the OPERATION field might cause this field to be invalid. When no OPERATION is using this field, it is Reserved, SBZ. For more information, see *DTI-TBU invalidation operations* on page 3-73.

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

This field relates to Range invalidate operations. The encoding of the OPERATION field might cause this field to be invalid. When no OPERATION is using this field, it is Reserved, SBZ. For more information, see *DTI-TBU invalidation operations* on page 3-73.

SIZE, bits [15:12], when OPERATION is a TLBI_PA or DPT invalidate operation.

The SIZE field overlaps with **TG**, **TTL**, and **SSID** fields.
This field relates to GPC invalidate operations. The encoding of the OPERATION field might cause this field to be invalid. When no OPERATION is using this field, it is Reserved, SBZ. For more information, see *GPC invalidate operations* on page 3-79.

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

This field relates to Range invalidate operations. The encoding of the OPERATION field might cause this field to be invalid. When no OPERATION is using this field, it is Reserved, SBZ. For more information, see *DTI-TBU invalidation operations* on page 3-73.

OPERATION[7:0], 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, it is recommended 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 3-73.

S_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 acknowledgment.

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:

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

Bits [7:4]

Reserved, SBZ

M_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.

─────── **Note** ───────

It is legal to receive the message even when there are no prior invalidation requests to synchronize.

Flow control result

None

Field descriptions

The DTI_TBU_SYNC_REQ bit assignments are:

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

Bits [7:4]

Reserved, SBZ

S_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				M_MSG_TYPE				0

Bits [7:4]

Reserved, SBZ

M_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 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.
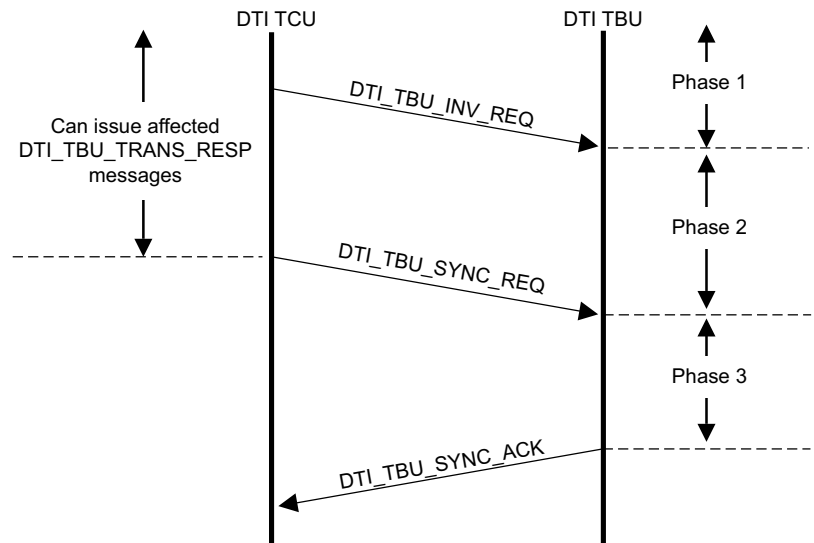
Figure 3-1 Phases of the invalidation sequence

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.

—— **Note** ——
If a transaction receives a DTI_TBU_TRANS_FAULT message with FAULT_TYPE != TranslationStall, then it is considered as having 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. Table 3-10 describes the phases and required actions.

Table 3-10 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 on invalidated data, then it is 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

To avoid deadlocks, the following rules must be followed for DTI-TBU:

- 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 3-11 specifies the OPERATION field encodings for DTI-TBU. It describes how the type of invalidation being performed affects the scope of the DTI_TBU_INV_REQ message. Other encodings of the OPERATION field are Reserved.

Table 3-11 DTI-TBU 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, ADDR, 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[a]	EL1-S2	Secure	VMID, ADDR, 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, ADDR, 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[b]	EL1-S2	Secure	VMID, ADDR, 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, ADDR, RANGE, INC_ASET1, SCALE, NUM, TG, TTL
0xB8	TLBI_NS_EL1_ASID	EL1	Non-secure	VMID, ASID, RANGE, INC_ASET1

Table 3-11 DTI-TBU list of invalidation operations (continued)

Code	Invalidation operation	StreamWorld affected	SEC_SID affected	Valid fields
0xB9	TLBI_NS_EL1_VA	EL1	Non-secure	VMID, ASID, ADDR, RANGE, INC_ASET1, SCALE, NUM, TG, TTL
0xB5	TLBI_NS_EL1_S2_IPA	EL1-S2	Non-secure	VMID, ADDR, RANGE, INC_ASET1, SCALE, NUM, TG, TTL
0x180	TLBI_RL_EL1_ALL	EL1, EL1-S2	Realm	INC_ASET1
0x192	TLBI_RL_EL1_S1_VMID	EL1	Realm	VMID, RANGE, INC_ASET1
0x190	TLBI_RL_EL1_S12_VMID	EL1, EL1-S2	Realm	VMID, RANGE, INC_ASET1
0x191	TLBI_RL_EL1_VAA	EL1	Realm	VMID, ADDR, RANGE, INC_ASET1, SCALE, NUM, TG, TTL
0x198	TLBI_RL_EL1_ASID	EL1	Realm	VMID, ASID, RANGE, INC_ASET1
0x199	TLBI_RL_EL1_VA	EL1	Realm	VMID, ASID, ADDR, RANGE, INC_ASET1, SCALE, NUM, TG, TTL
0x195	TLBI_RL_EL1_S2_IPA	EL1-S2	Realm	VMID, ADDR, RANGE, INC_ASET1, SCALE, NUM, TG, TTL
0xC0	TLBI_S_EL2_ALL	EL2	Secure	INC_ASET1
0xC1	TLBI_S_EL2_VAA	EL2	Secure	ADDR, INC_ASET1, SCALE, NUM, TG, TTL
0xC8	TLBI_S_EL2_ASID	EL2	Secure	ASID, INC_ASET1
0xC9	TLBI_S_EL2_VA	EL2	Secure	ASID, ADDR, INC_ASET1, SCALE, NUM, TG, TTL
0xE0	TLBI_NS_EL2_ALL	EL2	Non-secure	INC_ASET1
0xE1	TLBI_NS_EL2_VAA	EL2	Non-secure	ADDR, 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, ADDR, INC_ASET1, SCALE, NUM, TG, TTL
0x1C0	TLBI_RL_EL2_ALL	EL2	Realm	INC_ASET1
0x1C1	TLBI_RL_EL2_VAA	EL2	Realm	ADDR, INC_ASET1, SCALE, NUM, TG, TTL
0x1C8	TLBI_RL_EL2_ASID	EL2	Realm	ASID, INC_ASET1
0x1C9	TLBI_RL_EL2_VA	EL2	Realm	ASID, ADDR, INC_ASET1, SCALE, NUM, TG, TTL
0x40	TLBI_S_EL3_ALL	EL3	Secure	INC_ASET1
0x41	TLBI_S_EL3_VA	EL3	Secure	ADDR, INC_ASET1, SCALE, NUM, TG, TTL
0x00	CFGI_S_ALL	-	Secure	-

Table 3-11 DTI-TBU list of invalidation operations (continued)

Code	Invalidation operation	StreamWorld affected	SEC_SID affected	Valid fields
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
0x100	CFGI_RL_ALL	-	Realm	-
0x110	CFGI_RL_SID	-	Realm	SID, RANGE
0x118	CFGI_RL_SID_SSID	-	Realm	SID, SSID
0x047	TLBI_PA	ALL	-	ADDR, SIZE
0x104	DPTI_RL_ALL[c]	ALL	Realm	-
0x105	DPTI_RL_PA[c]	ALL	Realm	ADDR, SIZE
0x06	INV_ALL	All	All	-

    a. Only matches translations with a Non-secure IPA. For more information, see *Determination of IPA space*.

    b. Only matches translations with a Secure IPA. For more information, see *Determination of IPA space*.

    c. Invalidation operations DPTI_RL_ALL and DPTI_RL_PA are only legal when DTI_TBU_CONDIS_ACK.VERSION > DTI-TBUv3.

When DTI_TBU_CONDIS_ACK.VERSION is DTI-TBUv3, then the following encodings are Reserved:

- 0x104
- 0x105

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

The following invalidation operations will invalidate GlobalBypass translations with DTI_TBU_TRANS_REQ.MMUV = 1 and GlobalDisabled translations of the appropriate security level:

- CFGI_NS_ALL
- CFGI_RL_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, and CFGI_RL_ALL invalidation and sync operations to invalidate GlobalBypass translations with DTI_TBU_TRANS_REQ.MMUV = 1, and GlobalDisable translations as part of the process for changing certain SMMUv3 control registers.

————————

Translations with DTI_TBU_TRANS_RESP.TRANS_RNG = 0b1111 are not invalidated by TLBI_PA.

Translations with DTI_TBU_TRANS_RESP.TRANS_RNG != 0b1111 are invalidated by TLBI_PA irrespective of the value of MMUV.

GlobalBypass translations with DTI_TBU_TRANS_REQ.MMUV = 0 can only be invalidated by TLBI_PA or INV_ALL.

The INV_ALL operation invalidates all caches, including Secure, Non-secure, and Realm translations and translations with DTI_TBU_TRANS_REQ.MMUV = 0.

DTI_TBU_TRANS_RESPEX.MECID is invalidated by INV_ALL and matching CFGI invalidation operations.

Table 3-12 shows the representation of invalidation operations:

Table 3-12 Representation of invalidation operations

Specification use	Represents collectively
CFGI-all	CFGI_S_ALL, CFGI_NS_ALL, CFGI_RL_ALL
CFGI-by-context	CFGI_S_SID, CFGI_NS_SID, CFGI_RL_SID, CFGI_S_SID_SSID, CFGI_NS_SID_SSID, CFGI_RL_SID_SSID
TLBI-not-by-pa	TLBI_S_*, TLBI_NS_*, TLBI_RL_*

Table 3-13 shows the which invalidation operation applies to which type of translation.

Table 3-13 Translation type and applicable invalidation operations

BYPASS	BP_TYPE	MMUV	TRANS_RNG	INV_ALL	CFGI-all	CFGI-by-context	TLBI-not-by-pa	DPTI	TLBI_PA
1	GlobalBypass	0	= 0b1111	Y	N	N	N	N	N
1	GlobalBypass	0	!= 0b1111	Y	N	N	N	N	Y
1	GlobalBypass	1	= 0b1111	Y	Y	N	N	N	N
1	GlobalBypass	1	!= 0b1111	Y	Y	N	N	N	Y
1	GlobalDisabled	1	-	Y	Y	N	N	N	N
1	StreamBypass	1	= 0b1111	Y	Y	Y	N	N	N
1	StreamBypass	1	!= 0b1111	Y	Y	Y	N	N	Y
1	DPTBypass	1	!= 0b1111	Y	Y	Y	N	Y	Y
1	StreamDisabled	1	-	Y	Y	Y	N	N	N
0	-	1	!= 0b1111	Y	Y	Y	Y	N	Y

Range Invalidate operations

DTI-TBU supports Range Invalidation operations. These operations do not involve any RANGE fields specified in messages. The operations in this section refer to TLBI-not-by-pa.

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

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

Table 3-14 shows the Translation_Granule_Size mapping:

Table 3-14 Translation_Granule_Size mapping

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

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 DTI_TBU_TRANS_RESP.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 DTI_TBU_TRANS_RESP.INVAL_RNG in the translation response. Table 3-15 indicates encodings of DTI_TBU_TRANS_RESP.INVAL_RNG that are within scope of an invalidation, dependent upon the TG and TTL fields:

Table 3-15 DTI-TBU encodings of DTI_TBU_TRANS_RESP.INVAL_RNG

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

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.

A TCU must return DTI_TBU_TRANS_RESP.INVAL_RNG values that ensure correct invalidation by a TBU implementing the above rules. That means that DTI_TBU_TRANS_RESP.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 translation table entry.

When the fields of the invalidation operation match any of the following, no invalidation is required to occur. Although not required, an implementation is permitted to perform invalidation as a result of such malformed invalidation operations.

- TG == 0b01 && TTL == 0b01 && Address[29:12] != 0
- TG == 0b01 && TTL == 0b10 && Address[20:12] != 0
- TG == 0b10 && TTL == 0b01 && Address[35:14] != 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.

DTI_TBU_INV_REQ[71] is SCALE[5]. This extends the SCALE field to 6 bits, to enable larger ranges to be specified for invalidation.

Stage 1 translations can be defined for low from address 0 upwards, and high address ranges from address (2^64)-1 downwards. A range invalidation never crosses from one range to the other:

- A range invalidation operation whose upper address exceeds (2^64)-1 does not wrap around to cover addresses from address 0.
- A range invalidation operation starting at an address with ADDR[63] == 0 is not required to invalidate any entries with ADDR[63] == 1.

Configuration invalidate operations

Configuration invalidate operations invalidate StreamDisabled translation and translations with DTI_TBU_TRANS_RESP.BYPASS = 0 or DTI_TBU_TRANS_RESP.BP_TYPE = StreamBypass.

Table 3-16 shows the SMMUv3 commands that map that to DTI configuration invalidate operations.

Table 3-16 Mappings of SMMUv3 commands onto DTI invalidate operations

SMMUv3 command	DTI invalidate operation
CMD_CFGI_ALL	CFGI_S_ALL, CFGI_NS_ALL, CFGI_RL_ALL
CMD_CFGI_STE	CFGI_S_SID, CFGI_NS_SID, CFGI_RL_SID
CMD_CFGI_STE_RANGE	CFGI_S_SID, CFGI_NS_SID, CFGI_RL_SID
CMD_CFGI_CD_ALL	CFGI_S_SID, CFGI_NS_SID, CFGI_RL_SID
CMD_CFGI_CD	CFGI_S_SID_SSID, CFGI_NS_SID_SSID, CFGI_RL_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 CFGI_S_SID_SSID, CFGI_RL_SID_SSID, and CFG_NS_SID_SSID operations.

Realm Invalidation

Each code operates on SEC_SID = Realm. The operations are otherwise identical to the corresponding Non-secure invalidation operation. See Table 3-11 for the list of DTI-TBUv3 invalidation operations.

Realm invalidation operations are only permitted when DTI_TBU_CONDIS_REQ.STAGES is MG.

GPC invalidate operations

The only invalidation operation added is TLBI_PA. For TLBI_PA:

- These operation fields are valid: ADDR and SIZE.
- It invalidates all translations with DTI_TBU_TRANS_RESP.TRANS_RNG != 0b1111 and a translated address that matches the given address and size, with any value of DTI_TBU_TRANS_RESP.NS or DTI_TBU_TRANS_RESP.NSE.
- OPERATION is 0x47.

The SIZE field is DTI_TBU_INV_REQ[15:12]. It overlaps with TG, TTL, and SSID[3:0].

———— **Note** ————

The TLBI_PA operation invalidates previous translations based on their output address in the DTI_TBU_TRANS_RESP.OA field, not the input address.

———————————

The encodings of SIZE are:

0b0000	4KB
0b0001	16KB
0b0010	64KB
0b0011	2MB
0b0100	32MB
0b0101	512MB
0b0110	1GB
0b0111	16GB
0b1000	64GB
0b1001	512GB

All other values are Reserved.

———— **Note** ————

The encodings here do not match those used in DTI_TBU_TRANS_RESP. The encodings here are chosen to match DVM.

———————————

TLBI_PA operation performs range-based invalidation and invalidates translations starting from the address in ADDR, within the range as specified in the SIZE field.

The set of addresses A to be invalidated is given by: ADDR <= A < ADDR + "region size given by SIZE".

———— **Note** ————

The TLBI_PA invalidate range is given by SIZE which is different than the range calculation for TLBI_S_*, TLBI_NS_*, and TLBI_RL_*.

———————————

TLBI_PA operation affects a translation if any address to be invalidated is within the range of the translation, as defined by DTI_TBU_TRANS_RESP.TRANS_RNG in the translation response.

———— **Note** ————

It is DTI_TBU_TRANS_RESP.TRANS_RNG instead of DTI_TBU_TRANS_RESP.INVAL_RNG because DTI_TBU_TRANS_RESP.TRANS_RNG reflects the size of GPC performed for this translation.

———————————

If ADDR is not aligned to the size of the value of SIZE field, no translations are required to be invalidated.

When DTI_TBU_CONDIS_REQ.STAGES is M, TLBI_PA invalidation operations are not permitted.

When DTI_TBU_CONDIS_REQ.STAGES is G, the only invalidation operations permitted are TLBI_PA and INV_ALL.

──── **Note** ────

There is no TLBI_PA_ALL operation required by the TBU because the existing INV_ALL operation can be used.

────────────

DPT invalidation

DPTI_RL_PA invalidates all TLB entries allocated with DTI_TBU_TRANS_REQ.SEC_SID == Realm, DTI_TBU_TRANS_RESP.BYPASS == 1, and DTI_TBU_TRANS_RESP.BP_TYPE == DPTBypass with an input address that matches the given address and size.

DPTI_RL_ALL invalidates all TLB entries allocated with DTI_TBU_TRANS_REQ.SEC_SID == Realm, DTI_TBU_TRANS_RESP.BYPASS == 1, and DTI_TBU_TRANS_RESP.BP_TYPE == DPTBypass.

DPTI_RL_PA and DPTI_RL_ALL are only permitted when DTI_TBU_CONDIS_ACK.VERSION > DTI-TBUv3 and DTI_TBU_CONDIS_REQ.STAGES is MG.

DPTI_RL_PA operation performs range-based invalidation and invalidates translations starting from the address in ADDR, within the range as specified in the SIZE field.

A translation only guaranteed to be invalidated by DPTI_RL_PA operation if SIZE selects a value equal to or greater than DTI_TBU_TRANS_RESP.TRANS_RNG.

If ADDR is not aligned to the the size of the value of SIZE field, no translations are required to be invalidated.

DPTBypass translation is invalidated by INV_ALL, TLBI_PA, DPTI_RL_PA, DPTI_RL_ALL, and configuration invalidate operations of the appropriate security level.

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 512KB of register space.

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_REQ.SUP_REG was 1 during the connect sequence.

Flow control result

None

Field descriptions

The DTI_TBU_REG_WRITE bit assignments are:

7	6	5	4	3	2	1	0	LSB
			DATA					56 48 40 32
		Reserved					NSE	24
NS			ADDR[18:12]					16
			ADDR[11:4]					8
ADDR[3:2]		Reserved		S_MSG_TYPE				0

DATA, bits [63:32]

This field holds the data to be written.

Bits [31:25]

Reserved, SBZ

NSE, bit [24]

{NSE, NS} indicates the physical address space of the register access. For more information, see NS, bit [23].

NS, bit [23]

This bit indicates the physical address space of the register access. The encodings of {NSE, NS} are as follows:

0b00   Secure

0b01   Non-secure

0b10   Root

0b11   Realm

For more information, see NSE, bit [24].

ADDR, bits [22:6]

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

Bits [5:4]

Reserved, SBZ

S_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 acknowledgment.

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:
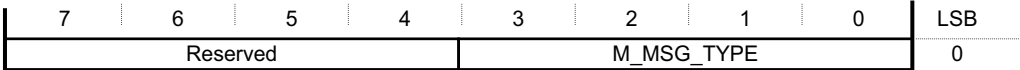
7	6	5	4	3	2	1	0	LSB
	Reserved				M_MSG_TYPE			0

Bits [7:4]

Reserved, SBZ

M_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 register reads or writes.
- DTI_TBU_CONDIS_REQ.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							NSE	24
NS	ADDR[18:12]							16
ADDR[11:4]								8
ADDR[3:2]		Reserved		S_MSG_TYPE				0

Bits [31:25]

Reserved, SBZ

NSE, bit [24]

{NSE, NS} indicates the physical address space of the register access. For more information, see NSE, bit [23].

NS, bit [23]

{NSE, NS} indicates the physical address space of the register access. The encodings of {NSE, NS} are as follows:

0b00 Secure

0b01 Non-secure

0b10 Root

0b11 Realm

ADDR, bits [22:6]

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

S_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_RDATA bit assignments are:

7	6	5	4	3	2	1	0	LSB
								56
			DATA					48
								40
								32
								24
			Reserved					16
								8
		Reserved			M_MSG_TYPE			0

DATA, bits [63:32]

This field holds the read data.

Bits [31:4]

Reserved, SBZ

M_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.

3.5 Message dependencies for DTI-TBU

The message dependencies for the DTI-TBU protocol are shown in Figure 3-2.

In this dependency diagram:

- The light gray box indicates the messages originated from the TCU.

- The dark gray box indicates the messages originated from the TBU.

- The dotted line box indicates the translated transaction that is not a DTI message but is useful for analyzing the dependency.

- An arrow starting from message A and ending with message B represents a dependency on the DTI-TBU link between TBU and TCU.

    — When two messages travel in the same direction, the "must not overtake" rules mean that if the message B is observed before message A by the sender, then the same order must be observed by the receiver.

    — When two messages travel in the opposite directions, the "must wait" rules mean that the sender of message A expects to receive message B, and message B must arrive before message A can be presented on DTI.

- A message dependency could be any of the following:

    — Messages that are credit controlled. For example, the maximum number of DTI_TBU_INV_ACK messages before another DTI_TBU_INV_REQ message can be issued, is DTI_TBU_CONDIS_REQ.TOK_INV_GNT.

    — The messages with order requirements defined in this protocol.

    — Any dependency because of an external protocol, for example, DVM dependency.
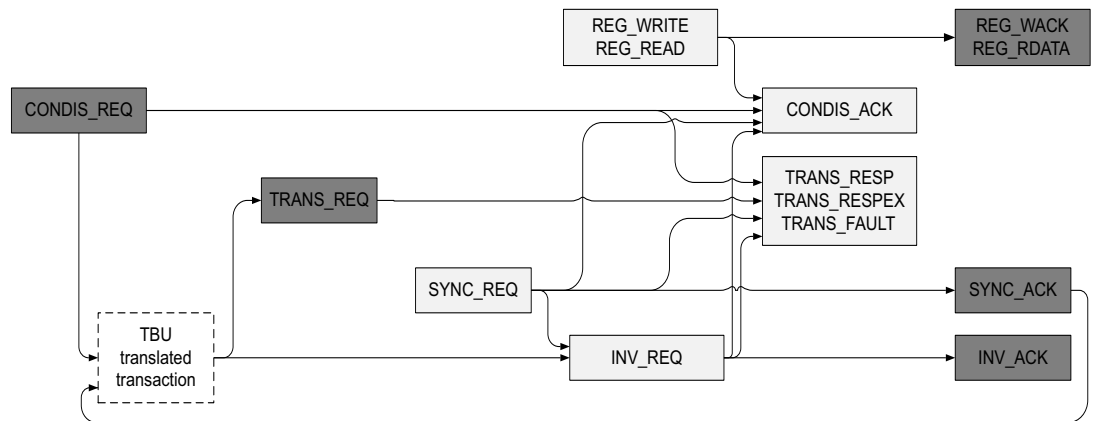
Figure 3-2 DTI-TBU message dependency sequence

The message dependencies rules are as follows:

- DTI_TBU_REG_WRITE message must wait for any outstanding DTI_TBU_REG_WACK or DTI_TBU_REG_RDATA message.

- DTI_TBU_REG_READ message must wait for any outstanding DTI_TBU_REG_WACK or DTI_TBU_REG_RDATA message.

- DTI_TBU_CONDIS_REQ message must wait for any outstanding DTI_TBU_CONDIS_ACK message.

- DTI_TBU_CONDIS_REQ message must wait for any outstanding DTI_TBU_TRANS_RESP, DTI_TBU_TRANS_RESPEX or DTI_TBU_TRANS_FAULT messages.

- DTI_TBU_CONDIS_REQ message must wait for all translated transactions to complete in the TBU.

- DTI_TBU_INV_REQ, DTI_TBU_SYNC_REQ, DTI_TBU_REG_READ, and DTI_TBU_REQ_WRITE messages must not overtake DTI_TBU_CONDIS_ACK message.

- DTI_TBU_TRANS_REQ message must wait for outstanding DTI_TBU_TRANS_RESP, DTI_TBU_TRANS_RESPEX or DTI_TBU_TRANS_FAULT messages to return tokens when there is no translation token left.

- DTI_TBU_INV_REQ message must wait for outstanding DTI_TBU_INV_ACK messages to return tokens when there is no invalidation token left.

- DTI_TBU_INV_REQ message must not overtake any DTI_TBU_TRANS_RESP, DTI_TBU_TRANS_RESPEX or DTI_TBU_TRANS_FAULT messages which are invalidated by this DTI_TBU_INV_REQ message, unless DO_NOT_CACHE is set to 1.

- DTI_TBU_SYNC_REQ message must wait for any outstanding DTI_TBU_SYNC_ACK message.

- DTI_TBU_SYNC_REQ message must not overtake any DTI_TBU_INV_REQ messages.

- DTI_TBU_SYNC_REQ message must not overtake any DTI_TBU_TRANS_RESP, DTI_TBU_TRANS_RESPEX or DTI_TBU_TRANS_FAULT messages that were invalidated by any preceding DTI_TBU_INV_REQ message.

- DTI_TBU_SYNC_ACK message can wait for translated transactions to complete in the TBU.

- TBU translated transaction can wait for DTI_TBU_TRANS_REQ message when the TBU translated transaction has order dependency with another TBU translated transaction still requiring translation.

- TBU translated transaction can wait for DTI_TBU_INV_REQ message. If the translated transaction requires cache coherent memory access, it might result in dependency with DVM invalidate messages sharing the same snooping interface.

Chapter 4
DTI-ATS Messages

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

It contains the following sections:

4.1 Connection and disconnection message group

The DTI-ATS protocol is designed to enable a TCU to simultaneously support DTI-ATSv1, DTI-ATSv2, DTI-ATSv3, and DTI-ATSv4 connections from different PCIe RPs.

This section contains the following subsections:

4.1.1 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
TOK_TRANS_REQ[11:8]				Reserved		SUP_T	NO_TRANS	24
TOK_INV_GNT				TOK_TRANS_REQ[7:4]				16
TOK_TRANS_REQ[3:0]				VERSION				8
Reserved		PROTOCOL	STATE	M_MSG_TYPE				0

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

The size of TOK_TRANS_REQ field is dependent on the version of the DTI-ATS protocol.

DTI-ATSv1, DTI-ATSv2, DTI-ATSv3

TOK_TRANS_REQ[11:8] is Reserved, SBZ.

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

DTI-ATSv4

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

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

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

When (NO_TRANS == 0 or DTI_ATS_CONDIS_REQ.VERSION == DTI-ATSv1) 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 or DTI_ATS_CONDIS_REQ.VERSION == DTI-ATSv1) 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 and DTI_ATS_CONDIS_REQ.VERSION > DTI-ATSv1:

Reserved, SBZ

Bits [27:26]

Reserved, SBZ

SUP_T, bit [25]

DTI-ATSv1, DTI-ATSv2

Reserved, SBZ

DTI-ATSv3, DTI-ATSv4

- When DTI_ATS_CONDIS_REQ.STATE == 1, this bit indicates the requested T bit usage in DTI_ATS_TRANS_REQ, DTI_ATS_INV_REQ, DTI_ATS_PAGE_REQ, and DTI_ATS_PAGE_RESP messages.
  — 0: T must be 0.
  — 1: T can be 0 or 1.
- When DTI_ATS_CONDIS_REQ.STATE == 0:
  — This field is ignored.

NO_TRANS, bit [24]

DTI-ATSv1

Reserved, SBZ

DTI-ATSv2, DTI-ATSv3, DTI-ATSv4

When this bit is 1 and DTI_ATS_CONDIS_ACK.VERSION > DTI-ATSv1:

- 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 0 and the VERSION field in the DTI_ATS_CONDIS_ACK message that established the connection was greater than DTI-ATSv1, then the value of this field must match with the value of NO_TRANS in the previous connect request with STATE == 1.

When STATE is 0 and the VERSION field in the DTI_ATS_CONDIS_ACK message that established the connection was DTI-ATSv1, then the value of this field must be 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 or DTI_ATS_CONDIS_REQ.VERSION == DTI-ATSv1:

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 and DTI_ATS_CONDIS_REQ.VERSION > DTI-ATSv1:

Reserved, SBZ

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

See the description for TOK_TRANS_REQ [11:8], bits [31:28].

VERSION, bits [11:8]

This field indicates the requested protocol version.

`0b0000`	DTI-ATSv1
`0b0001`	DTI-ATSv2
`0b0010`	DTI-ATSv3
`0b0011`	DTI-ATSv4

All other encodings are for future protocol versions and are currently not defined.

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.

M_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

4.1.2 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 acknowledgment.

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
TOK_TRANS_GNT[11:8]				Reserved		SUP_T	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	S_MSG_TYPE				0

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

> The size of this field is dependent on the version of the DTI-ATS protocol. The meaning of this field depends on the value of DTI_ATS_CONDIS_REQ.NO_TRANS.

> **DTI-ATSv1, DTI-ATSv2, DTI-ATSv3**

>> TOK_TRANS_GNT[11:8] is Reserved, SBZ.
>> TOK_TRANS_GNT[7:0] is bits [19:12].

> **DTI-ATSv4**

>> TOK_TRANS_GNT[11:8] is bits [31:28].
>> TOK_TRANS_GNT[7:0] is bits [19:12].

> **When DTI_ATS_CONDIS_REQ.NO_TRANS == 0 or DTI_ATS_CONDIS_ACK.VERSION == DTI-ATSv1**:

>> Indicates the number of preallocated 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.

>> **DTI-ATSv1, DTI-ATSv2**:

>>> When STATE is 1, 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.

>> **DTI-ATSv3, DTI-ATSv4**:

>>> When STATE is 1, the value of this field must equal the value of the TOK_TRANS_REQ field in the DTI_ATS_CONDIS_REQ message that initiated the connection.

>> When STATE is 0, this field is ignored.

> **When DTI_ATS_CONDIS_REQ.NO_TRANS == 1 and DTI_ATS_CONDIS_ACK.VERSION > DTI-ATSv1**:

>> Reserved, SBZ

Bits [27:26]

> Reserved, SBZ

SUP_T, bit [25]

> **DTI-ATSv1, DTI-ATSv2**

>> Reserved, SBZ

> **DTI-ATSv3, DTI-ATSv4**

>> • When DTI_ATS_CONDIS_ACK.STATE == 1 and DTI_ATS_CONDIS_REQ.SUP_T == 1, this bit indicates the granted T bit usage in DTI_ATS_TRANS_REQ, DTI_ATS_INV_REQ, DTI_ATS_PAGE_REQ, and DTI_ATS_PAGE_RESP messages.

>> — 0: T must be 0.

>> — 1: T can be 0 or 1.

- When DTI_ATS_CONDIS_ACK.STATE == 1 and DTI_ATS_CONDIS_REQ.SUP_T == 0:

    — This field is Reserved, SBZ.

- When DTI_ATS_CONDIS_ACK.STATE == 0:

    — This field is 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, DTI-ATSv3, DTI-ATSv4

Reserved, SBZ

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-ATSv1

The DTI_ATS_PAGE_RESP message is permitted to be issued regardless of the value of this bit. A PCIe RP must be able to accept DTI_ATS_PAGE_RESP messages but does not have to process them.

DTI-ATSv2, DTI-ATSv3, DTI-ATSv4

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

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

See the description for TOK_TRANS_GNT [11:8], bits [31:28].

VERSION, bits [11:8]

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

0b0000	DTI-ATSv1
0b0001	DTI-ATSv2
0b0010	DTI-ATSv3
0b0011	DTI-ATSv4

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.

S_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

4.2 Translation request message group

This section contains the following subsections:

- *DTI_ATS_TRANS_REQ*
- *DTI_ATS_TRANS_RESP* on page 4-97
- *DTI_ATS_TRANS_FAULT* on page 4-103
- *The ATS translation sequence* on page 4-105

4.2.1 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 consumes a translation token.

Field descriptions

The DTI_ATS_TRANS_REQ bit assignments are:

7	6	5	4	3	2	1	0	LSB
								152
								144
			IA[63:16]					136
								128
								120
								112
IA[15:12]				Reserved				104
Reserved								96
			SSID[19:4]					88
								80
SSID[3:0]				Reserved				72
Reserved								64
								56
			SID					48
								40
								32
TRANSLATION_ID[11:8]				Reserved				24
Reserved	CXL	SSV	T	nW	InD	PnU	Protocol	16
TRANSLATION_ID[7:0]								8
QOS				M_MSG_TYPE				0

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.

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

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

> **DTI-ATSv1, DTI-ATSv2, DTI-ATSv3**
>
> > TRANSLATION_ID[11:8] is Reserved, SBZ.
> >
> > TRANSLATION_ID[7:0], bits [15:8].
> > Any 8-bit translation ID in TRANSLATION_ID[7:0] can be used if the maximum number of outstanding translation requests is not exceeded.
>
> **DTI-ATSv4**
>
> > TRANSLATION_ID[11:8], bits [31:28].
> >
> > TRANSLATION_ID[7:0], bits [15:8].
> >
> > Any 12-bit translation ID can be used if the maximum number of outstanding translation requests is not exceeded.

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.

Bits [31:23]

Reserved, SBZ

CXL, bit [22]

> **DTI-ATSv1, DTI-ATSv2**
>
> > Reserved, SBZ
>
> **DTI-ATSv3, DTI-ATSv4**
>
> > This is set to the value of the Source_CXL bit in the ATS request.

SSV, bit [21]

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

0   The SSID field is not valid.

1   The SSID field is valid.

T, bit [20]

> **DTI-ATSv1, DTI-ATSv2**
>
> > Reserved, SBZ
>
> **DTI-ATSv3, DTI-ATSv4**
>
> > When DTI_ATS_CONDIS_REQ.SUP_T and DTI_ATS_CONDIS_ACK.SUP_T are both 1 for the connection, various messages include a T bit to indicate that the message corresponds to a trusted entity. In each case:
> >
> > **0**   Indicates a Non-secure StreamID.
> >
> > **1**   Indicates a Realm StreamID.
> >
> > If DTI_ATS_CONDIS_REQ.SUP_T and DTI_ATS_CONDIS_ACK.SUP_T are not both 1 during the connection sequence, this field must be 0.

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

See TRANSLATION_ID[11:8], 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.

M_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

PCIe Translation Request mapping to DTI_ATS_TRANS_REQ

When a PCIe Translation Request is received, the DTI_ATS_TRANS_REQ fields should be driven as shown in Table 4-1:

Table 4-1 PCIe Translation Request mapping to DTI_ATS_TRANS_REQ

DTI_ATS_TRANS_REQ field	Value
SID	SID[15:0] is the Requester ID, otherwise known as BDF (Bus, Device, Function). Higher-order bits of SID uniquely identify the PCIe segment in the StreamID space that is used by the SMMU.
CXL[a]	This field is CXL_src.
nW	This field is NW.
SSV	If the PCIe Translation Request has a PASID, this field is 1. Otherwise, this field is 0.
PnU	If the PCIe Translation Request has a PASID, this field is Priv. Otherwise, this field is 0.

Table 4-1 PCIe Translation Request mapping to DTI_ATS_TRANS_REQ (continued)

DTI_ATS_TRANS_REQ field	Value
InD	If the PCIe Translation Request has a PASID, this field is Exe. Otherwise, this field is 0.
SSID	If the PCIe Translation Request has a PASID, this field is PASID. Otherwise, this field is 0.
T[a]	T

    a.   This is only applicable to DTI-ATSv3 and DTI-ATSv4.

4.2.2 DTI_ATS_TRANS_RESP

The DTI_ATS_TRANS_RESP message is used to 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:

7	6	5	4	3	2	1	0	LSB
								152
								144
			OA[63:16]					136
								128
								120
								112
	OA[15:12]				Reserved			104
			Reserved					96
Reserved		AMA			Reserved			88
	Reserved				TRANS_RNG			80
	TRANSLATION_ID [11:8]				Reserved			72
Reserved	TE		Reserved		ALLOW_X	ALLOW_W	ALLOW_R	64
								56
								48
			Reserved					40
								32
								24
		Reserved				BYPASS	Reserved	16
Reserved		CXL_IO	UNTRANSLATED		TRANSLATION_ID[7:4]			8
TRANSLATION_ID[3:0]				S_MSG_TYPE				0

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, DTI-ATSv3, DTI-ATSv4

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, DTI-ATSv3, DTI-ATSv4

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

DTI-ATSv1, DTI-ATSv2

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
0b0111	16GB
0b1000	4TB
0b1001	128TB

All other values are Reserved.

——— **Note** ———

When DTI-ATSv1 or DTI-ATSv2 is used and translated page size is 64GB or 512GB, the TCU uses a value of 0b0111 instead, indicating a 16GB range.

DTI-ATSv3, DTI-ATSv4

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
0b0111	16GB
0b1010	64GB
0b1011	512GB
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, DTI-ATSv3, DTI-ATSv4

Reserved, SBZ

TRANSLATION_ID[11:8], bits [79:76]

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

DTI-ATSv1, DTI-ATSv2, DTI-ATSv3

TRANSLATION_ID[11:8] is Reserved, SBZ.

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

DTI-ATSv4

TRANSLATION_ID[11:8] is bits [79:76].

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.

Bits [75:71]

Reserved, SBZ

TE, bit [70]

DTI-ATSv1, DTI-ATSv2

Reserved, SBZ

DTI-ATSv3, DTI-ATSv4

Indicates the security space of the address.

0	OA is a Non-secure address.
1	OA is a Realm address.

When DTI_ATS_TRANS_REQ.T is 0, TE is Reserved, SBZ.

Bits [69:67]

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.

It is expected that TCU should drive this field to 1 if the permission is granted by the translation.

ALLOW_W, bit [65]

This bit indicates permissions for data write accesses.

0	Not permitted
1	Permitted

It is expected that TCU should drive this field to 1 if the permission is granted by the translation.

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.

It is expected that TCU should drive this field to 1 if the permission is granted by the translation.

Bits [63:18]

Reserved, SBZ

BYPASS, bit [17]

This field indicates that translation for this StreamID is bypassed. When the value of this field is 1, the VA and the PA of the translation are the same.

0	Normal translation
1	Translation bypassed

DTI-ATSv1

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, DTI-ATSv3, DTI-ATSv4

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.

DTI-ATSv3, DTI-ATSv4

Reserved, SBZ when DTI_ATS_TRANS_REQ.CXL= 0.

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

See TRANSLATION_ID[11:8], bits [79:76].

S_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

DTI_ATS_TRANS_RESP mapping to PCIe Translation Completion

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

Table 4-2 DTI_ATS_TRANS_RESP mapping to PCIe Translation Completion

PCIe TLP field	Value
Translated Address and 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
U	DTI_ATS_TRANS_RESP.UNTRANSLATED
R	DTI_ATS_TRANS_RESP.ALLOW_R
W	DTI_ATS_TRANS_RESP.ALLOW_W
CXL.io	DTI-ATSv2 — If Source_CXL set in PCIe translation request: DTI_ATS_TRANS_RESP.CXL_IO — Else: `0b0` DTI-ATSv3 and DTI-ATSv4 — DTI_ATS_TRANS_RESP.CXL_IO
AMA[b]	DTI_ATS_TRANS_RESP.AMA
T[c]	DTI_ATS_TRANS_REQ.T

a. Previous versions (Edition 0 to Edition 3) of this specification included a GLOBAL field in DTI_ATS_TRANS_RESP. This was an error since the SMMUv3 architecture requires the Global field in a Translation Completion to be 0.

b. DTI-ATSv2, DTI-ATSv3, and DTI-ATSv4

c. DTI-ATSv3 and DTI-ATSv4

4.2.3 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:

7	6	5	4	3	2	1	0	LSB
TRANSLATION_ID[11:8]				Reserved				24
Reserved					FAULT_TYPE		Reserved	16
Reserved				TRANSLATION_ID[7:4]				8
TRANSLATION_ID[3:0]				S_MSG_TYPE				0

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

The size of this field is dependent on the version of the DTI-ATS protocol.

DTI-ATSv1, DTI-ATSv2, DTI-ATSv3

TRANSLATION_ID[11:8] is Reserved, SBZ.

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

DTI-ATSv4

TRANSLATION_ID[11:8] is 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.

Bits [31:19]

Reserved, SBZ

FAULT_TYPE, bits [18:17]

This field 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 [7:0], bits [11:4]

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

S_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

4.2.4 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 4-1 shows the steps required in a full ATS translation process that is supported by DTI.
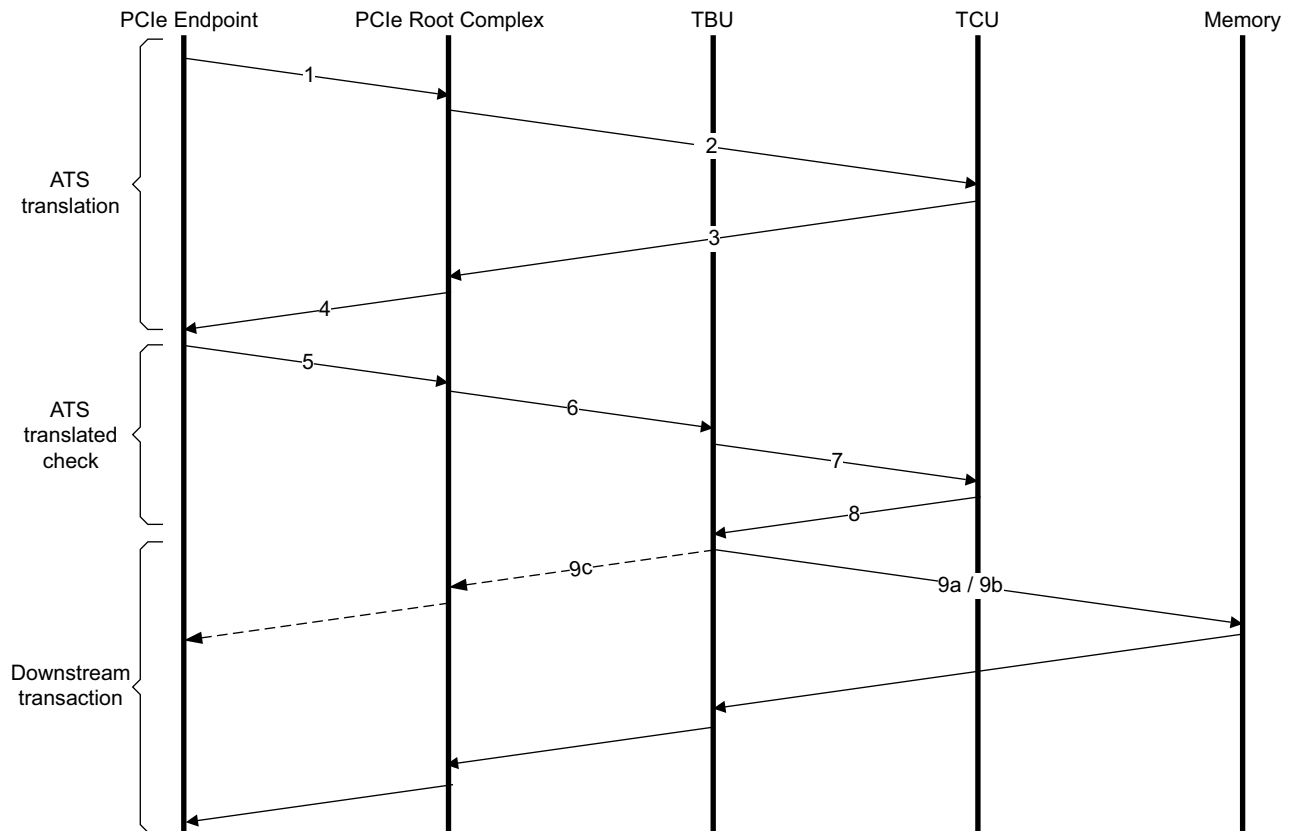
Figure 4-1 Example complete ATS translation sequence in DTI

The steps in Figure 4-1 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 performs 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.

4.3 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:

- *DTI_ATS_INV_REQ*
- *DTI_ATS_INV_ACK* on page 4-109
- *DTI_ATS_INV_COMP* on page 4-110
- *DTI_ATS_SYNC_REQ* on page 4-111
- *DTI_ATS_SYNC_ACK* on page 4-112
- *The DTI-ATS invalidation sequence* on page 4-113
- *DTI-ATS invalidation operations* on page 4-114

4.3.1 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:

7	6	5	4	3	2	1	0	LSB
								120
								112
			VA[63:16]					104
								96
								88
								80
	VA[15:12]			ITAG				72
ITAG	T			RANGE				64
								56
			SID					48
								40
								32
			SSID[19:4]					24
								16
	SSID[3:0]			OPERATION[7:4]				8
	OPERATION[3:0]			S_MSG_TYPE				0

VA, bits [127:76]

> The Virtual Address or Intermediate Physical Address to be invalidated.

ITAG, bits [75:71]

> **DTI-ATSv1, DTI-ATSv2**
>
>> Reserved, SBZ
>
> **DTI-ATSv3, DTI-ATSv4**
>
>> This can be any value which does not match a prior DTI_ATS_INV_REQ message with the same SID that has not yet received a corresponding DTI_ATS_INV_COMP message.

T, bit [70]

> **DTI-ATSv1, DTI-ATSv2**
>
>> Reserved, SBZ
>
> **DTI-ATSv3, DTI-ATSv4**
>
>> When DTI_ATS_CONDIS_REQ.SUP_T and DTI_ATS_CONDIS_ACK.SUP_T are both 1 for the connection, various messages include a T bit to indicate that the message corresponds to a trusted entity. In each case:
>>
>> **0**   Indicates a Non-secure StreamID.
>>
>> **1**   Indicates a Realm StreamID.
>>
>> When DTI_ATS_CONDIS_REQ.SUP_T and DTI_ATS_CONDIS_ACK.SUP_T are not both 1 for the connection then DTI_ATS_INV_REQ messages with T == 1 must not be sent to Root Ports.

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 bottom RANGE 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.
>
> ──── **Note** ────
>
> The PCIe RP must be aware of the StreamID range which it occupies. When the StreamID is outside of its range, the PCIe RP must generate a DTI_ATS_INV_ACK message instead of trying to issue an invalidate request message to an endpoint. This is not an error case and must not cause ERROR bit set in the DTI_ATS_INV_COMP message or DTI_ATS_SYNC_ACK message.
>
> ────────

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, it is recommended 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 4-114.

S_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

DTI_ATS_INV_REQ mapping to PCIe Invalidate Request

When a DTI_ATS_INV_REQ message is received, the PCIe Invalidate Request fields should be driven as shown in Table 4-3:

Table 4-3 DTI_ATS_INV_REQ mapping to PCIe Invalidate Request

PCIe TLP field	Value
Untranslated Address, and S	Depends on VA and RANGE fields of DTI_ATS_INV_REQ
PASID	When DTI_ATS_INV_REQ.OPERATION is ATCI_PASID or ATCI_PASID_GLOBAL, the PCIe Invalidate Request has a PASID and the PASID value is DTI_ATS_INV_REQ.SSID. Otherwise, PCIe Invalidate Request does not have a PASID.
T[a]	DTI_ATS_INV_REQ.T
ITag[a]	DTI_ATS_INV_REQ.ITAG
Global Invalidate	When DTI_ATS_INV_REQ.OPERATION is ATCI_PASID_GLOBAL, this is 1. Otherwise, this is 0.
Destination ID	This field is DTI_ATS_INV_REQ.SID[15:0]. Higher-order bits of DTI_ATS_INV_REQ.SID uniquely identify the PCIe segment in the StreamID space that is used by the SMMU.

a. DTI-ATSv3 and DTI-ATSv4

4.3.2   DTI_ATS_INV_ACK

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

Description

A cache data invalidate acknowledgment.

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				M_MSG_TYPE				0

Bits [7:4]

Reserved, SBZ

M_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

4.3.3   DTI_ATS_INV_COMP

When RP receives all PCIe invalidate completion messages from the EP for a PCIe invalidate request, the RP must return a DTI_ATS_INV_COMP message.

Description

A cache data invalidate completion.

Source

PCIe RP

Usage constraints

Protocol version is DTI-ATSv3 or greater.

The TCU must have previously issued a DTI_ATS_INV_REQ that has not yet had a corresponding DTI_ATS_INV_COMP.

Flow control result

None

Field descriptions

The DTI_ATS_INV_COMP bit assignments are:

7	6	5	4	3	2	1	0	LSB
Reserved								88 / 80
Reserved				ITAG				72
ITAG	T	Reserved						64
SID								56 / 48 / 40 / 32
Reserved								24 / 16 / 8
Reserved		ERROR		M_MSG_TYPE				0

Bits [95:76]

Reserved, SBZ

ITAG, bits [75:71]

Must match the corresponding DTI_ATS_INV_REQ

T, bit [70]

Must match the corresponding DTI_ATS_INV_REQ

Bits [69:64]

Reserved, SBZ

SID, bits [63:32]

Must match the corresponding DTI_ATS_INV_REQ

Bits [31:5]

      Reserved, SBZ

ERROR, bit [4]

      0   Invalidation completion return successfully.

      1   Invalidation could not be completed.

———— **Note** ————

The PCIe RP must be aware of the StreamID range which it occupies. When the StreamID is outside of its range, it is not an error case and must not cause this bit to be 1.

M_MSG_TYPE, bits [3:0]

      This field identifies the message type. It must be `0xB`.

The DTI_ATS_INV_COMP is corresponding to a DTI_ATS_INV_REQ if their ITAG and SID both fields are matching.

There can only be one DTI_ATS_INV_COMP message for each DTI_ATS_INV_REQ message.

There must be one DTI_ATS_INV_COMP message for each DTI_ATS_INV_REQ message.

A DTI_ATS_INV_COMP message is permitted to arrive before a DTI_ATS_INV_ACK corresponding to a DTI_ATS_INV_REQ.

If multiple PCIe invalidate completion messages are sent for the same invalidation to cover multiple Traffic Classes, these must be coalesced by the Root Port into a single DTI_ATS_INV_COMP message, which sets ERROR if any error occurred.

A DTI_ATS_INV_COMP does not require any invalidation tokens.

It is possible to have more DTI_ATS_INV_COMP messages outstanding than available invalidation tokens.

4.3.4   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

DTI-ATSv1, DTI-ATSv2

      The TCU must have received a DTI_ATS_INV_ACK for all previous DTI_ATS_INV_REQ messages.

      The TCU must have received a DTI_ATS_SYNC_ACK for all previous DTI_ATS_SYNC_REQ messages.

DTI-ATSv3, DTI-ATSv4

      The TCU must have received a DTI_ATS_SYNC_ACK for all previous DTI_ATS_SYNC_REQ messages.

———— **Note** ————

It is legal to receive the message even when there are no prior invalidation requests to synchronize.

Flow control result

      None

Field descriptions

The DTI_ATS_SYNC_REQ bit assignments are:

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

Bits [7:4]

Reserved, SBZ

S_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

4.3.5 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:

7	6	5	4	3	2	1	0	LSB
Reserved			ERROR	M_MSG_TYPE				0

Bits [7:5]

Reserved, SBZ

ERROR, bit [4]

DTI-ATSv1, DTI-ATSv2

This bit indicates that a PCIe error has occurred.

0   Success

1   Error

———— **Note** ————

The PCIe RP must be aware of the StreamID range which it occupies. When the StreamID is outside of its range, it is not an error case and must not cause this bit to be 1.

————

DTI-ATSv3, DTI-ATSv4

Reserved, SBZ

M_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

4.3.6   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 4-2.
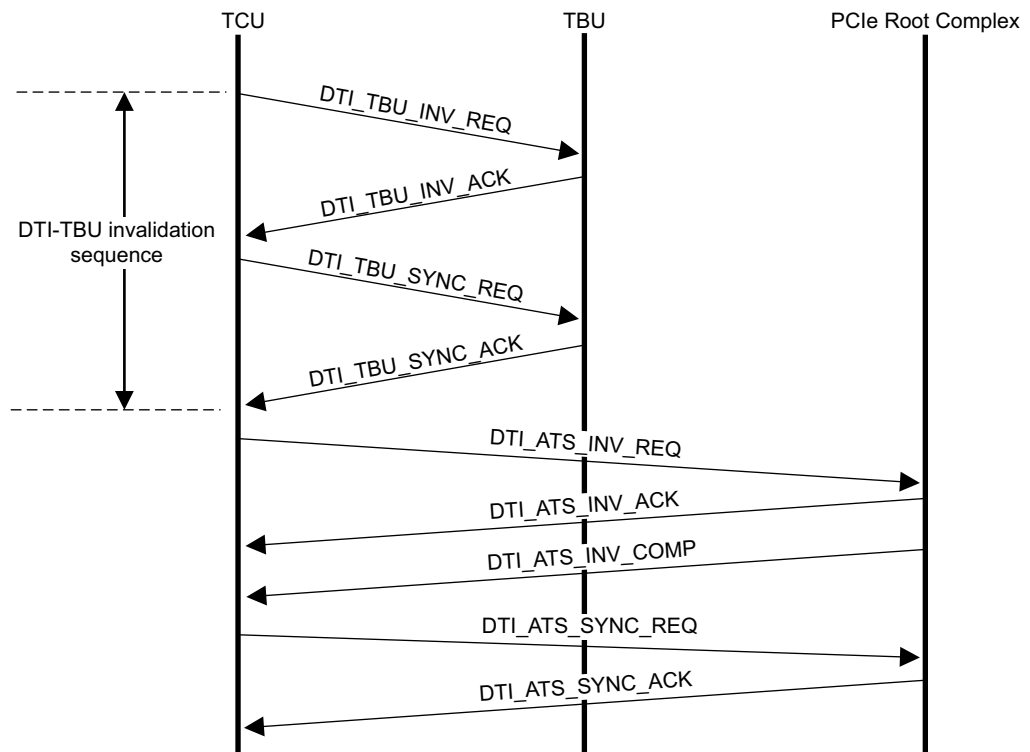
Figure 4-2 DTI-ATS invalidation sequence

The invalidation sequence in Figure 4-2 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.

--------- **Note** ---------
In step 3, the DTI_ATS_INV_COMP message is only applicable to DTI-ATSv3 or later version.

In DTI-ATSv1 and DTI-ATSv2, 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.

In DTI-ATSv3 or later, the return of a DTI_ATS_SYNC_ACK message instead indicates that:

- No further accesses to memory are made using translations invalidated by DTI_ATS_INV_REQ messages which returned a DTI_ATS_INV_COMP message before the DTI_ATS_SYNC_REQ was received.

———— **Note** ————

A DTI_ATS_SYNC_ACK message is likely to be dependent on 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 depends 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 back pressure. 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 back pressure 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_ACK 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.

4.3.7 DTI-ATS invalidation operations

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

Types of invalidation operation

Table 4-4 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 4-4 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

─── **Note** ───

Arm recommends that the ATCI_PASID_GLOBAL operation is not used.

- PCIe endpoints are only required to support Invalidation Requests that have the Global Invalidate bit set if the Global Invalidate Supported bit is set in the PCIe ATS Capability Register. The PCIe specification strongly recommends that this bit is clear, that is, the Global Invalidate bit is ignored in all Invalidate Requests.
- DTI-ATS does not create ATS global translations.
- The DTI-ATS protocol does not provide a PASID with ATCI_PASID_GLOBAL invalidation operations.
- The PCIe specification requires that a PASID is provided for any Invalidation Request with the Global Invalidate bit set.

Software is not expected to issue ATC global invalidation operations.

Arm recommends that:

- If a PCIe RP receives ATCI_PASID_GLOBAL, it always issues the ATS Invalidate Request with Global Invalid bit set and including a PASID. The PASID value is recommended to be SSID or otherwise 0.

Mapping DTI-ATS to SMMUv3 invalidate operations

DTI-ATS invalidation operations are generated as a result of commands in the SMMU Command queue. Table 4-5 shows how these are mapped to DTI-ATS invalidate operations.

Table 4-5 Mapping DTI-ATS operation to SMMUv3 command

SMMUv3 Command	SSV 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

Table 4-6 provides additional information for mapping the SMMUv3 CMD_ATC_INV command to the DTI_ATS_INV_REQ message.

Table 4-6 CMD_ATC_INV command mapping to DTI_ATS_INV_REQ

DTI_ATS_INV_REQ field	Value
VA	Address
T[a]	If the CMD_ATC_INV command is issued on a Realm command queue, this is 1. Otherwise, this is 0.
RANGE	Size
SID	StreamID
SSID	SubstreamID

　　a.　T mapping only applies to DTI-ATSv3 and DTI-ATSv4.

For more information, see the *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3*.

4.4 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:

- *DTI_ATS_PAGE_REQ*
- *DTI_ATS_PAGE_ACK* on page 4-119
- *DTI_ATS_PAGE_RESP* on page 4-120
- *DTI_ATS_PAGE_RESPACK* on page 4-122
- *Generating the page response* on page 4-123

4.4.1 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:

7	6	5	4	3	2	1	0	LSB
								120
								112
		ADDR[63:16]						104
								96
								88
								80
ADDR[15:12]				Reserved			PRG_INDEX[8]	72
PRG_INDEX[7:0]								64
								56
		SID						48
								40
								32
SSID[19:4]								24
								16
SSID[3:0]				SSV	LAST	WRITE	READ	8
INST	PRIV	T	PROTOCOL	M_MSG_TYPE				0

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   The SSID field is not valid.

1   The SSID field is 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

T, bit [5]

    DTI-ATSv1, DTI-ATSv2

        Reserved, SBZ

    DTI-ATSv3, DTI-ATSv4

        When DTI_ATS_CONDIS_REQ.SUP_T and DTI_ATS_CONDIS_ACK.SUP_T are both 1 for the connection, various messages include a T bit to indicate that the message corresponds to a trusted entity. In each case:

        0   Indicates a Non-secure StreamID.

        1   Indicates a Realm StreamID.

If DTI_ATS_CONDIS_REQ.SUP_T and DTI_ATS_CONDIS_ACK.SUP_T are not both 1 during the connection sequence, this field must be 0.

PROTOCOL, bit [4]

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

1   DTI-ATS

This bit must be 1.

M_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

PCIe Page Request mapping to DTI_ATS_PAGE_REQ

When a PCIe Page Request is received, the DTI_ATS_PAGE_REQ fields should be driven as shown in Table 4-7:

Table 4-7 PCIe Page Request mapping to DTI_ATS_PAGE_REQ

DTI_ATS_PAGE_REQ Field	Value
ADDR	This is Page Address.
PRG_INDEX	This is Page Request Group Index.
SID	SID[15:0] is the Requester ID, otherwise known as BDF (Bus, Device, Function).Higher-order bits of SID uniquely identify the PCIe segment in the StreamID space that is used by the SMMU.
SSID	If the PCIe Translation Request has a PASID, this field is PASID. Otherwise, this field is 0.
SSV	If the PCIe Translation Request has a PASID, this field is 1. Otherwise, this field is 0.
Inst	If the PCIe Translation Request has a PASID, this field is Exe. Otherwise, this field is 0.
Priv	If the PCIe Translation Request has a PASID, this field is Priv. Otherwise, this field is 0.
Last	This is L.
Write	This is W.
Read	This is R.
T[a]	This is T.

a.   DTI-ATSv3 and DTI-ATSv4 only

4.4.2   DTI_ATS_PAGE_ACK

The DTI_ATS_PAGE_ACK message is used to acknowledge a page request.

Description

A page request acknowledgment.

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				S_MSG_TYPE				0

Bits [7:4]

Reserved, SBZ

S_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

4.4.3   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

- **DTI-ATSv1**: None

- **DTI-ATSv2**, **DTI-ATSv3**, **DTI-ATSv4**: There must be no current unacknowledged DTI_ATS_PAGE_RESP message.

Flow control result

None

Field descriptions

The DTI_ATS_PAGE_RESP bit assignments are:

7	6	5	4	3	2	1	0	LSB
Reserved								88
								80
Reserved		RESP		Reserved			PRG_INDEX[8]	72
PRG_INDEX[7:0]								64
SID								56
								48
								40
								32
SSID[19:4]								24
								16
SSID[3:0]				SSV		Reserved		8
Reserved	T	Reserved		S_MSG_TYPE				0

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.

——— **Note** ———

The PCIe RP must be aware of the StreamID range that it occupies. When the StreamID is outside of its range, the PCIe RP must generate a DTI_ATS_PAGE_RESPACK message instead of trying to issue a PRI response message to an endpoint.

—————————————

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   The SSID field is not valid.

1   The SSID field is valid.

Bits [10:7]

Reserved, SBZ

T, bit [6]

DTI-ATSv1, DTI-ATSv2

Reserved, SBZ

DTI-ATSv3, DTI-ATSv4

When DTI_ATS_CONDIS_REQ.SUP_T and DTI_ATS_CONDIS_ACK.SUP_T are both 1 for the connection, various messages include a T bit to indicate that the message corresponds to a trusted entity. In each case:

0   Indicates a Non-secure StreamID.

1   Indicates a Realm StreamID.

When DTI_ATS_CONDIS_REQ.SUP_T and DTI_ATS_CONDIS_ACK.SUP_T are not both 1 for the connection then DTI_ATS_PAGE_RESP messages with T == 1 must not be sent to Root Ports.

Bits, bit [5:4]

Reserved, SBZ

S_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

DTI_ATS_PAGE_RESP mapping to PCIe PRG Response

When a DTI_ATS_PAGE_RESP message is received, the PCIe PRG Response fields should be driven as shown in Table 4-8:

Table 4-8 DTI_ATS_PAGE_RESP mapping to PCIe PRG Response

PCIe TLP field	Value
PRG Index	DTI_ATS_PAGE_RESP.PRG_INDEX
PASID	If DTI_ATS_PAGE_RESP.SSV is 1, the PCIe PRG Response has a PASID and the PASID value is DTI_ATS_PAGE_RESP.SSID. Otherwise, the PCIe PRG Response does not have a PASID.
T[a]	DTI_ATS_PAGE_RESP.T
Destination ID	This field is DTI_ATS_PAGE_RESP.SID[15:0]. Higher-order bits of DTI_ATS_PAGE_RESP.SID uniquely identify the PCIe segment in the StreamID space that is used by the SMMU.

a. DTI-ATSv3 and DTI-ATSv4 only

The mapping of DTI_ATS_PAGE_RESP.RESP with the PCIe PRG Response is shown in Table 4-9:

Table 4-9 Response code for PCIe PRG Response

DTI_ATS_PAGE_RESP.RESP	Response Code in PCIe PRG Response
ResponseFailure	Response Failure
InvalidRequest	Invalid Request
Success	Success

4.4.4 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. Protocol version is DTI-ATSv2 or greater.

Flow control result

None

Field descriptions

The DTI_ATS_PAGE_RESPACK bit assignments are:

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

Bits [7:4]

Reserved, SBZ

M_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.

`0b1001`   DTI_ATS_PAGE_RESPACK

4.4.5 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.*

4.5 Message dependencies for DTI-ATS

The message dependencies for the DTI-ATS protocol are shown in Figure 4-3.

In this dependency diagram:

- The light gray box indicates the messages originated from the TCU.
- The dark gray box indicates the messages originated from PCIe RP.
- The dotted line box indicates the translated transaction that is not a DTI message but is useful for analyzing the dependency.
- An arrow is a dependency from the message at the start of the arrow to the message at the end of the arrow.
- An arrow starting from message A and ending with message B represents a dependency on the DTI-ATS link between PCIe RP and TCU.
  — When two messages travel in the same direction, the "must not overtake" rules mean that if the message B is observed before message A by the sender, then the same order must be observed by the receiver.
  — When two messages travel in the opposite directions, the "must wait" rules mean that message B is expected by the sender of message A, and message B must be received by this sender before message A can be presented on DTI.
- A message dependency could be any of the following:
  — Messages that are credit controlled. For example, the maximum number of DTI_ATS_INV_ACK messages before another DTI_ATS_INV_REQ message can be issued is DTI_ATS_CONDIS_REQ.TOK_INV_GNT.
  — The messages with order requirements defined in this protocol.
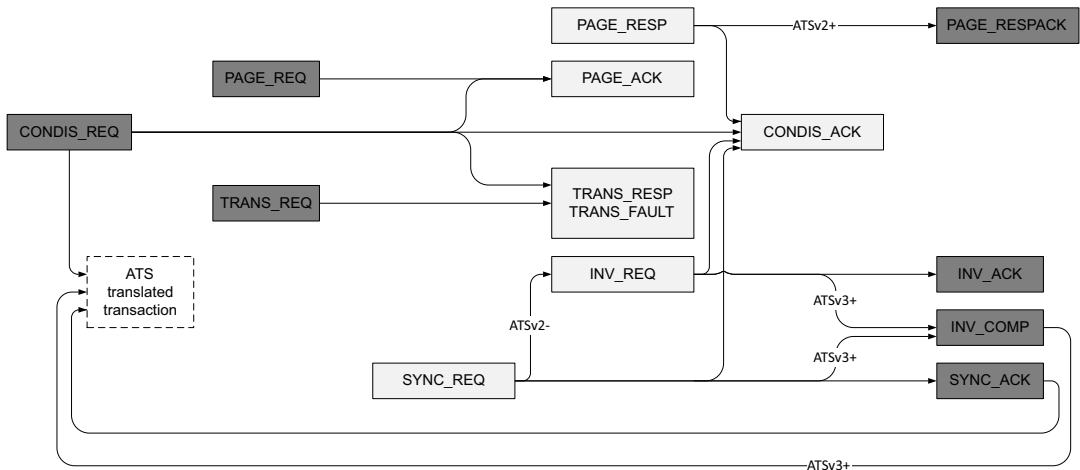
Figure 4-3 DTI-ATS message dependency sequence

The message dependencies rules are as follows:

- DTI_ATS_PAGE_REQ message must wait for any outstanding DTI_ATS_PAGE_ACK message.
- DTI_ATS_CONDIS_REQ message must wait for any outstanding DTI_ATS_CONDIS_ACK message.
- DTI_ATS_CONDIS_REQ message must wait for any outstanding DTI_ATS_PAGE_ACK messages.
- DTI_ATS_CONDIS_REQ message must wait for any outstanding DTI_ATS_TRANS_RESP or any outstanding DTI_ATS_TRANS_FAULT messages.
- DTI_ATS_CONDIS_REQ message must wait for all translated transactions to complete in the PCIe RP.
- DTI_ATS_PAGE_RESP, DTI_ATS_INV_REQ, and DTI_ATS_SYNC_REQ messages must not overtake DTI_ATS_CONDIS_ACK message.
- DTI_ATS_TRANS_REQ message must wait for outstanding DTI_ATS_TRANS_RESP and DTI_ATS_TRANS_FAULT messages to return tokens when there is no translation token left.
- DTI_ATS_INV_REQ message must wait for outstanding DTI_ATS_INV_ACK messages to return tokens when there is no invalidation token left.

- DTI_ATS_SYNC_REQ message must wait for any outstanding DTI_ATS_SYNC_ACK message.
- DTI_ATS_SYNC_ACK message can wait for translated transactions using translations obtained from DTI-ATS.
- DTI-ATSv2 or later
    — DTI_ATS_PAGE_RESP message must wait for any outstanding DTI_ATS_PAGE_RESPACK message.
- DTI-ATSv1, DTI-ATSv2
    — DTI_ATS_SYNC_REQ message must not overtake any DTI_ATS_INV_REQ messages.
- DTI-ATSv3 or later
    — DTI_ATS_INV_REQ message must wait for outstanding DTI_ATS_INV_COMP messages when it runs out of ITAG value to use. The ITAG value is required to be unique for outstanding DTI_ATS_INV_COMP messages with the same SID.
    — DTI_ATS_SYNC_REQ message must wait for any outstanding DTI_ATS_INV_COMP messages for invalidates that are in scope of the SYNC.
    — DTI_ATS_INV_COMP message can wait for translated transactions using translations obtained from DTI-ATS.

Chapter 5
Transport Layer

This chapter describes the transport layer of the DTI protocol.

It contains the following sections:

5.1　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.

5.2 AXI4-Stream transport protocol

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

This section contains the following subsections:

- *AXI4-Stream signals*
- *Interleaving* on page 5-130
- *Usage of the TID and TDEST signals* on page 5-130

5.2.1 AXI4-Stream signals

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

Table 5-1 shows the mapping of AXI4-Stream signals for the DTI protocol.

Table 5-1 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]**. It is recommended that TDATA is driven to zero for null bytes indicated by TKEEP being LOW.
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 destination of the message.
TUSER	Not implemented	The DTI protocol does not require this signal.
TWAKEUP	Interface activity indication	Wakeup signal

The signal names of the AXI4-Stream interface are given a suffix to indicate the direction of the interface they are using. Table 5-2 shows how the signals are suffixed.

Table 5-2 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.

5.2.2 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.

5.2.3 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. It is recommended 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.

Chapter 6
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 for A-profile architecture*.

It contains the following sections:

6.1 Memory attributes

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

This section contains the following subsections:

- *Memory attribute types*
- *Memory attribute decoding* on page 6-133
- *Memory attribute processing* on page 6-134

6.1.1 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
)
```

F.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;
```

6.1.3   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;
```

6.2 Cache lookup

This section contains the following subsections:

6.2.1 MatchTranslation

This section provides an example implementation for matching an incoming translation request against a translation in cache.

```
// MatchTranslation()
// ==================
// Match an incoming translation request reqIN with a cached translation formed by
// (reqC,respC). The incoming request can use the translation if the result is True.

boolean MatchTranslation(
  DTI_TBU_TRANS_REQ reqIN, // Incoming request to match with the cached translation
  DTI_TBU_TRANS_REQ reqC,  // Translation request for the cached translation
  DTI_TBU_TRANS_RESP respC // Translation response for the cached translation
)

  transrngbits = DecodeTransRng(respC.TRANS_RNG);

  bit secsid_match = MatchSECSID(reqIN, reqC);
  bit sid_match = MatchSID(reqIN, reqC, respC.CONT);
  bit ssid_match = MatchSSID(reqIN, reqC);
  bit flow_match = MatchFlow(reqIN, reqC);
  bit pas_match = reqIN.{NSE,NS} == reqC.{NSE,NS};
  bit oas_match = reqIN.IA <= OAS;
  bit addr_match = reqIN.IA[51:transrngbits] == reqC.IA[51:transrngbits];
  bit tbi_addr_match = reqIN.IA[55:transrngbits] == reqC.IA[55:transrngbits] &&
                       (respC.TBI || reqIN.IA[63:56] == reqC.IA[63:56]);

  bit hit_gpc_only_bypass_full_rng = !reqIN.MMUV && !reqC.MMUV &&
                                     TRANS_RNG == 0b1111 && oas_match;
  bit hit_gpc_only_bypass_not_full = !reqIN.MMUV && !reqC.MMUV &&
                                     TRANS_RNG != 0b1111 && oas_match &&
                                     pas_match && addr_match;

  bit hit_non_gpc_glblbyp_full_rng = reqIN.MMUV && reqC.MMUV && respC.BYPASS &&
                                     respC.BP_TYPE == GlobalBypass &&
                                     PermissionCheck(reqIN,respC) &&
                                     respC.TRANS_RNG == 0b1111 && secsid_match &&
                                     flow_match && oas_match;
  bit hit_non_gpc_glblbyp_not_full = reqIN.MMUV && reqC.MMUV && respC.BYPASS &&
                                     respC.BP_TYPE == GlobalBypass &&
                                     PermissionCheck(reqIN,repsC) &&
                                     respC.TRANS_RNG != 0b1111 && secsid_match &&
                                     flow_match && oas_match &&
                                     pas_match && addr_match;

  bit hit_streambypass_full_rng = reqIN.MMUV && reqC.MMUV && respC.BYPASS &&
                                  respC.BP_TYPE == StreamBypass &&
                                  PermissionCheck(reqIN,respC) &&
                                  respC.TRANS_RNG == 0b1111 && secsid_match &&
                                  flow_match && sid_match && ssid_match &&
                                  oas_match && pas_match;
  bit hit_streambypass_not_full = reqIN.MMUV && reqC.MMUV && respC.BYPASS &&
                                  respC.BP_TYPE == StreamBypass &&
                                  PermissionCheck(reqIN,respC) &&
                                  respC.TRANS_RNG != 0b1111 && secsid_match &&
                                  flow_match && sid_match && ssid_match &&
```

```
                                oas_match && pas_match && addr_match;

         bit hit_dptbypass_not_full = reqIN.MMUV && reqC.MMUV && respC.BYPASS &&
                                respC.BP_TYPE == DPTBypass &&
                                PermissionCheck(reqIN,respC) &&
                                secsid_match && flow_match && sid_match &&
                                oas_match && pas_match && addr_match;

         bit hit_non_bypass = reqIN.MMUV && reqC.MMUV && !respC.BYPASS && !reqIN.IDENT &&
                           PermissionCheck(reqIN,respC) &&
                           secsid_match && flow_match && sid_match && ssid_match &&
                           pas_match && tbi_addr_match;

    if hit_gpc_only_bypass_full_rng ||
       hit_gpc_only_bypass_not_full ||
       hit_non_gpc_glblbyp_full_rng ||
       hit_non_gpc_glblbyp_not_full ||
       hit_streambypass_full_rng    ||
       hit_streambypass_not_full    ||
       hit_dptbypass_not_full       ||
       hit_non_bypass then

      return True
    else
      return False
```

6.2.2 MatchFault

This section provides an example implementation for matching an incoming translation request against a fault transaction in cache.

```
// MatchFault()
// ==================
// Match an incoming translation request reqIN with a cached fault
// formed by (reqC,respC). The incoming request can use the fault
// translation if the result is True.

boolean MatchFault(DTI_TBU_TRANS_REQ   reqIN,  // Incoming request to match with the cached fault
                   DTI_TBU_TRANS_REQ   reqC,   // Translation request for the cached fault
                   DTI_TBU_TRANS_FAULT respC) // Translation response for the cached fault

  bit secsid_match   = MatchSECSID(reqIN, reqC);
  bit sid_match      = MatchSID(reqIN, reqC, respC.CONT);
  bit flow_match     = MatchFlow(reqIN, reqC);

  bit hit_globaldisabled = reqIN.MMUV && reqC.MMUV && respC.FAULT_TYPE == GlobalDisabled &&
                           secsid_match && flow_match;
  bit hit_streamdisabled = reqIN.MMUV && reqC.MMUV && respC.FAULT_TYPE == StreamDisabled &&
                           secsid_match && flow_match && sid_match;

  if hit_globaldisabled || hit_streamdisabled then
    return True
  else
    return False
```

6.2.3 PermissionCheck

```
// PermissionCheck()
// =================
// Check Non-secure instruction read permission for GlobalBypass or StreamBypass
// translations when MMUV=1. And check the RWX permissions for Non-bypass
// translation. No permission check required for a translation with MMUV=0.
boolean PermissionCheck(DTI_TBU_TRANS_REQ req, DTI_TBU_TRANS_RESP resp)
  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 s_byp_eff_ns_pas = ((resp.ATTR_OVR.NSCFG == "Use incoming") && req.NS) ||
                             (resp.ATTR_OVR.NSCFG == "Non-secure");
      bit rl_byp_eff_ns_pas = (resp.BYPASS && resp.BP_TYPE == StreamBypass) ?
                              ((resp.ATTR_OVR.NSCFG == "Use incoming") && !req.NSE) ||
                              (resp.ATTR_OVR.NSCFG == "Non-secure") :
                              !resp.NSE;
      bit req_R = ((req.PERM == "R") && !effective_InD) || (req.PERM == "RW")
      bit req_W = (req.PERM == "W") || (req.PERM == "RW")
      bit req_X = (req.PERM == "R") && effective_InD

      if (resp.BYPASS && (req.SEC_SID == Secure) && !resp.ALLOW_NSX && req_X &&
          s_byp_eff_ns_pas) ||
         (resp.BYPASS && (req.SEC_SID == Realm) && req_X && rl_byp_eff_ns_pas) ||
         (resp.BYPASS && (resp.BP_TYPE == DPTBypass) &&
          ((!resp.ALLOW_UW && req_W && !effective_PnU) ||
           (!resp.ALLOW_PW && req_W && effective_PnU))) ||
         (!resp.BYPASS && (
          ((req.SEC_SID == Realm) && !resp.NSE && (resp.ALLOW_UX || resp.ALLOW_PX)) ||
          (!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))) then
        return False
      else
        return True
```

6.2.4   Shared pseudocode

This section describes the common pseudocode.

```
// DecodeTransRng()
// =================
// Decode the address size indicated by DTI_TBU_TRANS_RESP.TRANS_RNG
int DecodeTransRng(bits(4) trans_rng)
  case trans_rng of
    when '0000'
      page_sz = 12
    when '0001'
      page_sz = 14
    when '0010'
      page_sz = 16
    when '0011'
      page_sz = 21
    when '0100'
      page_sz = 25
    when '0101'
      page_sz = 29
    when '0110'
      page_sz = 30
    when '0111'
      page_sz = 34
    when '1010'
      page_sz = 36
    when '1011'
      page_sz = 39
    when '1000'
      page_sz = 42

  return page_sz

// MatchSECSID()
// =================
// Match SECSID in two translation requests
```

```
boolean MatchSECSID(DTI_TBU_TRANS_REQ req1, DTI_TBU_TRANS_REQ req2)
  return (req1.SEC_SID == req2.SEC_SID)

// MatchSID()
// ==================
// Match SID between translation request1 and translation2 considering CONT field
boolean MatchSID(DTI_TBU_TRANS_REQ req1, DTI_TBU_TRANS_REQ req2, bits (4) cont)
  return (req1.SID[31:cont] == req2.SID[31:cont])

// MatchSSID()
// ==================
// Match SSID and SSV in two translation requests
boolean MatchSSID(DTI_TBU_TRANS_REQ req1, DTI_TBU_TRANS_REQ req2)
  return (req1.SSV == req2.SSV && (!req1.SSV || (req1.SSID == req2.SSID)))

// MatchFlow()
// ==================
// Match FLOW=ATST in two translation requests
boolean MatchFlow(DTI_TBU_TRANS_REQ req1, DTI_TBU_TRANS_REQ req2)
  return ((req1.FLOW != ATST && req2.FLOW != ATST) || (req1.FLOW == ATST && req2.FLOW == ATST))
```

ARM IHI 0088G
ID062824

Appendix A
Revisions

This appendix describes the technical changes between released issues of this specification.

Table A-1 Differences between Issue E.b and Issue E

Change	Location
Addition of DTI_ATS_PAGE_RESPACK message to the *DTI-ATS protocol downstream message* table	*DTI-ATS protocol downstream message* on page 2-26
Correction to the FLOW[1] bit of the DTI_TBU_TRANS_REQ message	*DTI_TBU_TRANS_REQ* on page 3-37
DTI_TBU_SYNC_REQ usage constraints clarification	*DTI_TBU_SYNC_REQ* on page 3-69
Clarification of the SID bits of the DTI_ATS_INV_REQ message	*DTI_ATS_INV_REQ* on page 4-107
DTI_ATS_SYNC_REQ usage constraints clarification	*DTI_ATS_SYNC_REQ* on page 4-111
DTI_ATS_PAGE_RESP usage constraints clarification	*DTI_ATS_PAGE_RESP* on page 4-120

Table A-1 Differences between Issue E.b and Issue E (continued)

Change	Location
Clarification of the SID bits of the DTI_ATS_PAGE_RESP message	*DTI_ATS_PAGE_RESP on page 4-120*
DTI_ATS_PAGE_RESPACK usage constraints clarification	*DTI_ATS_PAGE_RESPACK on page 4-122*
Correction to the M_MSG_TYPE bits of the DTI_ATS_PAGE_RESPACK message	*DTI_ATS_PAGE_RESPACK on page 4-122*

Table A-2 Differences between Issue F and Issue E.b

Change	Location
This issue describes only DTI-TBUv3, DTI-ATSv1, DTI-ATSv2, and DTI-ATSv3. For information on DTI-TBUv1 and DTI-TBUv2, see Arm Developer, https://developer.arm.com/documentation.	Throughout the specification
New message: DTI_TBU_TRANS_RESPEX message	*DTI_TBU_TRANS_RESPEX on page 3-55*
New message: DTI_ATS_INV_COMP message	*DTI_ATS_INV_COMP on page 4-110*
New feature: *Granule Protection Checks*	*Message groups of the DTI Protocol on page 2-24*
Update: Message type field names • Changed MST_MSG_TYPE to M_MSG_TYPE • Changed SLV_MSG_TYPE to S_MSG_TYPE	Throughout the specification
Update: DTI_TBU_CONDIS_REQ message	*DTI_TBU_CONDIS_REQ on page 3-32*
Update: DTI_TBU_CONDIS_ACK message	*DTI_TBU_CONDIS_ACK on page 3-34*
Update: DTI_TBU_TRANS_REQ message	*DTI_TBU_TRANS_REQ on page 3-37*
Update: DTI_TBU_TRANS_RESP message	*DTI_TBU_TRANS_RESP on page 3-41*
Update: DTI_TBU_TRANS_FAULT message	*DTI_TBU_TRANS_FAULT on page 3-57*
Update: DTI_TBU_INV_REQ message	*DTI_TBU_INV_REQ on page 3-66*
Update: DTI_TBU_SYNC_REQ message	*DTI_TBU_SYNC_REQ on page 3-69*
Update: *DTI-TBU invalidation operations* table	*DTI-TBU list of invalidation operations on page 3-73*
Update: *Range Invalidate operations* section	*Range Invalidate operations on page 3-76*
Update: *DTI-TBU encodings of INVAL_RNG* table	*DTI-TBU encodings of DTI_TBU_TRANS_RESP.INVAL_RNG on page 3-77*
New feature: *Realm Invalidation*	*Realm Invalidation on page 3-78*
New feature: *GPC invalidate operations*	*GPC invalidate operations on page 3-79*
Update: *DTI_TBU_REG_WRITE* message	*DTI_TBU_REG_WRITE on page 3-81*
Update: *DTI_TBU_REG_READ* message	*DTI_TBU_REG_READ on page 3-82*
Update: *DTI_ATS_CONDIS_REQ* message	*DTI_ATS_CONDIS_REQ on page 4-88*

Table A-2 Differences between Issue F and Issue E.b (continued)

Change	Location
Update: *DTI_ATS_CONDIS_ACK* message	*DTI_ATS_CONDIS_ACK* on page 4-90
Update: *DTI_ATS_TRANS_REQ* message	*DTI_ATS_TRANS_REQ* on page 4-94
Update: DTI_ATS_TRANS_RESP message	*DTI_ATS_TRANS_RESP* on page 4-97
Update: DTI_ATS_INV_REQ message	*DTI_ATS_INV_REQ* on page 4-107
Update: DTI_ATS_SYNC_REQ message	*DTI_ATS_SYNC_REQ* on page 4-111
Update: DTI_ATS_SYNC_ACK message	*DTI_ATS_SYNC_ACK* on page 4-112
Update: *DTI-ATS invalidation sequence* diagram	*The DTI-ATS invalidation sequence* on page 4-113
Update: *DTI_ATS_PAGE_REQ* message	*DTI_ATS_PAGE_REQ* on page 4-117
Update: *DTI_ATS_PAGE_RESP* message	*DTI_ATS_PAGE_RESP* on page 4-120
Update: *Mapping of AXI4-Stream to the DTI protocol* table	*Mapping of AXI4-Stream to the DTI protocol* on page 5-129
Update: Pseudocode appendix	*Memory attributes* on page 6-132 and *Cache lookup* on page 6-138
Removed: *DTI-TBU Caching Model* chapter	-
Addition: DTI-TBU message dependencies	*Message dependencies for DTI-TBU* on page 3-85
Addition: DTI-ATS message dependencies	*Message dependencies for DTI-ATS* on page 4-124
Addition: DTI-ATS and PCIe TLP mappings for DTI_ATS_TRANS_REQ, DTI_ATS_INV_REQ, DTI_ATS_PAGE_REQ, and DTI_ATS_PAGE_RESP	• *DTI_ATS_TRANS_REQ* on page 4-94 • *DTI_ATS_INV_REQ* on page 4-107 • *DTI_ATS_PAGE_REQ* on page 4-117 • *DTI_ATS_PAGE_RESP* on page 4-120
Addition: • Cache lookup process	*Cache lookup process* on page 3-65

Table A-3 Differences between Issue G and Issue F

Change	Location
This issue describes only DTI-TBUv3, DTI-TBUv4, DTI-ATSv1, DTI-ATSv2, DTI-ATSv3, and DTI-ATSv4. For information on DTI-TBUv1 and DTI-TBUv2, see Arm Developer, https://developer.arm.com/documentation.	Throughout the specification
Support for DTI-TBUv4	• *DTI_TBU_CONDIS_REQ* on page 3-32 • *DTI_TBU_CONDIS_ACK* on page 3-34
Support for DTI-ATSv4	• *DTI_ATS_CONDIS_REQ* on page 4-88 • *DTI_ATS_CONDIS_ACK* on page 4-90 • *DTI_ATS_TRANS_REQ* on page 4-94 • *DTI_ATS_TRANS_RESP* on page 4-97 • *DTI_ATS_TRANS_FAULT* on page 4-103
Update • ALLOW_PW, ALLOW_UW, and BP_TYPE bits for DPTBypass response support • DCP, DRE bits for bypass response • Addition of the following table: — Summary of DTI-TBUv4 permitted translation response contexts when DTI_TBU_CONDIS_REQ.STAGES = MG • Physical Address Space constraints • NSCFG field	• *DTI_TBU_TRANS_RESP* on page 3-41 • *Matching field values for future transactions* on page 3-52 • *Summary of DTI-TBUv3 permitted translation response contexts when DTI_TBU_CONDIS_REQ.STAGES = MG* on page 3-52 • *Summary of DTI-TBUv4 permitted translation response contexts when DTI_TBU_CONDIS_REQ.STAGES = MG* on page 3-53 • *The NSCFG field* on page 3-63
Update: • Relaxation of MECID constraint • PARTID extension	• *DTI_TBU_TRANS_RESPEX* on page 3-55 • *DTI_TBU_TRANS_RESP* on page 3-41
Correction: DTI_TBU_TRANS_REQ.IA upper bits check	*Additional rules on permitted translation responses* on page 3-59
Addition: DPT invalidation section	• *DPT invalidation* on page 3-80 • *DTI-TBU list of invalidation operations* on page 3-73
Clarification on rules when IA is out of range	*Rules when IA out of range* on page 3-59
Correction and clarification on ATS global invalidate	• *DTI_ATS_INV_REQ mapping to PCIe Invalidate Request* on page 4-109 • *CMD_ATC_INV command mapping to DTI_ATS_INV_REQ* on page 4-116
Correction and clarification: Message dependencies for DTI-TBU section	*Message dependencies for DTI-TBU* on page 3-85
Clarification: SUP_PRI bit of DTI_ATS_CONDIS_ACK message	*DTI_ATS_CONDIS_ACK* on page 4-90
Correction and clarification: Message dependencies for DTI-ATS section	*Message dependencies for DTI-ATS* on page 4-124
Pseudocode changes	• *MatchTranslation* on page 6-138 • *PermissionCheck* on page 6-139