Arm® AMBA® Distributed Translation Interface (DTI) Protocol Specification

Revision: r0p0

Edition 3

arm

100225_0000_03_en

Arm® AMBA® Distributed Translation Interface (DTI) Protocol Specification

Edition 3

Release Information

Document History

Issue	Date	Confidentiality	Change
0000-00	18 November 2016	Non-Confidential	Edition 0 (First release).
0000-01	09 May 2017	Non-Confidential	Edition 1.
0000-02	11 September 2017	Non-Confidential	Edition 2.
0000-03	13 July 2018	Non-Confidential	Edition 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

Non-Confidential Proprietary Notice

This document is NON-CONFIDENTIAL and any use by you is subject to the terms of this notice and the Arm AMBA Specification Licence set about below.

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm Limited ("Arm"). **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version shall prevail.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement specifically covering this document with Arm, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms.

In this document, where the term Arm is used to refer to the company it means "Arm or any of its subsidiaries as appropriate".

Arm 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 THE RELEVANT AMBA SPECIFICATION ACCOMPANYING THIS LICENCE. ARM IS ONLY WILLING TO LICENSE THE RELEVANT AMBA SPECIFICATION TO YOU ON CONDITION THAT YOU ACCEPT ALL OF THE TERMS IN THIS LICENCE. BY CLICKING "I AGREE" OR OTHERWISE USING OR COPYING THE RELEVANT AMBA SPECIFICATION YOU INDICATE THAT YOU AGREE TO BE BOUND BY ALL THE TERMS OF THIS LICENCE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENCE, ARM IS UNWILLING TO LICENSE THE RELEVANT AMBA SPECIFICATION TO YOU AND YOU MAY NOT USE OR COPY THE RELEVANT AMBA SPECIFICATION AND YOU SHOULD PROMPTLY RETURN THE RELEVANT AMBA SPECIFICATION TO ARM.

"LICENSEE" means You and your Subsidiaries.

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

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

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

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

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

2. LICENSEE hereby agrees that the licence granted in Clause 1 is subject to the following restrictions:

(i) where a product created under Clause 1(i) is an integrated circuit which includes a CPU then either: (a) such CPU shall only be manufactured under licence from Arm; or (b) such CPU is neither substantially compliant with nor marketed as being compliant with the Arm instruction sets licensed by Arm from time to time;

(ii) the licences granted in Clause 1(iii) shall not extend to any portion or function of a product that is not itself compliant with part of the relevant AMBA Specification; and

(iii) no right is granted to LICENSEE to sublicense the rights granted to LICENSEE under this Agreement.

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

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

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

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

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

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

Contents
Arm® AMBA® Distributed Translation Interface (DTI) Protocol Specification Edition 3

Preface

This preface introduces the *Arm® AMBA® Distributed Translation Interface (DTI) Protocol Specification Edition 3*.

It contains the following:

About this book

Intended audience

This specification is intended for the following audiences:

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

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction
This chapter introduces the DTI protocol.

Chapter 2 DTI Protocol Overview
This chapter provides an overview of the DTI protocol.

Chapter 3 DTI-TBU Messages
This chapter describes the message groups of the DTI-TBU protocol.

Chapter 4 DTI-TBU Caching Model
This chapter describes the caching model for the DTI-TBU protocol.

Chapter 5 DTI-ATS Messages
This chapter describes the message groups of the DTI-ATS protocol.

Chapter 6 Transport Layer
This chapter describes the transport layer of the DTI protocol.

Appendix A Revisions
This appendix describes the changes between released issues of this book.

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

Glossary

The Glossary is a list of terms used in documentation, together with definitions for those terms. The Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the *Arm® Glossary* for more information.

Typographic conventions

italic
Introduces special terminology, denotes cross-references, and citations.

bold
Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace`
Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

`mono`space
Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic`
Denotes arguments to monospace text where the argument is to be replaced by a specific value.

`monospace bold`
Denotes language keywords when used outside example code.

`<and>`
Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

SMALL CAPITALS
Used in body text for a few terms that have specific technical meanings, that are defined in the *Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Timing diagrams

The following figure explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

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

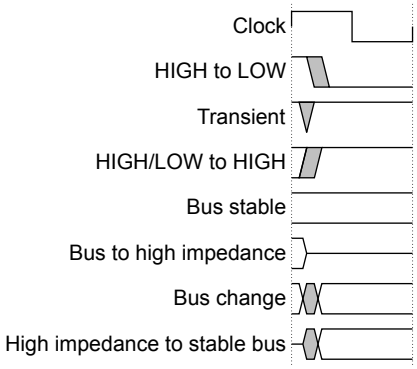


Figure 1 Key to timing diagram conventions

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.

Additional reading

This book contains information that is specific to this product. See the following documents for other relevant information.

Arm publications
- *Arm® Architecture Reference Manual Arm®v8, for Arm®v8-A architecture profile* (ARM DDI 0487).
- *Arm® System MMUv3 (SMMUv3) Architecture Specification* (ARM IHI 0070).
- *Arm® AMBA® 4 AXI4-Stream Protocol Specification* (ARM IHI 0051).

Other publications

- *PCI Express Base Specification Revision 4.0*.
- *PCI Express Address Translation Services Revision 1.1*.

Feedback

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to *errata@arm.com*. Give:

- The title *Arm AMBA Distributed Translation Interface (DTI) Protocol Specification Edition 3*.
- The number 100225_0000_03_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

——— **Note** ———

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

———————————

Chapter 1
Introduction

This chapter introduces the DTI protocol.

It contains the following section:

1.1 About the protocol

This section introduces the DTI protocol and describes the components of a DTI-compliant implementation.

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

- A *Translation Control Unit* (TCU) that performs translation table walks and implements the SMMUv3 programmers' model.
- At least one *Translation Buffer Unit* (TBU). The TBU intercepts transactions in need of translation and translates the addresses of those transactions. The TBU requests translations from the TCU and caches those translations for use by other transactions. The TCU communicates with the TBU to invalidate cached translation when necessary.
- A *PCI Express* (PCIe) *Root Complex* bus master with *Address Translation Services* (ATS) support. For more information, see the *PCI Express Base Specification Revision 4.0* and *PCI Express Address Translation Services Revision 1.1* specifications. 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 masters and the TCU.

This specification outlines two protocols that have different purposes:

- Communication between a TBU and a TCU, where the protocol is DTI-TBU.
- Communication between a PCIe Root Complex and a TCU, where the protocol is DTI-ATS.

These two protocols are collectively termed the DTI protocol.

The DTI protocol is a point-to-point protocol. Each channel consists of a link, a DTI master, and a DTI slave. The DTI masters in the respective protocols are:

- The TBU, in the DTI-TBU protocol.
- The PCIe Root Complex, in the DTI-ATS protocol.

The DTI Slave in both DTI-TBU and DTI-ATS is the TCU.

Components using the SMMU must provide correct context information to the DTI master so that it can request translations using the correct StreamID and SubstreamID. For ATS translated transactions, a PCIe master must provide additional information.

The following figure shows an example SMMU system that implements DTI.

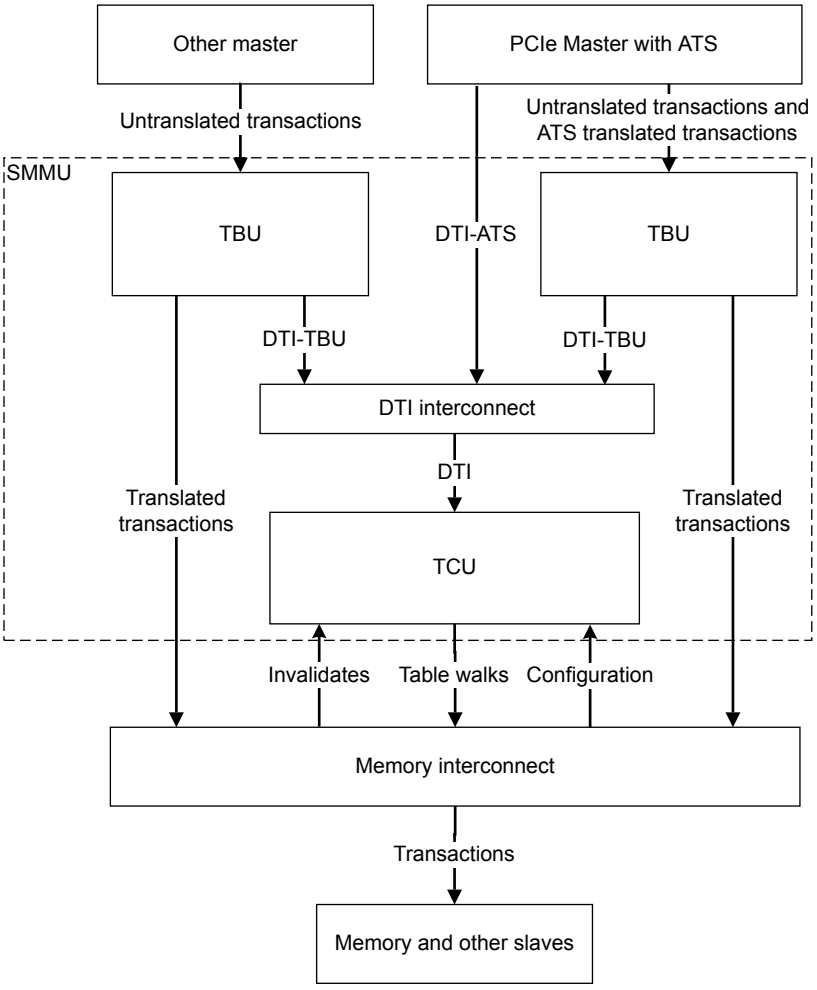


Figure 1-1 An example SMMU system

This figure includes the necessary components of a DTI-compliant implementation. However, DTI connections can cover large distances across an SoC, and so most implementations do not include a standalone SMMU component. DTI allows an implementation to distribute the functions of the SMMU across the chip, with TBUs located close to the bus masters that require translation.

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

- A bus master to incorporate advanced or specialized prefetching or translation caching requirements that cannot be met by a general-purpose TBU design.
- A bus master that requires a fully coherent connection to the memory interconnect and requires very low latency translation. For fully coherent operations, all caches in the bus master 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 bus master.

1.1.1 DTI Protocol Specification Terminology

This document uses the following terms and abbreviations.

StreamID

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

SubstreamID
A SubstreamID might optionally be provided to an SMMU implementing stage 1 translation. The SubstreamID differentiates streams of traffic originating from the same logical block in order to associate different application address translations to each.

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

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

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

VA
Virtual address.

IPA
Intermediate physical address.

PA
Physical address.

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

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

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.

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

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.

E2H
EL2 Host Mode. The Virtualization Host Extensions, introduced in *Arm Architecture Reference Manual ARMv8, for ARMv8-A architecture profile issue B*, extend the EL2 translation regime providing ASID-tagged translations.

Chapter 2
DTI Protocol Overview

This chapter provides an overview of the DTI protocol.

It contains the following sections:

2.1 DTI protocol messages

The following section gives an overview of the DTI protocol messages.

This section contains the following subsections:

2.1.1 Message groups

DTI protocol messages are grouped according to function. The following table shows the DTI message groups.

Table 2-1 Message groups of the DTI Protocol

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

2.1.2 Message listing

DTI messages are a fixed length and are 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.

Certain message types include a protocol field and the message is identified by the combination of its message-type and protocol field values.

The message-type encodings are defined independently for messages originating from:

- DTI masters
- DTI slaves.

Note

Some encoding combinations are common to both encoding spaces.

DTI-TBU protocol master-initiated messages

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

Table 2-2 DTI-TBU protocol master-initiated messages

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

DTI-TBU protocol slave-initiated messages

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

Table 2-3 DTI-TBU protocol slave-initiated messages

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

DTI-ATS protocol master-initiated messages

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

Table 2-4 DTI-ATS protocol master-initiated messages

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

DTI-ATS protocol slave-initiated message

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

Table 2-5 DTI-ATS protocol slave-initiated messages

Message group	Message	SLV_MSG_TYPE field encoding	Message length in bits
Connection and disconnection.	DTI_ATS_CONDIS_ACK	0x0	32
Translation request.	DTI_ATS_TRANS_FAULT	0x1	32
Translation request.	DTI_ATS_TRANS_RESP	0x2	160
Invalidation and synchronization.	DTI_ATS_INV_REQ	0xC	128
Invalidation and synchronization.	DTI_ATS_SYNC_REQ	0xD	8
Page request.	DTI_ATS_PAGE_ACK	0x8	8
Page request.	DTI_ATS_PAGE_RESP	0x9	96
IMPLEMENTATION DEFINED.	-	0xE	-
IMPLEMENTATION DEFINED.	-	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 which are designed to expect them when in permitted channel states, see *2.2.1 Channel states* on page 2-21. 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 token:

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

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

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

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

2.1.4 Reserved fields

Reserved fields in messages are described as *Should Be Zero* (SBZ).

The recipient of a message with reserved fields must ignore these fields. This specification recommends that the sender drive a reserved field to 0.

2.1.5 IMPLEMENTATION DEFINED fields

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

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

2.2 Managing DTI connections

This section describes the management of DTI connections.

This section contains the following subsections:

2.2.1 Channel states

The four possible states of the DTI channel are:

DISCONNECTED
The DTI master might be powered down. A DTI slave must always be able to accept a Connect Request whenever a DTI master 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 DTI master has issued a Connect Request. The DTI slave must provide an appropriate handshaking response to either establish or reject the connection.

CONNECTED
The channel is connected. The DTI master and DTI slave can issue messages as permitted by the protocol rules.

REQ_DISCONNECT
The DTI master has issued a Disconnect Request. The DTI slave issues a Disconnect Accept in response.

2.2.2 Handshaking

On powerup, the channel between the DTI master and the DTI slave is initially in the DISCONNECTED state. The following figure shows how the channel state of the channel changes in response to connect and disconnect messages.

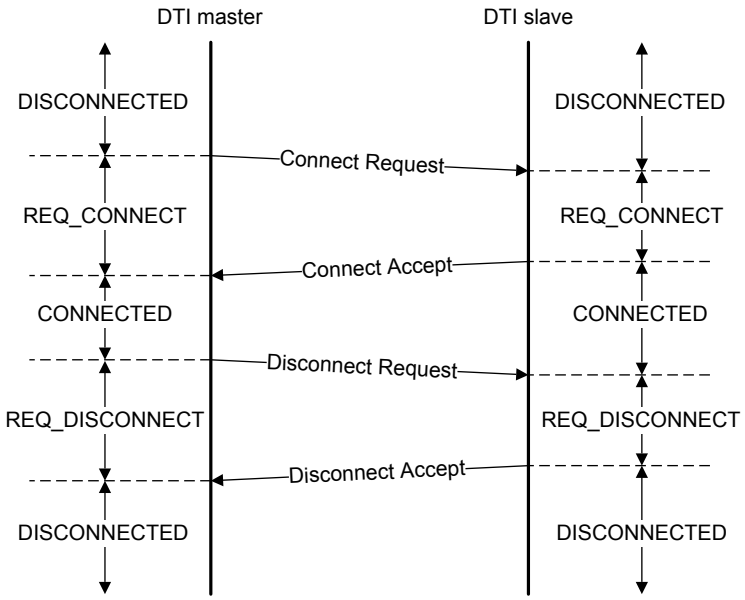


Figure 2-1 Handshake accept

Alternatively, a Connect Request might be denied, as shown in the following diagram.

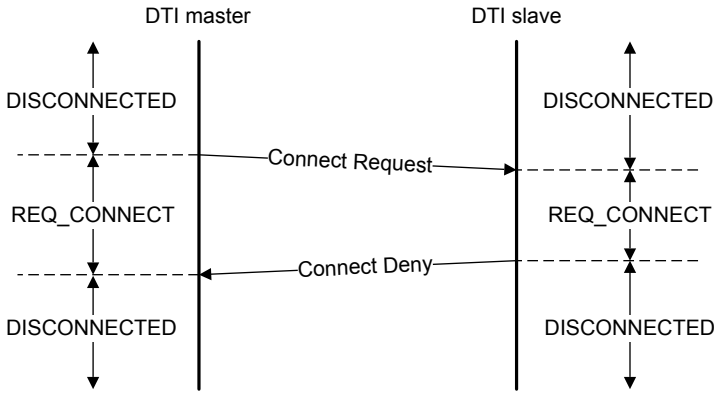


Figure 2-2 Handshake deny

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

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

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

Channel state	DTI master permitted messages	DTI slave 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 DTI master enters DISCONNECTED state and invalidates its caches.
- Outstanding register access responses, DTI_TBU_REG_RDATA or DTI_TBU_REG_WACK, are not returned.
- The DTI master must continue to accept protocol appropriate requests from the DTI slave. No response is given to the requests, and they can be ignored.

2.2.3 State restoration

For the DTI-TBU protocol, when the DTI-TBU master enters the DISCONNECTED state, all state information is lost, including cache and register contents. The DTI-TBU master must invalidate its caches before entering CONNECTED state. The DTI slave must reinitialize any necessary register contents after the connection handshake.

For the DTI-ATS protocol, there is no state restoration. 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 DTI masters to a DTI slave

A DTI channel is a point-to-point link between a single DTI master and a single DTI slave. If a DTI slave is connected to multiple physical DTI masters using a single interface, then each DTI master has its own DTI channel.

Therefore:

- If a DTI slave is required to send a message to multiple masters, 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

This section describes the connection and disconnection message group.

The connection and disconnection of the DTI-TBU protocol channel might require state restoration, for more information see *2.2.3 State restoration* on page 2-22.

This section contains the following subsections:

3.1.1 DTI_TBU_CONDIS_REQ

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

Description

Connection state change request.

Source

DTI master.

Usage constraints

The DTI-TBU master 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 DTI-TBU master 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
Reserved							SUP_REG	24
TOK_INV_GNT				TOK_TRANS_REQ[7:4]				16
TOK_TRANS_REQ[3:0]				VERSION				8
IMP DEF	Reserved	PROTOCOL	STATE	MST_MSG_TYPE				0

Bits [31:25]

Reserved, SBZ.

SUP_REG, bits [24]

This field indicates when register accesses are supported.

0 Register accesses are not supported.

1 Register accesses are supported.

When STATE is 1 and the value of this bit is 0, the DTI slave 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, bits [19:12]

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

When 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.

When 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.

VERSION, bits [11:8]

This field identifies the requested protocol version.

0000 DTI-TBUv1.

All other encodings are reserved.

A DTI-TBU master can request any protocol version it supports. Only DTI-TBUv1 is currently defined, however a DTI-TBU slave must accept requests for later protocol versions. The DTI_TBU_CONDIS_ACK message indicates the protocol version to use.

IMPLEMENTATION DEFINED, bit [7]

IMPLEMENTATION DEFINED.

Bits [6]

Reserved, SBZ.

PROTOCOL, bit [5]

This bit identifies the protocol that is used by this DTI master.

0 DTI-TBU.

This bit must be 0.

STATE, bit [4]

This bit identifies the new channel state requested.

0 Disconnect request.

1 Connect request.

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

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

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for master-initiated messages, see *DTI-TBU protocol master-initiated messages* on page 2-18.

0000 DTI_TBU_CONDIS_REQ.

3.1.2 DTI_TBU_CONDIS_ACK

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

Description

A connection state change acknowledgement.

Source

DTI slave.

Usage constraints

The DTI master 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
Reserved							OAS[3]	24
OAS[2:0]			Reserved	TOK_TRANS_GNT[7:4]				16
TOK_TRANS_GNT[3:0]				VERSION				8
IMP DEF	Reserved		STATE	SLV_MSG_TYPE				0

Bits [31:25]

Reserved, SBZ.

OAS, bits [24:21]

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

0000 32 bits (4GB).

0001 36 bits (64GB).

0010 40 bits (1TB).

0011 42 bits (4TB).

0100 44 bits (16TB).

0101 48 bits (256TB).

0110 52 bits (4PB).

All other values are Reserved.

Bit [20]

Reserved, SBZ.

TOK_TRANS_GNT, bits [19:12]

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

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

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

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

VERSION, bits [11:8]

The protocol version that is granted by the DTI slave.

0000 DTI-TBUv1.

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

IMPLEMENTATION DEFINED, bit [7]

IMPLEMENTATION DEFINED.

Bits [6:5]

Reserved, SBZ.

STATE, bit [4]

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

0 DISCONNECTED.

1 CONNECTED.

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

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

SLV_MSG_TYPE, bits [3:0]

Identifies the message type. The value of this field is taken from the list of encodings for slave-initiated messages, see *DTI-TBU protocol slave-initiated messages* on page 2-18.

0000 DTI_TBU_CONDIS_ACK.

3.2 Translation request message group

This section describes the translation request message group. The DTI-TBU translation request messages enable the DTI-TBU master to find the translation for a given transaction, or prefetch a translation. The DTI slave responds with either a successful translation or a fault.

This section contains the following subsections:

3.2.1 DTI_TBU_TRANS_REQ

The DTI_TBU_TRANS_REQ message is used to initiate a translation request.

Description

A translation request.

Source

DTI master.

Usage constraints

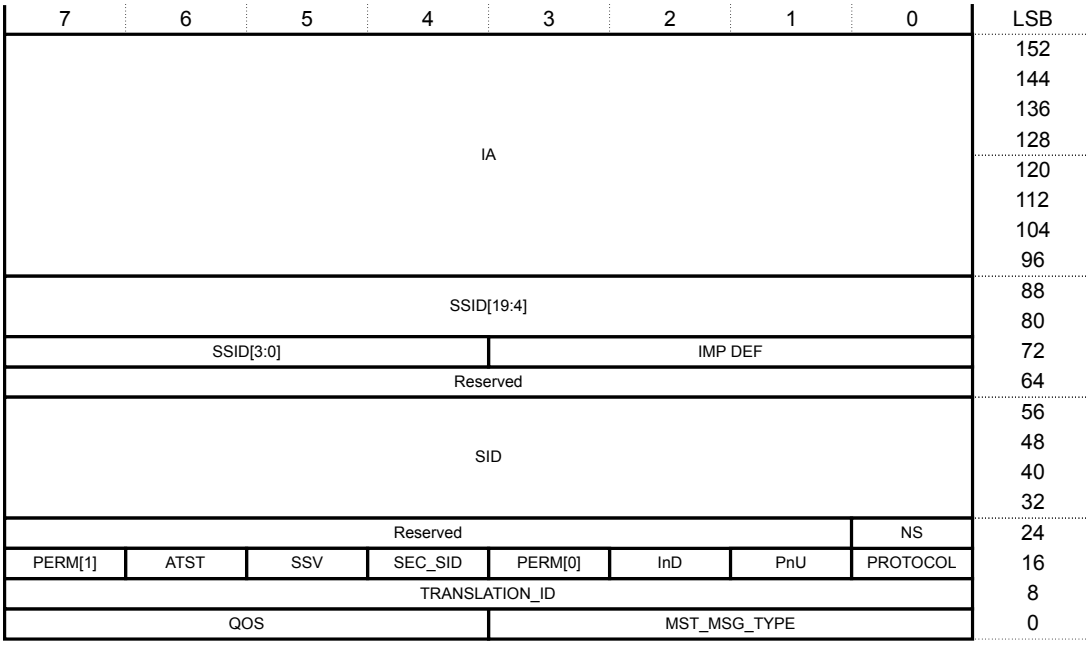
The DTI master must have at least one translation token.

Flow control result

The DTI master consumes a translation token.

Field descriptions

The DTI_TBU_TRANS_REQ bit assignments are:



IA, bits [159:96]

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

SSID, bits [95:76]

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

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

IMPLEMENTATION DEFINED, bit [75:72]

IMPLEMENTATION DEFINED.

Bits [71:64]

Reserved, SBZ.

SID, bits [63:32]

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

Bits [31:25]

Reserved, SBZ.

NS, bit [24]

This bit indicates the security level of the transaction.

0 Secure.

1 Non-secure.

Must be 1 if SEC_SID=0.

PERM[1], bit [23]

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

The encoding of PERM[1:0] is:

00 W: Write permission required.

01 R: Read permission required.

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

10 RW: Read and write permission required.

——— **Note** ———

The only change in behavior between Edition 1 and Edition 2 is the addition of the RW encoding of this field. The other encodings behave identically to the previous SPECULATIVE and RnW fields.

———————————

ATST, bit [22]

This bit indicates whether the transaction is ATS-translated.

0 Not ATS-translated.

1 ATS-translated.

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

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 ATST is 1, this bit must be 0.

SEC_SID, bit [20]

This bit indicates whether the StreamID is Secure.

0 Non-secure StreamID.

1 Secure StreamID.

When the value of ATST is 1, this bit must be 0.

PERM[0], bit [19]

See PERM[1], bit [23].

InD, bit [18]

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

0 Data access.

1 Instruction access.

When the value of REQ.PERM[1:0] is W, RW or SPEC, 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 REQ.PERM[1:0] is SPEC, 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, bits [15:8]

This field gives the identification number of this translation.

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

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

QOS, bits [7:4]

This field indicates the Quality of Service priority level.

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

This field is a hint.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for master-initiated messages, see *DTI-TBU protocol master-initiated messages* on page 2-18.

0010 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 DTI slave 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 *3.2.4 Faulting expressions of the translation request message* on page 3-47.

Description

A DTI translation response.

Source

DTI slave.

Usage constraints

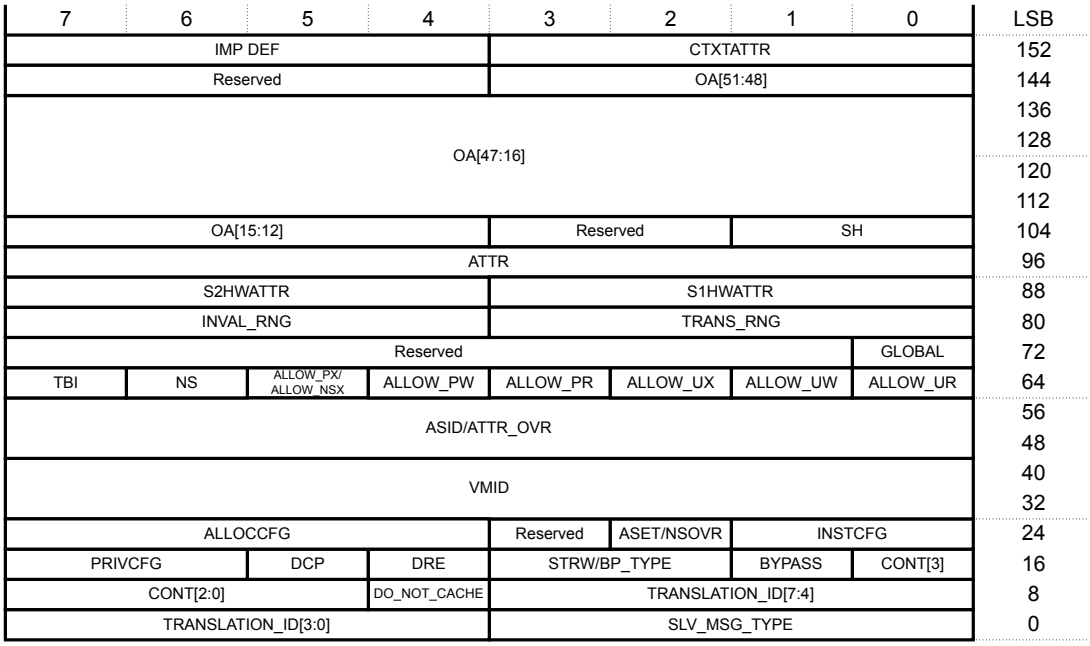
The DTI master must have previously issued a translation request that has not yet generated either a translation response or a fault message.

Flow control result

The DTI slave returns a translation token to the DTI master.

Field descriptions

The DTI_TBU_TRANS_RESP bit assignments are:



IMPLEMENTATION DEFINED, bit [159:156]

IMPLEMENTATION DEFINED.

CTXTATTR, bits [155:152]

This field gives IMPLEMENTATION DEFINED attributes for the translation context.

Bits [151:148]

Reserved, SBZ.

OA, bits [147:108]

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

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

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.

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

Bits [107:106]

Reserved, SBZ.

SH, bits[105:104]

This field indicates the shareability of the translation.

00 Non-shareable.

01 Reserved.

10 Outer-shareable.

11 Inner-shareable.

Note

This value represents the Shareability attribute which is stored in the page tables. In some cases the resulting Shareability of the translation might be different from the value that is shown here. For more information, see *Consistency check on combination of translation attributes* on page 3-51.

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 follows:

0000 Device memory. See encoding of bits [99:96] for the device memory type.

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

0100 Normal Memory, Outer Non-Cacheable.

01RW When RW is not 00 this field is Normal Memory, Outer Write-back transient.

10RW Normal Memory, Outer Write-through non-transient.

11RW 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]:

Bits [99:96]	When [103:100] is 0000	When [103:100] is not 0000
0000	Device-nGnRnE memory.	Reserved.
00RW, RW is not 00	Reserved.	Normal Memory, Inner Write-through transient.
0100	Device-nGnRE memory	Normal memory, Inner Non-cacheable.
01RW, RW is not 00	Reserved.	Normal Memory, Inner Write-back transient.
1000	Device-nGRE memory	Normal Memory, Inner Write-through non-transient. (RW=00)
10RW, RW is not 00	Reserved.	Normal Memory, Inner Write-through non-transient.

(continued)

Bits [99:96]	When [103:100] is 0000	When [103:100] is not 0000
1100	Device-GRE memory.	Normal Memory, Inner Write-back non-transient. (RW=00)
11RW, RW is not 00	Reserved.	Normal Memory, Inner Write-back non-transient.

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

The R and W bits have the following encoding:

0 Do not allocate.

1 Allocate.

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

S2HWATTR, bits [95:92]

This field gives the IMPLEMENTATION DEFINED stage 2 hardware attributes.

These attributes are provided in the stage 2 page tables for IMPLEMENTATION DEFINED purposes. Bits which are not enabled for hardware use must be 0.

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

The value of this field must be 0 if either of the following conditions are true:
- The value of BYPASS is 1.
- The value of BYPASS is 0 and either:
  - The value of SEC_SID is 1.
  - The value of STRW is either EL2 or EL3.

S1HWATTR, bits [91:88]

This field gives the IMPLEMENTATION DEFINED stage 1 hardware attributes.

These attributes are provided in the stage 1 page tables for IMPLEMENTATION DEFINED purposes. Bits which are not enabled for hardware use must be 0.

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

The value of this field must be 0 if either of the following conditions are true:
- The value of BYPASS is 1.
- The value of BYPASS is 0 and value of STRW is EL1-S2.

INVAL_RNG, bits [87:84]

This field indicates the range of addresses for invalidation.

0000	4KB.
0001	16KB.
0010	64KB.
0011	2MB.
0100	32MB.
0101	512MB.
0110	1GB.
1000	4TB.

All other values are Reserved.

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

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

If an invalidation request is received, this translation must be invalidated under the following conditions:

- 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.

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

TRANS_RNG, bits [83:80]

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

When BYPASS=0

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

0000	4KB.
0001	16KB.
0010	64KB.
0011	2MB.
0100	32MB.
0101	512MB.
0110	1GB.
0111	16GB.
1000	4TB.
1001	128TB.

All other values are Reserved.

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

This field must not indicate a size of 4TB or 128TB unless the OAS field of the DTI_TBU_CONDIS_ACK message received during the connection sequence indicates a size of 52 bits.

When BYPASS=1

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

0000	32 bits (4GB).
0001	36 bits (64GB).
0010	40 bits (1TB).
0011	42 bits (4TB).
0100	44 bits (16TB).

0101 48 bits (256TB).

0110 52 bits (4PB).

All other values are Reserved.

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

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

This is a static property of the system, until the link is disconnected every translation in which the value of the BYPASS field is 1 must return the same value for this field. This field must show a range large enough to contain the IA of the transaction. For example, if DTI_TBU_TRANS_REQ.IA=0x0000_0001_0000_0000 or greater, this field cannot show a range of 32 bits (4GB).

Bits [79:73]

Reserved, SBZ.

GLOBAL, bit [72]

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

0 Non-global.

1 Global.

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

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

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

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

TBI, bit [71]

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

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

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

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

NS, bit [70]

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

0 Secure.

1 Non-secure.

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

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

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

ALLOW_PX, bit [69] when BYPASS=0

This bit indicates permissions for privileged instruction reads.

0 Not permitted.

1 Permitted.

ALLOW_NSX, bit [69] when BYPASS=1

This bit indicates permissions for Non-secure instruction reads.

0 Not permitted.

1 Permitted.

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

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

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

ALLOW_PW, bit [68]

This bit indicates permissions for privileged data write accesses.

0 Not permitted.

1 Permitted.

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

ALLOW_PR, bit [67]

This bit indicates permissions for privileged data read accesses.

0 Not permitted.

1 Permitted.

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

ALLOW_UX, bit [66]

This bit indicates permissions for unprivileged instruction reads.

0 Not permitted.

1 Permitted.

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

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

ALLOW_UW, bit [65]

This bit indicates permissions for unprivileged data write accesses.

0 Not permitted.

1 Permitted.

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

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

ALLOW_UR, bit [64]

This bit indicates permissions for unprivileged data read accesses.

0 Not permitted.

1 Permitted.

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

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

ASID/ATTR_OVR, bits [63:48]

This field is 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.

This field is ASID when the following condition is met:

- 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.

———————————

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

This field might be combined with the ATTR and SH field to give different values for the attributes of this translation. For more information about this and the subfields of this field, see *3.2.5 Calculating transaction attributes* on page 3-48.

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

VMID, bits [47:32]

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

This field must be 0 when BYPASS is 0 and any of the following is true:

- The value of SEC_SID in the translation request is 1.
- The value of STRW is either EL2 or EL3.

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 see *3.2.5 Calculating transaction attributes* on page 3-48.

Bit [27]

Reserved, SBZ.

ASET, bit [26] when BYPASS = 0

This bit indicates the shareability of the ASID set.

0 Shared set.

1 Non-shared set.

——— **Note** ———

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

———————————

This field is reserved, SBZ, when BYPASS is 1.

NSOVR, bit [26] when BYPASS = 1

This bit indicates the Non-secure bit override.

0 Use the upstream NS value.

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

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

INSTCFG, bits [25:24]

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

00 Use incoming.

01 Reserved.

10 Data.

11 Instruction.

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

PRIVCFG, bits [23:22]

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

00 Use incoming.

01 Reserved.

10 Unprivileged.

11 Privileged.

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

DCP, bit[21]

This bit indicates whether directed cache prefetch hints are permitted.

0 Not permitted.

1 Permitted.

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

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

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

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

DRE, bit[20]

This bit indicates whether destructive reads are permitted.

0 Not permitted.

1 Permitted.

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

——— **Note** ———

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

———————————

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

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

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

The bit indicates the SMMU StreamWorld, which is the Exception level that is used by the translation context.

00 EL1.

01 EL1-S2.

10 EL2.

11 EL3.

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

- When the value of SEC_SID is 0, this field is not permitted to be EL3
- When the value of SEC_SID is 1, the field is permitted to be EL1 or EL3.
- When the value of ATST is 1, this field must be EL1-S2.
- When the value of SSV is 1, this field must not be EL1-S2.

——— **Note** ———

In AArch32, all Secure transactions use EL1.

This field is EL1-S2 when stage 2 translation is enabled but stage 1 translation is disabled. In this case, the IA field in the translation request holds an IPA instead of a VA.

DTI does not indicate whether EL2-E2H is supported and enabled by the SMMU. If E2H is not enabled, then the privileged permission bits [69:67] are the same as the unprivileged permission bits [66:64], and the following fields are set to the given values:

- GLOBAL is set to 1.
- ASID is set to 0.

———————————

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

This bit indicates the scope of this translation.

00 Reserved.

01 GlobalBypass.

10 StreamBypassNoSSV.

11 Reserved.

The following table shows the fields of the translation request that must match for this translation to apply to future transactions.

Table 3-1 Matching field values for future transactions

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

This field must not be StreamBypassNoSSV if SSV = 1 in the translation request.

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 an ATST field set to 1 and with the ATSCHK bit of the SMMU set to clear.

BYPASS, bit [17]

This field indicates whether translation is bypassed.

0 Normal translation.

1 Translation bypassed.

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

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

CONT, bits [16:13]

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

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

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

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

DO_NOT_CACHE, bit [12]

This bit indicates to the DTI master when not to cache a translation.

0 Can be cached.

1 Must not be cached.

When the value of this bit is 0, the translation has not been invalidated before this message was sent.

When the value of this bit is 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 DTI-TBU master can use this field to simplify invalidation, by not caching any translations that have a value of 1 for this field.

———————————

TRANSLATION_ID, bits [11:4]

This field gives the identification number for the translation.

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

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for slave-initiated messages, see *DTI-TBU protocol slave-initiated messages* on page 2-18.

0010 DTI_TBU_TRANS_RESP.

3.2.3 DTI_TBU_TRANS_FAULT

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

Description

A translation fault response.

Source

DTI slave.

Usage constraints

The DTI master 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 DTI slave returns a translation token to the DTI master.

Field descriptions

The DTI_TBU_TRANS_FAULT bit assignments are:

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

Bits [31:19]

Reserved, SBZ.

FAULT_TYPE, bits [18:17]

This bit indicates to the DTI master how to handle the fault.

00 NonAbort.

01 Abort.

10 StreamDisabled.

11 GlobalDisabled.

If the value of ATST in the translation request was 1, this field must not be GlobalDisabled.

When the value of REQ.PERM[1:0] is SPEC, this field must not indicate an Abort.

When the value of this field is NonAbort, the transaction is handled as follows, depending on REQ.PERM[1:0]:

- When REQ.PERM[1:0] is R or RW: Return read data of 0.
- When REQ.PERM[1:0] is W or RW: The write is ignored.
- When REQ.PERM[1:0] is SPEC: If the translation request is for a speculative read transaction, the master that issued the speculative read transaction must be notified that the read is unsuccessful.

——— **Note** ———

A faulting speculative read transaction is always terminated with an abort, regardless of the value of this field.

———————————

When the value of this field is Abort, the translation has failed and the transaction must terminate with an abort.

When the value of this field is StreamDisabled, the stream is disabled. The DTI master must abort all transactions for this stream.

When the value of this field is GlobalDisabled, the security world is disabled. The DTI master must abort all transactions for this security level.

The following table 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

FAULT_TYPE	SEC_SID	ATST	SID	SSV	SSID
StreamDisabled	Yes	Yes	Yes	No	No
GlobalDisabled	Yes	Yes	No	No	No

If this field is NonAbort or Abort then this translation only applies to this 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 non-zero, SID[CONT-1:0] in the translation request can be ignored when determining whether this translation matches future transactions.

When the value of FAULT_TYPE is NonAbort or Abort, this field is Reserved, SBZ.

DO_NOT_CACHE, bit [12]

This bit indicates to the DTI master when not to cache a translation.

0 Can be cached.

1 Must not be cached.

When the value of FAULT_TYPE is NonAbort or Abort:

- The value of this field must be 1.

When the value of FAULT_TYPE is StreamDisabled or GlobalDisabled:

- When the value of this bit is 0, the translation has not been invalidated before this message was sent.
- When the value of this bit is 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.

TRANSLATION_ID, bits [11:4]

This field gives the identification number for the translation.

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

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for slave-initiated messages, see *DTI-TBU protocol slave-initiated messages* on page 2-18.

0001 DTI_TBU_TRANS_FAULT.

3.2.4 Faulting expressions of the translation request message

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

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

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

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

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

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

3.2.5 Calculating transaction attributes

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

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

- Memory type.
- Shareability.
- Allocation hints.

Fields used to calculate the attributes

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

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

Note

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

The ATTR_OVR field is composed of subfields which are shown in the following table.

Table 3-3 ATTR_OVR subfields

Field bits	Field name
[3:0]	MemAttr
[4]	MTCFG
[6:5]	SHCFG
[15:7]	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, for example because it is a Device or Non-cacheable transaction, 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 3-49.
3. The allocation hints are modified based on the value of ALLOCCFG. For more information, see *The ALLOCCFG field* on page 3-50.
4. The shareability domain is modified based on the value of SHCFG. For more information see *The SHCFG field* on page 3-50
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 3-50.
6. A consistency check is applied to eliminate illegal attribute combinations. For more information, see *Consistency check on combination of translation attributes* on page 3-51.

The precise algorithm is as follows:

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

```
    MemoryAttributes attr_out;

    attr_out = DefaultMemAttrHints(attr_in);

    if ((resp.BYPASS == '1' || resp.STRW == EL1_S2)
            && (resp.ATTR_OVR.MTCFG == '1')) then
        attr_out = ModifyMemoryType(attr_out, resp.ATTR_OVR.MemAttr);
    else
        attr_out = attr_out;

    attr_out = ModifyAllocHints(attr_out, resp.ALLOCCFG);

    if resp.BYPASS == '1' || resp.STRW == EL1_S2 then
        attr_out = ModifyShareability(attr_out, resp.ATTR_OVR.SHCFG);

    attr_out = CombineAttributes(attr_out, resp);

    attr_out = ConsistencyCheck(attr_out);

    return attr_out;
```

The MemAttr and MTCFG fields

If the value of MTCFG is 1, then the MemAttr field provides the memory type override for incoming transactions. The following table shows the encoding of this field.

Table 3-4 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.	-	-
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;
```

If the value of MTCFG is 0, then the MemAttr field is Reserved, SBZ.

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

    if current_attr.inner.attrs IN {MemAttr_WT, MemAttr_WB} then
        current_attr.inner.Transient = T;
        current_attr.inner.ReadAllocate = RA;
        current_attr.inner.WriteAllocate = WA;

    if current_attr.outer.attrs IN {MemAttr_WT, MemAttr_WB} then
        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 *B.1.3 Memory attribute processing* on page Appx-B-118 for an example implementation.

Combining the translation response attributes

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

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

    MemoryAttributes attr_resp = DecodeAttr(resp.ATTR);

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

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

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

        // The allocation hints computed for the transaction so far are combined
        // with the allocation hints from the ATTR field.
```

```
        attr_txn = CombineAllocHints(attr_txn, attr_resp);

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

    return attr_txn;
```

When memory type, shareability and allocation hints are combined, the result is the strongest of each, as shown in the following table.

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

Figure 3-1 Combining the translation response attributes

See *B.1.3 Memory attribute processing* on page Appx-B-118 for the pseudocode implementation of this table.

Consistency check on combination of translation attributes

After the combination of the translation response attributes has been performed, the following additional conversions are performed to ensure that the attributes are consistent:

```
MemoryAttributes ConsistencyCheck(MemoryAttributes current_attr)

    if current_attr.type != MemType_Normal then
        current_attr.SH = OuterShareable;

    if (current_attr.type == MemType_Normal
            && current_attr.inner.attrs == MemAttr_NC) then
        current_attr.SH = OuterShareable;
        current_attr.inner.ReadAllocate = '0';
        current_attr.inner.WriteAllocate = '0';
        current_attr.inner.Transient = '0';

    if (current_attr.type == MemType_Normal
            && current_attr.outer.attrs == MemAttr_NC) then
        current_attr.SH = OuterShareable;
        current_attr.outer.ReadAllocate = '0';
        current_attr.outer.WriteAllocate = '0';
        current_attr.outer.Transient = '0';

    return current_attr;
```

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

3.2.6 Speculative transactions and translations

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

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

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

Note

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

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

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

Speculative write transactions are not supported.

Note

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

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

3.3 Invalidation and synchronization message group

This section describes the invalidation and synchronization message group. Invalidation operations are used by the DTI slave to indicate to the DTI master that certain information must no longer be cached.

For more information about the caching model used by the DTI-TBU Protocol, see *Chapter 4 DTI-TBU Caching Model* on page 4-69.

This section contains the following subsections:

3.3.1 DTI_TBU_INV_REQ

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

Description

An invalidation request.

Source

DTI slave.

Usage constraints

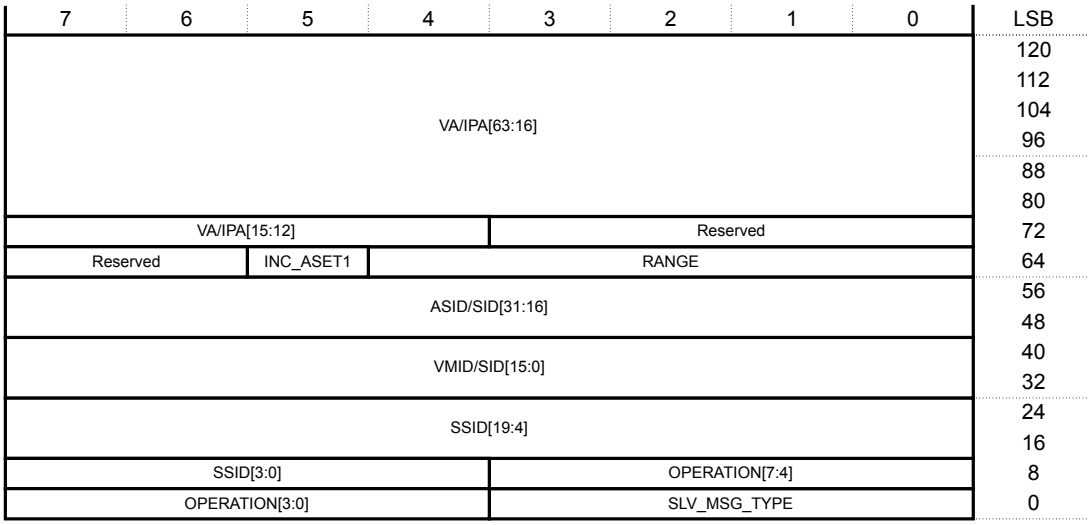
The DTI slave must have at least one invalidation token.

Flow control result

The DTI slave consumes an invalidation token.

Field descriptions

The DTI_TBU_INV_REQ bit assignments are:



VA/IPA, bits [127:76]

This field indicates the VA or IPA to be invalidated.

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

Bits [75:70]

Reserved, SBZ.

INC_ASET1, bit [69]

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

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

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

——— **Note** ———

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

———————————

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

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

- TLBI_S_EL1_ALL.
- TLBI_S_EL1_VAA.
- TLBI_NS_EL1_ALL.
- TLBI_NS_EL1_S1_VMID.
- TLBI_NS_EL1_S12_VMID.
- TLBI_NS_EL1_VAA.
- TLBI_NS_EL1_S2_IPA.
- TLBI_NS_EL2_ALL.
- TLBI_NS_EL2_VAA.
- TLBI_S_EL3_ALL.

RANGE, bits [68:64]

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

The range is calculated as two to the power of the value of this field.

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

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 *n* bits of the VMID field are ignored in both this message and the translations being considered for invalidation, where *n* is the value of this field.
- The value of this field must not be greater than four.

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

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

This field indicates the ASID value to invalidate.

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

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

This field indicates the VMID value to invalidate.

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

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

This field indicates the StreamID to invalidate.

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

SSID, bits [31:12]

This field indicates the SubstreamID to invalidate.

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

OPERATION, bits [11:4]

This field identifies the type of invalidation operation being performed.

When a DTI master receives a message with an unrecognized OPERATION field value, this specification recommends that the DTI master acknowledges the invalidation without performing any operation. For the encoding of this field and information on the effects of the invalidate operations, see *3.3.6 DTI-TBU invalidation operations* on page 3-62.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for slave-initiated messages, see *DTI-TBU protocol slave-initiated messages* on page 2-18.

0100 DTI_TBU_INV_REQ.

3.3.2 DTI_TBU_INV_ACK

The DTI_TBU_INV_ACK message is used to acknowledge an invalidation request.

Description

An invalidation acknowledgement.

Source

DTI master.

Usage constraints

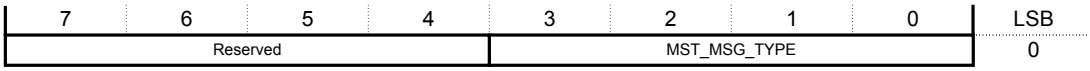
The DTI slave must have previously issued an invalidation request that has not yet been acknowledged.

Flow control result

The DTI master returns an invalidation token to the DTI slave.

Field descriptions

The DTI_TBU_INV_ACK bit assignments are:



Bits [7:4]

Reserved, SBZ.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for master-initiated messages, see *DTI-TBU protocol master-initiated messages* on page 2-18.

0100 DTI_TBU_INV_ACK.

3.3.3 DTI_TBU_SYNC_REQ

The DTI_TBU_SYNC_REQ message is used to request synchronization of the DTI master and DTI slave.

Description

A synchronization request.

Source

DTI slave.

Usage constraints

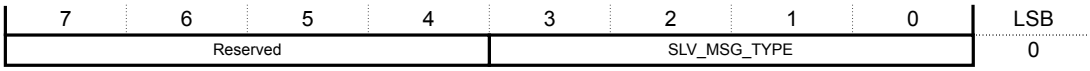
There must be no currently unacknowledged synchronization requests.

Flow control result

None.

Field descriptions

The DTI_TBU_SYNC_REQ bit assignments are:



Bits [7:4]

Reserved, SBZ.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for slave-initiated messages, see *DTI-TBU protocol slave-initiated messages* on page 2-18.

0101 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

DTI master.

Usage constraints

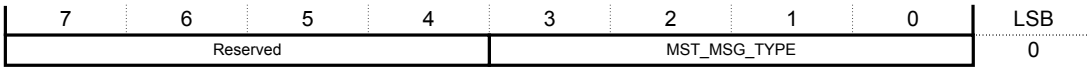
There must currently be an unacknowledged synchronization request.

Flow control result

None.

Field descriptions

The DTI_TBU_SYNC_ACK bit assignments are:



Bits [7:4]

Reserved, SBZ.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for master-initiated messages, see *DTI-TBU protocol master-initiated messages* on page 2-18.

0101 DTI_TBU_SYNC_ACK.

3.3.5 The DTI-TBU invalidation sequence

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

For all translations that are affected by the invalidation, the order in which they arrive at the DTI master determines how they are handled. The following figure shows the invalidation phases in which an affected DTI_TBU_TRANS_RESP can arrive.

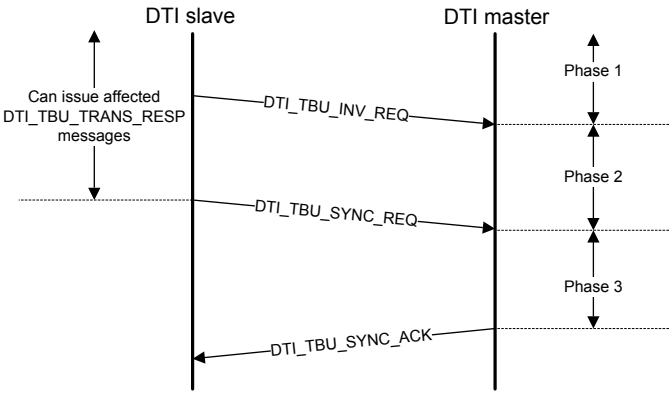


Figure 3-2 The phases of the invalidation sequence

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 DTI master must ensure translations within the scope of previous invalidations have been invalidated and transactions which use them have completed, before returning a DTI_TBU_SYNC_ACK message. The actions that must be taken depend upon in what phase of the invalidation sequence, the affected DTI_TBU_TRANS_RESP messages arrived. The following table describes the phases and required actions.

Table 3-5 Phases and actions of an invalidation sequence

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

Overlapping invalidations

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

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

Deadlock avoidance in the invalidation sequence

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

- A DTI master must not wait for an outstanding translation to complete before returning a DTI_TBU_SYNC_ACK message. Any outstanding translations must be discarded on receipt of a DTI_TBU_SYNC_REQ. The example following this list gives a case in which 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 DTI master 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 DTI master 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

The following table specifies the OPERATION field encodings and 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 3-6 List of invalidation operations

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

Table 3-6 List of invalidation operations (continued)

Field encoding	Invalidation operation	StreamWorld affected	SEC_SID affected	Valid fields
0x30	CFGI_NS_SID	-	Non-secure.	SID, RANGE
0x38	CFGI_NS_SID_SSID	-	Non-secure.	SID, SSID
0x06	INV_ALL	All.	All.	-

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

- Invalidate operations including a VA and ASID invalidate the translation regardless of the ASID being invalidated.
- Invalidate operations including an ASID but no VA do not invalidate the translation.

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

- CFGI_NS_ALL.
- CFGI_S_ALL.
- INV_ALL.

Note

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

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

Configuration invalidate operations

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

The following table shows the SMMUv3 commands that map that to DTI configuration invalidate operations.

Table 3-7 Mappings of SMMUv3 commands onto DTI invalidate operations

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

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

3.4 Register access message group

This section describes the register access message group.

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

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

This section contains the following subsections:

3.4.1 DTI_TBU_REG_WRITE

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

Description

A register write request.

Source

DTI slave.

Usage constraints

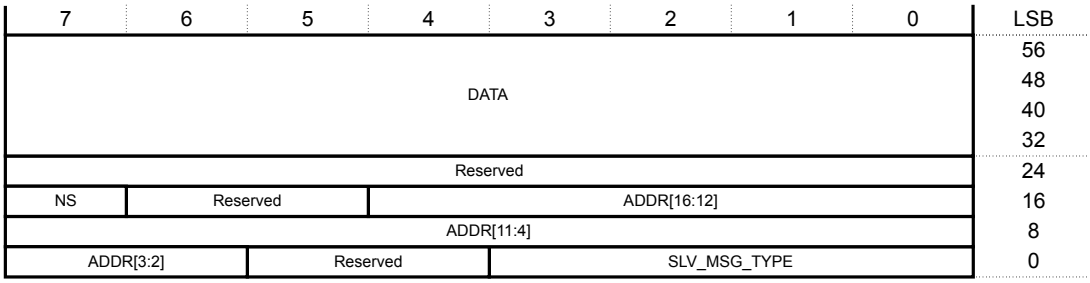
The DTI slave must have no outstanding register reads or writes.

Flow control result

None.

Field descriptions

The DTI_TBU_REG_WRITE bit assignments are:



DATA, bits [63:32]

This field holds the data to be written.

Bits [31:24]

Reserved, SBZ.

NS, bit [23]

This bit indicates the Security level of the register access.

0 Secure.

1 Non-secure.

Bits [22:21]

Reserved, SBZ.

ADDR, bits [20:6]

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

Bits [5:4]

Reserved, SBZ.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for slave-initiated messages, see *DTI-TBU protocol slave-initiated messages* on page 2-18.

0110 DTI_TBU_REG_WRITE.

3.4.2 DTI_TBU_REG_WACK

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

Description

A register write acknowledgement.

Source

DTI master.

Usage constraints

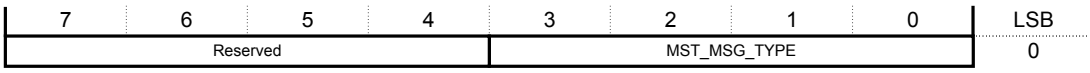
The DTI slave must have previously issued a register write request that has not yet been acknowledged.

Flow control result

None.

Field descriptions

The DTI_TBU_REG_WACK bit assignments are:



Bits [7:4]

Reserved, SBZ.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for master-initiated messages, see *DTI-TBU protocol master-initiated messages* on page 2-18.

0110 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

DTI slave.

Usage constraints

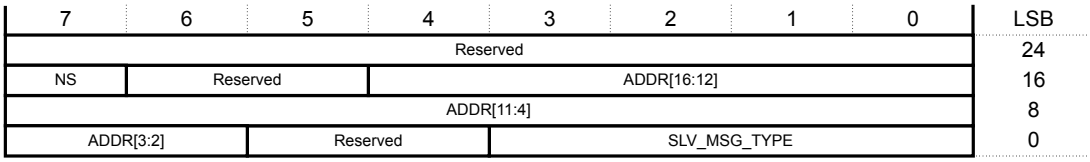
The DTI slave must have no outstanding reads or writes.

Flow control result

None.

Field descriptions

The DTI_TBU_REG_READ bit assignments are:



Bits [31:24]

Reserved, SBZ.

NS, bit [23]

This bit indicates the Security level of the register access.

0 Secure.

1 Non-secure.

Bits [22:21]

Reserved, SBZ.

ADDR, bits [20:6]

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

Bits [5:4]

Reserved, SBZ.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for slave-initiated messages, see *DTI-TBU protocol slave-initiated messages* on page 2-18.

0111 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

DTI master.

Usage constraints

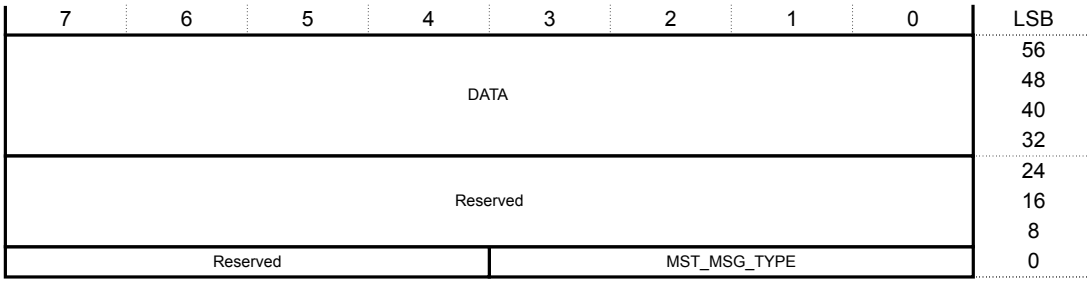
The DTI slave must have previously issued a register read request that has not yet received a response.

Flow control result

None.

Field descriptions

The DTI_TBU_REG_RACK bit assignments are:



DATA, bits [63:32]

This field holds the read data.

Bits [31:4]

Reserved, SBZ.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for master-initiated messages, see *DTI-TBU protocol master-initiated messages* on page 2-18.

0111 DTI_TBU_REG_RDATA.

3.4.5 Deadlock avoidance in register accesses

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

Chapter 4
DTI-TBU Caching Model

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

It contains the following sections:

4.1 Caching model

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

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

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

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

Each cache contains fields for the following:

Tag
This is compared against future transactions or invalidations.

Scope
This controls how much of the tag must match.

Data
This is used to translate a transaction.

4.2 Lookup process

A lookup into the caches progresses as shown in the following diagram.

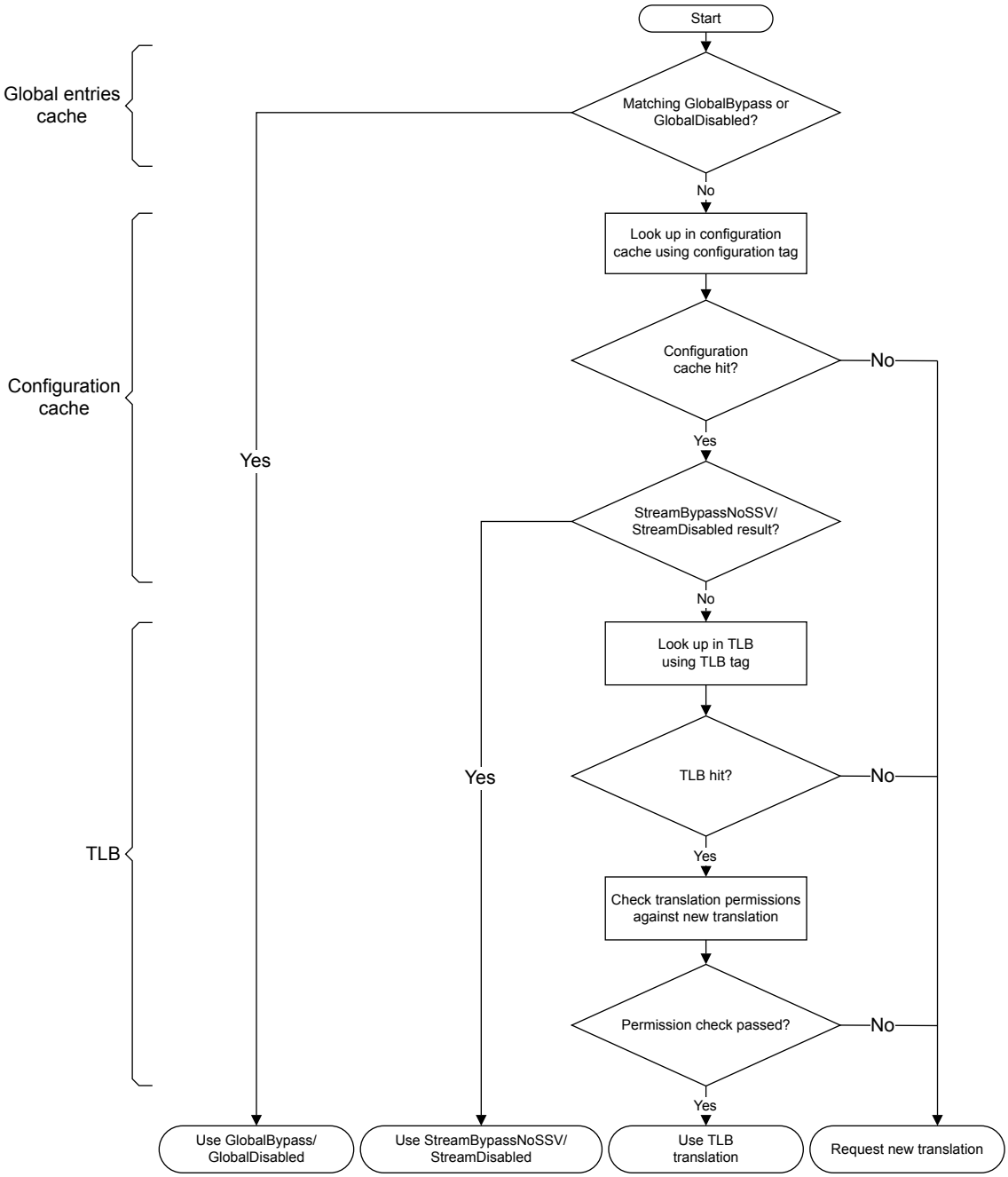


Figure 4-1 Lookup process

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

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

4.3 Global entry cache

The global entry cache can contain up to three entries:

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

The message fields that comprise the entry tag field combine to index these three entry types.

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

Tag fields

- DTI_TBU_TRANS_REQ.SEC_SID
- DTI_TBU_TRANS_REQ.ATS

Scope fields

- DTI_TBU_TRANS_RESP.TRANS_RNG

Data fields

- DTI_TBU_TRANS_RESP.NSOVR
- DTI_TBU_TRANS_RESP.ALLOCCFG
- DTI_TBU_TRANS_RESP.NS
- DTI_TBU_TRANS_RESP.PRIVCFG
- DTI_TBU_TRANS_RESP.INSTCFG
- DTI_TBU_TRANS_RESP.ATTR_OVR
- DTI_TBU_TRANS_RESP.CTXTATTR

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

Tag fields

- DTI_TBU_TRANS_REQ.SEC_SID
- DTI_TBU_TRANS_REQ.ATS

Scope fields

None.

Data fields

None.

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

4.4 Configuration cache

The configuration cache performs the following functions:

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

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

Tag fields

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

Scope fields

- DTI_TBU_TRANS_RESP.CONT
- DTI_TBU_TRANS_RESP.ALLOW_NSX

Data fields

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

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

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

Tag fields

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

Scope fields

- DTI_TBU_TRANS_FAULT.CONT

Data fields

None.

4.5 TLB

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

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

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

Tag fields

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

Scope fields

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

Data fields

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

Chapter 5
DTI-ATS Messages

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

It contains the following sections:

5.1 Connection and disconnection message group

This section describes the ATS connection and disconnection message group.

This section contains the following subsections:

5.1.1 DTI_ATS_CONDIS_REQ

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

Description

Connection state change request.

Source

DTI master.

Usage constraints

The DTI-ATS master 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 DTI-ATS master can only send a connect request when:

- The channel is in the DISCONNECTED state.

Flow control result

None.

Field descriptions

The DTI_ATS_CONDIS_REQ bit assignments are:

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

Bits [31:20]

Reserved, SBZ.

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, bits [19:12]

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

When 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 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.

VERSION, bits [11:8]

This field indicates the requested protocol version.

0000 DTI-ATSv1.

All other encodings are reserved.

A DTI-ATS master can request any protocol version it supports. Only DTI-ATSv1 is currently defined, however a DTI-ATS slave must accept requests for later protocol versions. 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 DTI master.

1 DTI-ATS.

——— **Note** ———

This bit must be 1.

———————————

STATE, bit [4]

This bit indicates the new channel state requested.

0 Disconnect request.

1 Connect request.

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

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

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for master-initiated messages, see *DTI-ATS protocol master-initiated messages* on page 2-18.

0000 DTI_ATS_CONDIS_REQ.

5.1.2 DTI_ATS_CONDIS_ACK

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

Description

A connection state change acknowledgement.

Source

DTI slave.

Usage constraints

The DTI master must have previously issued an unacknowledged DTI_ATS_CONDIS_REQ message.

Flow control result

None.

Field descriptions

The DTI_ATS_CONDIS_ACK bit assignments are:

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

Bits [31:25]

Reserved, SBZ.

OAS, bits [24:21]

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

0000 32 bits (4GB).

0001 36 bits (64GB).

0010 40 bits (1TB).

0011 42 bits (4TB).

0100 44 bits (16TB).

0101 48 bits (256TB).

0110 52 bits (4PB).

All other values are Reserved.

SUP_PRI, Bit [20]

This bit 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.

TOK_TRANS_GNT, bits [19:12]

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

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

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

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

VERSION, bits [11:8]

This bit indicates the protocol version that the DTI slave has granted.

0000 DTI-ATSv1.

All other encodings are reserved.

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

Bits [7:5]

Reserved, SBZ.

STATE, bit [4]

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

0 DISCONNECTED.

1 CONNECTED.

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

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

SLV_MSG_TYPE, bits [3:0]

Identifies the message type. The value of this field is taken from the list of encodings for slave-initiated messages, see *DTI-ATS protocol slave-initiated message* on page 2-19.

0000 DTI_ATS_CONDIS_ACK.

5.2 Translation request message group

This section describes the ATS translation request message group.

This section contains the following subsections:

5.2.1 DTI_ATS_TRANS_REQ

The DTI_ATS_TRANS_REQ message is used to initiate a translation request.

Description

A translation request.

Source

DTI master.

Usage constraints

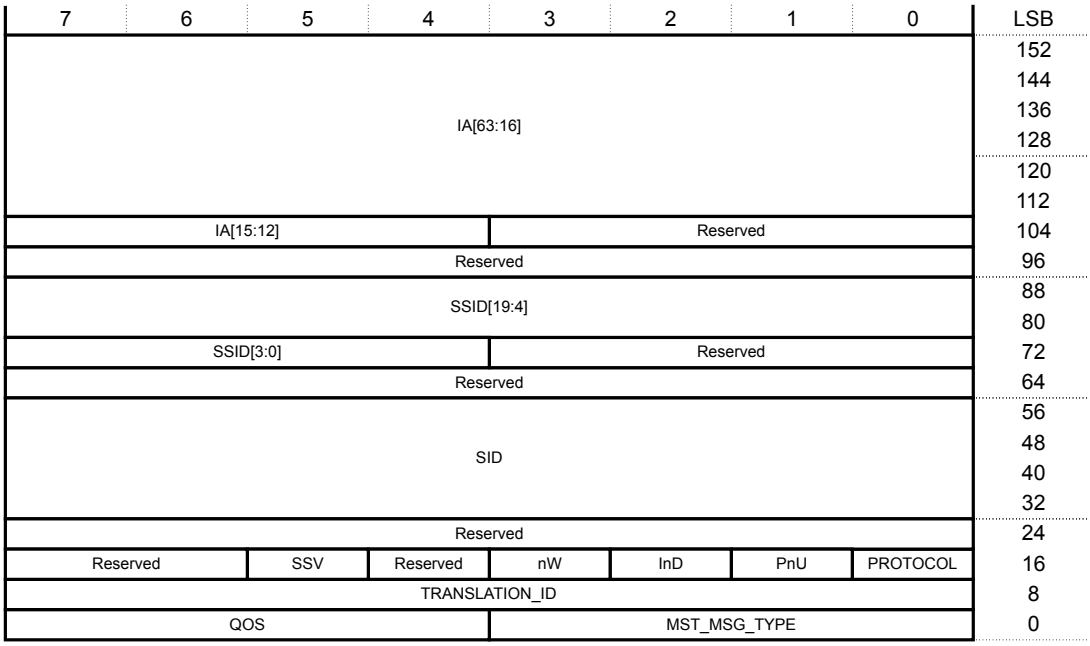
The DTI master must have at least one translation token.

Flow control result

The DTI master sends a translation token to the DTI slave.

Field descriptions

The DTI_ATS_TRANS_REQ bit assignments are:



IA, bits [159:108]

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

Bits [107:96]

Reserved, SBZ.

SSID, bits [95:76]

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

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

Bits [75:64]

Reserved, SBZ.

SID, bits [63:32]

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

Bits [31:22]

Reserved, SBZ.

SSV, bit [21]

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

0 SSID not valid.

1 SSID valid.

Bit [20]

Reserved, SBZ.

nW, bit [19]

This bit indicates whether write access is requested.

0 Read and write access.

1 Read-only access.

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

InD, bit [18]

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

0 The translation will only be used for data accesses.

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

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

PnU, bit [17]

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

0 Unprivileged.

1 Privileged.

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

PROTOCOL, bit [16]

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

1 DTI-ATS.

This bit must be 1.

TRANSLATION_ID, bits [15:8]

This field gives the identification number for the translation.

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

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

QOS, bits [7:4]

This field indicates the Quality of Service priority level.

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

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for master-initiated messages, see *DTI-ATS protocol master-initiated messages* on page 2-18.

0010 DTI_ATS_TRANS_REQ.

5.2.2 DTI_ATS_TRANS_RESP

The DTI_ATS_TRANS_RESP message is used respond to a translation request.

Description

A DTI translation response.

Source

DTI slave.

Usage constraints

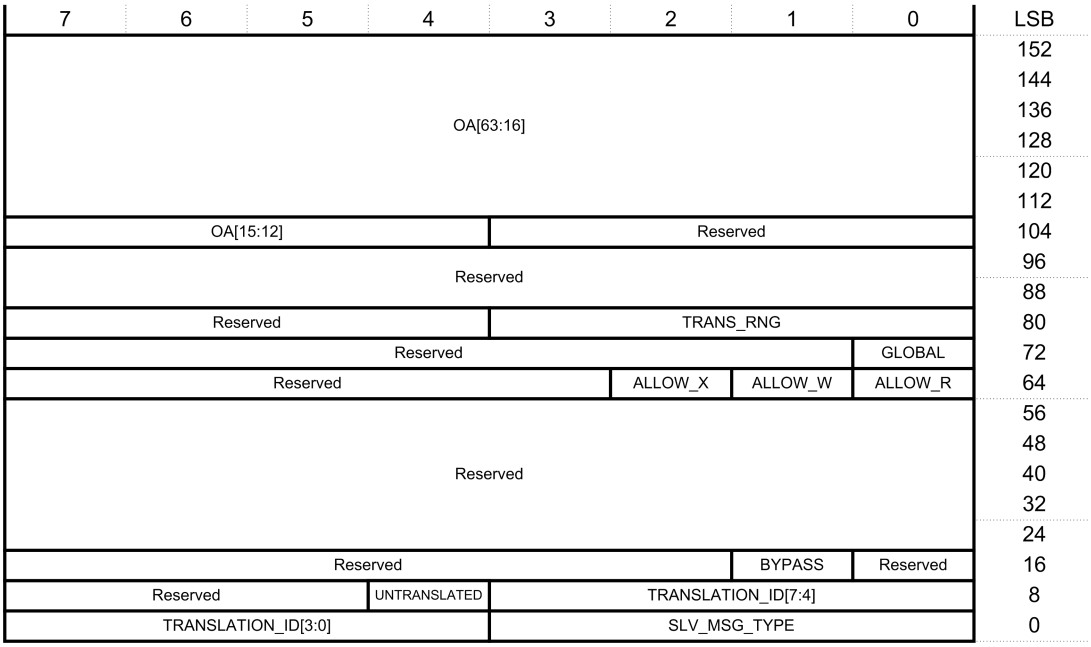
The DTI master must have previously issued a translation request that has not yet generated either a response or a fault message.

Flow control result

The DTI slave returns a translation token to the DTI master.

Field descriptions

The DTI_ATS_TRANS_RESP bit assignments are:



OA, bits [159:108]

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

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.

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

When BYPASS is 1, this field must be 0.

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

Bits [107:84]

Reserved, SBZ.

TRANS_RNG, bits[83:80]

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

When BYPASS=0

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

0000	4KB.
0001	16KB.
0010	64KB.
0011	2MB.
0100	32MB.
0101	512MB.
0110	1GB.
0111	16GB.
1000	4TB.
1001	128TB.

All other values are reserved.

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

This field must not indicate a size of 4TB or 128TB unless the OAS field of the DTI_ATS_CONDIS_ACK message received during the connection sequence indicates a size of 52 bits.

When BYPASS=1

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

0000	32 bits (4GB).
0001	36 bits (64GB).
0010	40 bits (1TB).
0011	42 bits (4TB).
0100	44 bits (16TB).
0101	48 bits (256TB).
0110	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.

Bits [79:73]

Reserved, SBZ.

GLOBAL, bit[72]

This bit indicates whether this translation applies to all SubstreamIDs.

0	Non-global.
1	Global.

When the value of the SSV bit in the requesting DTI_ATS_TRANS_REQ message is 0, this bit is reserved, SBZ.

When the value of the SSV bit in the requesting DTI_ATS_TRANS_REQ message is 1, and the value of BYPASS is 1, this bit must be 1.

Bits [71:67]

Reserved, SBZ.

ALLOW_X, bit [66]

This bit indicates permissions for instruction reads.

0 Not permitted.

1 Permitted.

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

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

ALLOW_W, bit [65]

This bit indicates permissions for data write accesses.

0 Not permitted.

1 Permitted.

ALLOW_R, bit [64]

This bit indicates permissions for data read accesses.

0 Not permitted.

1 Permitted.

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

Bits [63:18]

Reserved, SBZ.

BYPASS, bit[17]

This field indicates that translation for this StreamID is bypassed.

0 Normal translation.

1 Translation bypassed.

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

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

Bits [16:13]

Reserved, SBZ.

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 DTI slave 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 0, The PCIe Root Complex must access the page using untranslated transactions.

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

TRANSLATION_ID, bits [11:4]

This field gives the identification number for the translation.

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

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for slave-initiated messages, see *DTI-ATS protocol slave-initiated message* on page 2-19.

0010 DTI_ATS_TRANS_RESP.

5.2.3 DTI_ATS_TRANS_FAULT

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

Description

A translation fault response.

Source

DTI slave.

Usage constraints

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

Flow control result

The DTI slave returns a translation token to the DTI master.

Field descriptions

The DTI_ATS_TRANS_FAULT bit assignments are:

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

Bits [31:19]

Reserved, SBZ.

FAULT_TYPE, bits [18:17]

This bit is used to tell the DTI master how to handle the fault.

00 InvalidTranslation.

01 CompleterAbort.

10 UnsupportedRequest.

11 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 DTI master 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 DTI master 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 DTI master returns a Translation Completion message with a status value as Unsupported Request (UR).

Bits [16:12]

Reserved, SBZ.

TRANSLATION_ID, bits [11:4]

This field gives the identification number for the translation.

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

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for slave-initiated messages, see *DTI-ATS protocol slave-initiated message* on page 2-19.

0001 DTI_ATS_TRANS_FAULT.

5.2.4 The ATS translation sequence

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

The following diagram shows the steps required in a full ATS translation process that is supported by DTI.

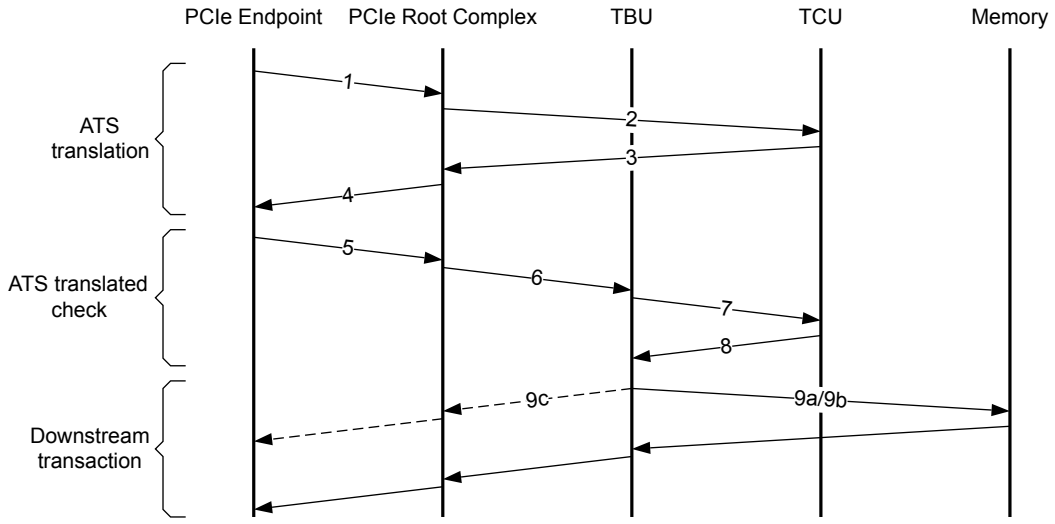


Figure 5-1 The steps of a complete ATS translation sequence in DTI

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

The SMMU can be configured to:

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

Requests for multiple translations

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

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

5.3 Invalidation and synchronization message group

This section describes the ATS invalidation and synchronization message group.

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

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

This section contains the following subsections:

5.3.1 DTI_ATS_INV_REQ

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

Description

An invalidation request.

Source

DTI slave.

Usage constraints

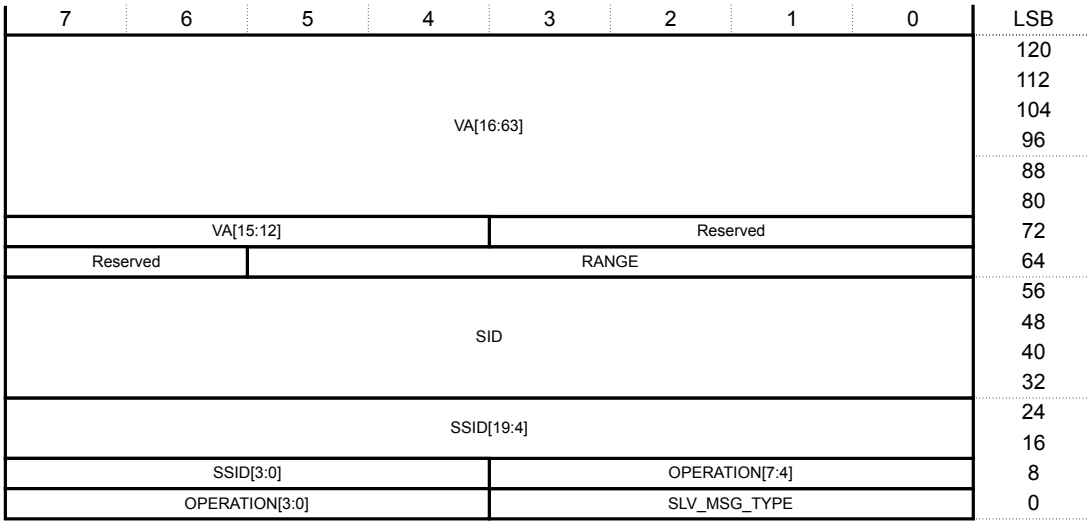
The DTI slave must have at least one invalidation token.

Flow control result

The DTI slave consumes an invalidation token.

Field descriptions

The DTI_ATS_INV_REQ bit assignments are:



VA, bits [127:76]

The Virtual Address or Intermediate Physical Address to be invalidated.

Bits [75:70]

Reserved, SBZ.

RANGE, bits [69:64]

This field identifies a range of Virtual Addresses for invalidation.

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

SID, bits [63:32]

This field indicates the StreamID to be invalidated.

The receiving master 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 master, the master must acknowledge this message without performing an operation.

SSID, bits [31:12]

This field indicates the SubstreamID to be invalidated.

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

OPERATION, bits [11:4]

This field identifies the type of invalidation operation being performed.

When a DTI master receives a message with an unrecognized OPERATION field value, this specification recommends that the DTI master 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 *5.3.6 DTI-ATS invalidation operations* on page 5-101.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for slave-initiated messages, see *DTI-ATS protocol slave-initiated message* on page 2-19.

1100 DTI_ATS_INV_REQ.

5.3.2 DTI_ATS_INV_ACK

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

Description

A cache data invalidate acknowledgement.

Source

DTI master.

Usage constraints

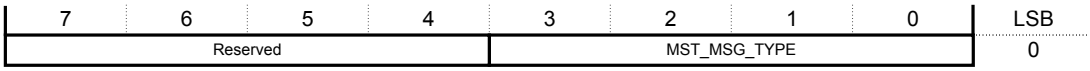
The DTI slave must have previously issued an invalidation request that has not yet been acknowledged.

Flow control result

The DTI master returns an invalidation token to the DTI slave.

Field descriptions

The DTI_TBU_INV_ACK bit assignments are:



Bits [7:4]

Reserved, SBZ.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for master-initiated messages, see *DTI-ATS protocol master-initiated messages* on page 2-18.

1100 DTI_ATS_INV_ACK.

5.3.3 DTI_ATS_SYNC_REQ

The DTI_ATS_SYNC_REQ message is used to request synchronization between the DTI master and DTI slave.

Description

A synchronization request.

Source

DTI slave.

Usage constraints

There must be no currently unacknowledged synchronization requests.

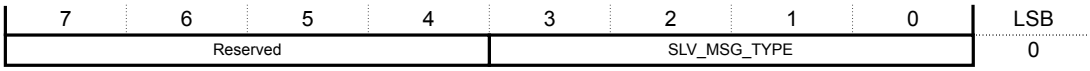
There must be no currently unacknowledged invalidation requests.

Flow control result

None.

Field descriptions

The DTI_ATS_SYNC_REQ bit assignments are:



Bits [7:4]

Reserved, SBZ.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for slave-initiated messages, see *DTI-ATS protocol slave-initiated message* on page 2-19.

1101 DTI_ATS_SYNC_REQ.

5.3.4 DTI_ATS_SYNC_ACK

The DTI_ATS_SYNC_ACK message is used to acknowledge a synchronization request.

Description

A synchronization acknowledge.

Source

DTI master.

Usage constraints

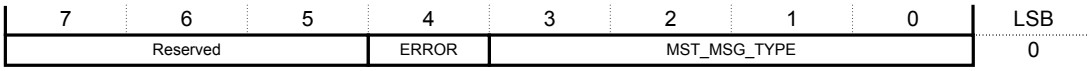
There must currently be an outstanding synchronization request.

Flow control result

None.

Field descriptions

The DTI_ATS_SYNC_ACK bit assignments are:



Bits [7:5]

Reserved, SBZ.

ERROR, bit [4]

This bit indicates that a PCIe error has occurred.

0 Success.

1 Error.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for master-initiated messages, see *DTI-ATS protocol master-initiated messages* on page 2-18.

1101 DTI_ATS_SYNC_ACK.

5.3.5 The DTI-ATS invalidation sequence

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

SMMUv3 requires that a DTI slave that intends to invalidate entries in an ATC must first invalidate the equivalent TBU entries. This results in an invalidation sequence as shown in the following diagram.

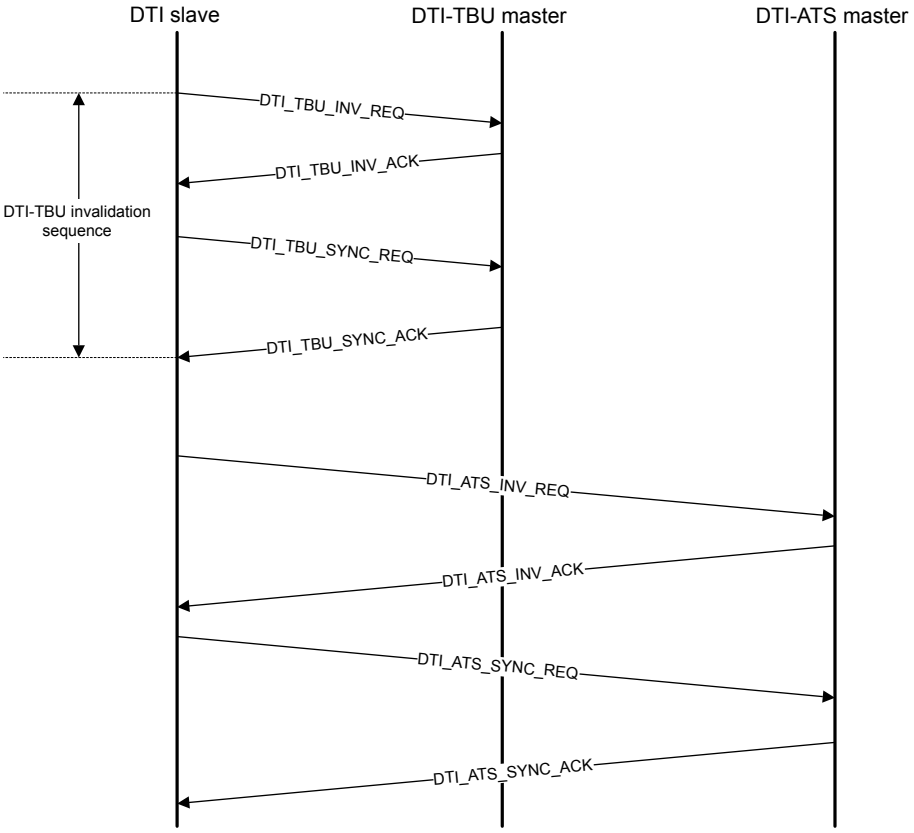


Figure 5-2 DTI-ATS invalidation sequence

The invalidation sequence that is shown in the diagram has the following steps:

1. Issue a TLB invalidate operation to the TBU and wait for it to complete.
2. Issue an invalidation synchronization operation to the TBU and wait for it to complete.
3. Issue an ATS invalidation operation to the PCIe Root Complex and wait for it to complete.
4. Issue an invalidation synchronization to the PCIe Root Complex and wait for it to complete.

The return of a DTI_ATS_SYNC_ACK message indicates that:

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

Note

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

Handling outstanding invalidations

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

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

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

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

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

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

Ensuring downstream transaction completion

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

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

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

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

5.3.6 DTI-ATS invalidation operations

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

Types of invalidation operation

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

Table 5-1 List of invalidation operations

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

Mapping DTI-ATS to SMMUv3 invalidate operations

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

Table 5-2 Mapping DTI-ATS operation to SMMUv3 command

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

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

5.4 Page request message group

This section describes the ATS page request message group.

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

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

This section contains the following subsections:

5.4.1 DTI_ATS_PAGE_REQ

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

Description

A speculative page request.

Source

DTI master.

Usage constraints

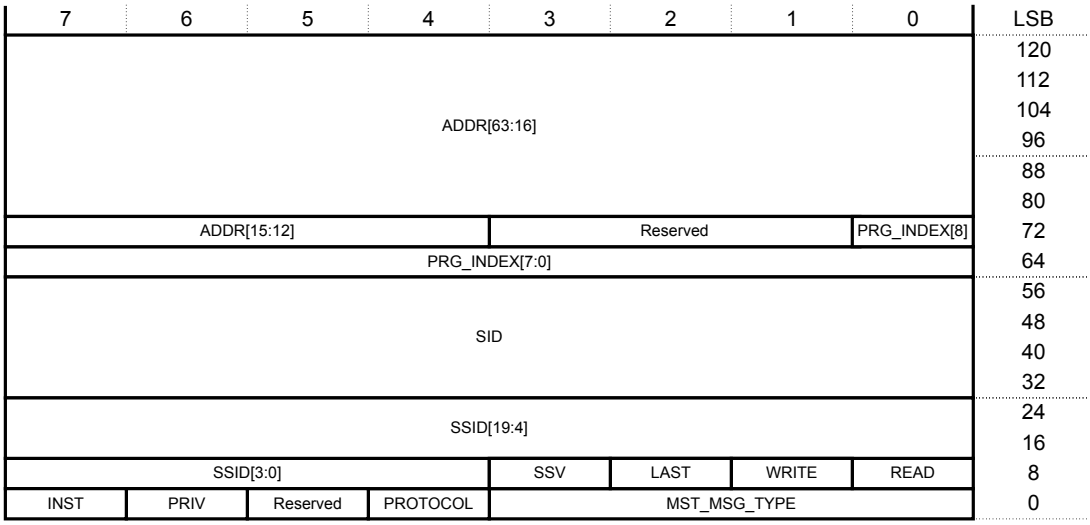
There must be no current outstanding unacknowledged DTI_ATS_PAGE_REQ message.

Flow control result

None.

Field descriptions

The DTI_ATS_PAGE_REQ bit assignments are:



ADDR, bits [127:76]

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

Bits [75:73]

Reserved, SBZ.

PRG_INDEX, bits [72:64]

This field identifies the Page Request group index.

SID, bits [63:32]

This field indicates the StreamID used for this transaction.

SSID, bits [31:12]

This field holds the SubstreamID used for this transaction.

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

SSV, bits [11]

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

0 SSID not valid.

1 SSID valid.

LAST, bit [10]

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

Note

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

WRITE, bit [9]

This bit indicates whether write access is requested.

0 Write access is not requested.

1 Write access is requested.

A page request does not set the Dirty flag.

READ, bit [8]

This bit indicates whether read access is requested.

0 Read access is not requested.

1 Read access is requested.

INST, bit [7]

This bit indicates whether execute access is requested.

0 Execute access is not requested.

1 Execute access is requested.

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

PRIV, bit [6]

This bit indicates whether privileged access is requested.

0 Unprivileged.

1 Privileged.

Bit [5]

Reserved, SBZ.

PROTOCOL, bit [4]

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

1 DTI-ATS.

This bit must be 1.

MST_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for master-initiated messages, see *DTI-ATS protocol master-initiated messages* on page 2-18.

1000 DTI_ATS_PAGE_REQ.

5.4.2 DTI_ATS_PAGE_ACK

The DTI_ATS_PAGE_ACK message is used to acknowledge a page request.

Description

A page request acknowledgement.

Source

DTI slave.

Usage constraints

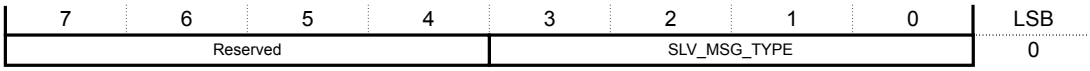
The DTI master must have previously issued a DTI_ATS_PAGE_REQ message that has not yet been acknowledged.

Flow control result

None.

Field descriptions

The DTI_ATS_PAGE_ACK bit assignments are:



Bits [7:4]

Reserved, SBZ.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for slave-initiated messages, see *DTI-ATS protocol slave-initiated message* on page 2-19.

1000 DTI_ATS_PAGE_ACK.

5.4.3 DTI_ATS_PAGE_RESP

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

Description

An ATS page response.

Source

DTI slave.

Usage constraints

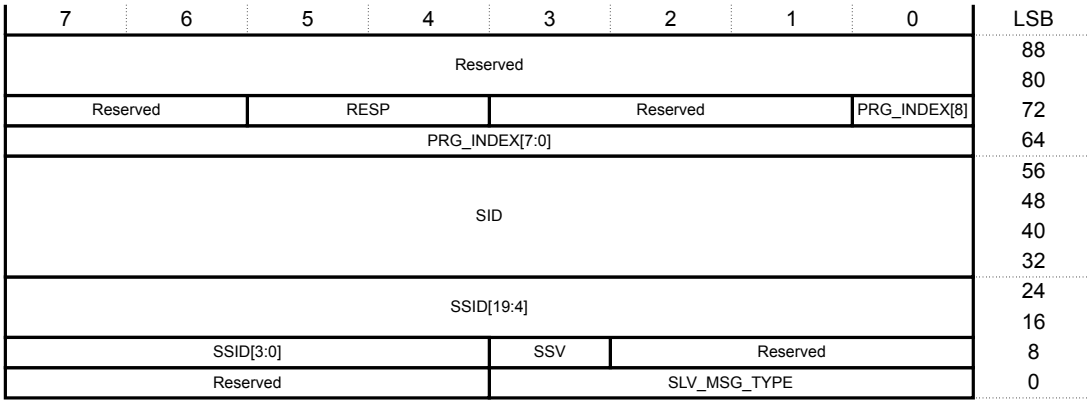
None.

Flow control result

None.

Field descriptions

The DTI_ATS_PAGE_RESP bit assignments are:



Bits [95:78]

Reserved, SBZ.

RESP, bits [77:76]

This field indicates the response code to the page request.

00 ResponseFailure.

01 InvalidRequest.

10 Success.

11 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 master 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 master, the master must ignore this message.

SSID, bits [31:12]

This field holds the SubstreamID used for this page request.

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

SSV, bits [11]

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

0 SSID not valid.

1 SSID valid.

Bits [10:4]

Reserved, SBZ.

SLV_MSG_TYPE, bits [3:0]

This field identifies the message type. The value of this field is taken from the list of encodings for slave-initiated messages, see *DTI-ATS protocol slave-initiated message* on page 2-19.

1001 DTI_ATS_PAGE_RESP.

5.4.4 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 master does not match an unanswered DTI_ATS_PAGE_REQ, for which 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 masters. As such, a DTI_ATS_PAGE_RESP message might be received with a StreamID that is not used by the master and which does not match any of the StreamIDs from its unanswered DTI_ATS_PAGE_REQ messages.

——— **Note** ———

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

———————————

Chapter 6
Transport Layer

This chapter describes the transport layer of the DTI protocol.

It contains the following sections:

6.1 Introduction

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

The transport layer is responsible for:

- Indicating the source or destination DTI Master.
- Managing the link-level flow control.

The transport layer is not permitted to:

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

6.2 AXI4-Stream transport protocol

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

This section contains the following subsections:

6.2.1 AXI4-Stream signals

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

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

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

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

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

Signal	Usage	Notes
TDEST	Destination node ID or not implemented.	The meaning of this signal depends on the direction of the interface: • For a master to slave interface, this signal is not implemented. There is only one slave in the network. • For a slave to master interface, this signal indicates which master the message is for.
TUSER	Not implemented.	The DTI protocol does not require this signal.

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

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

Direction	Suffix
Master to slave.	*_DTI_DN
Slave to master.	*_DTI_UP

For example, the **TDATA** signal passing from a DTI master to a DTI slave would be **TDATA_DTI_DN**.

Components can add a further suffix of *_S or *_M to distinguish between master and slave interfaces. An interconnect might use this suffix to provide a *_S suffix to signals on its slave interface and *_M to signals on its master interface.

6.2.2 Interleaving

Message of the DTI protocol must not be interleaved. Even in the cases where **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 which a new message is permitted.

6.2.3 Usage of the TID and TDEST signals

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

- A component implementing a DTI master interface does not implement the following:
  — The **TID** signal on the AXI4-Stream master port.
  — The **TDEST** signal on the AXI4-Stream slave port.
- An interconnect that connects multiple DTI masters to a single DTI slave adds additional bits, as required, to the **TID** signal on its DTI master interface. The interconnect accepts messages from the DTI slave and redirects them to the appropriate DTI master by IMPLEMENTATION DEFINED mapping of the **TID** signal.

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

Appendix A
Revisions

This appendix describes the changes between released issues of this book.

It contains the following section:

- *A.1 Revisions* on page Appx-A-114.

A.1 Revisions

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

Table A-1 Issue 0000-00

Change	Location
First release	-

Table A-2 Differences between issue 0000-00 and issue 0000-01

Change	Location
Replaced erroneous reference to the PROTOCOL field with a reference to the VERSION field.	*3.1.2 DTI_TBU_CONDIS_ACK* on page 3-28 *5.1.2 DTI_ATS_CONDIS_ACK* on page 5-80
Fixed erroneous introductory text in the Field Descriptions for the DTI_ATS_CONDIS_ACK message.	*5.1.2 DTI_ATS_CONDIS_ACK* on page 5-80

Table A-3 Differences between issue 0000-01 and issue 0000-02

Change	Location
Corrections and clarifications.	-

Table A-4 Differences between issue 0000-02 and issue 0000-03

Change	Location
Replaced SPECULATIVE and RnW fields with PERM[1:0].	*3.2.1 DTI_TBU_TRANS_REQ* on page 3-31
Added Pseudocode in various places.	-

Appendix B
Pseudocode

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

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

It contains the following sections:

B.1 Memory attributes

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

This section contains the following subsections:

B.1.1 Memory attribute types

These types are used in the rest of the specification to document 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
)
```

B.1.2 Memory attribute decoding

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

```
// 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()
// ===============
// 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()
// =======================
// 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()
// ============
// 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' mematttrs.type = MemoryType_GRE;
        mematttrs.inner = MemAttrHints UNKNOWN;
        mematttrs.outer = MemAttrHints UNKNOWN;
        mematttrs.SH = OuterShareable;

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

    return mematttrs;
```

B.1.3 Memory attribute processing

This section details the procedures for combining memory type information.

```
// 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()
// ===================
// 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()
// =====================
// 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()
// ===================
// 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()
// ====================
// 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;
```

B.2 Message field requirements

This section provides an example implementation of self-consistency checks for DTI-TBU messages.

If a message is omitted below, it has no well-formedness checks.

```
// CondisAckCheck()
// ================
// Assertions for well-formedness of connect/disconnect acknowledge messages.

CondisAckCheck(DTI_TBU_CONDIS_ACK ack, DTI_TBU_CONDIS_REQ req)

    case ack.ACK_STATE of
        when ConnectAck

            // Assert protocol version is not greater than master version field
            assert UInt(ack.VERSION) <= UInt(req.VERSION);

        when DisconnectAck
            // nothing to check

    return;

// TransReqCheck()
// ===============
// Assertions for well-formedness of translation requests.

TransReqCheck(DTI_TBU_TRANS_REQ req)

    if req.SEC_SID=='0' then
        assert (req.NS=='1');

    if req.ATST=='1' then
        assert (req.SSV=='0');
        assert (req.SEC_SID=='0');

    if req.PERM IN {W, RW, SPEC} then
        assert (req.InD=='0');

    if req.PERM == SPEC then
        assert (req.PnU=='0');

    if req.SSV == '0' then
        assert Master_SBZ(req.SSID);

    return;

// TransRespCheck()
// ================
// Assertions to check well-formedness of a translation response, including
// restrictions based on the originating translation request that.

TransRespCheck(DTI_TBU_TRANS_RESP resp, DTI_TBU_TRANS_REQ req)

    if resp.BYPASS == '1' then
        assert (Slave_SBZ(resp.SH));
        assert (Slave_SBZ(resp.ATTR));

    if (resp.BYPASS == '1'
            || req.SEC_SID == '1'
            || !(resp.STRW IN {EL1,EL1_S2})) then
        // Stage 2 not in use
        assert (resp.S2HWATTR == '0000');

    if (resp.BYPASS == '1' || resp.STRW == EL1_S2) then
        // Stage 1 not in use
        assert (resp.S1HWATTR == '0000');

    if resp.BYPASS == '0' then
        // INVAL_RNG must be smaller than OAS
        assert (INVAL_RNG_MSB(resp.INVAL_RNG)
            <= OAS_MSB(DTIProtocolState.CurrentOAS));
        // Additional restriction: 42-bit (4TB) range is only permitted if OAS
        // is 52-bit.
        if OAS_MSB(DTIProtocolState.CurrentOAS) != 52 then
            assert (INVAL_RNG_MSB(resp.INVAL_RNG) < 42);
    else
        assert (Slave_SBZ(resp.INVAL_RNG));

    if resp.BYPASS == '0' then
```

```
        // TRANS_RNG must fit within OAS
        assert (TRANS_RNG_MSB(resp.TRANS_RNG)
            <= OAS_MSB(DTIProtocolState.CurrentOAS));
        // Additional restriction: 42-bit (4TB) and 47-bit (128TB) ranges are
        // only permitted if OAS is 52-bit.
        if TRANS_RNG_MSB(resp.TRANS_RNG) >= 42 then
            assert (OAS_MSB(DTIProtocolState.CurrentOAS) == 52);
    else
        assert (BYPASS_TRANS_RNG_MSB(resp.TRANS_RNG)
            == OAS_MSB(DTIProtocolState.CurrentOAS));

    if resp.BYPASS == '1' then
        assert (Slave_SBZ(resp.GLOBAL));
    elsif (resp.STRW == EL3) then
        assert (resp.GLOBAL == '1');

    if resp.BYPASS == '1' then
        assert (Slave_SBZ(resp.TBI));

    if req.ATST == '1' then
        assert (resp.NS == '1');

    if req.SEC_SID == '0' then
        assert (resp.NS == '1');

    if (resp.BYPASS == '1' && resp.NSOVR == '0') then
        assert (Slave_SBO(resp.NS));

    if (resp.BYPASS == '1' && req.SEC_SID == '0') then
        assert (Slave_SBZ(resp.ALLOW_NSX));

    if (resp.BYPASS == '1') then
        assert (Slave_SBZ(resp.ALLOW_PW));
        assert (Slave_SBZ(resp.ALLOW_PR));
        assert (Slave_SBZ(resp.ALLOW_UX));
        assert (Slave_SBZ(resp.ALLOW_UW));
        assert (Slave_SBZ(resp.ALLOW_UR));
    elsif (resp.STRW == EL3) then
        assert (resp.ALLOW_UX == resp.ALLOW_PX);
        assert (resp.ALLOW_UW == resp.ALLOW_PW);
        assert (resp.ALLOW_UR == resp.ALLOW_PR);

    if resp.BYPASS == '0' && req.SSV == '1' then
        assert (resp.STRW != EL1_S2);

    if resp.BYPASS == '1' || resp.STRW == EL1_S2 then
        // ATTR_OVR field is valid
        if req.ATST == '1' then
            // Must be 0x0020
            assert (resp.ATTR_OVR.MemAttr == '0000');
            assert (resp.ATTR_OVR.MTCFG   == '0');
            assert (resp.ATTR_OVR.SHCFG   == SHCFG_UseIncoming);
    else
        // ASID Field is valid
        if resp.STRW == EL3 then
            assert (UInt(resp.ASID) == 0);

    if resp.BYPASS == '1' then
        assert (Slave_SBZ(resp.VMID));
    elsif (resp.STRW IN {EL2, EL3} || req.SEC_SID == '1') then
        assert (UInt(resp.VMID) == 0);

    if resp.BYPASS == '1' then
        if req.SEC_SID == '0' then
            assert (resp.NSOVR == '1');

    if resp.BYPASS == '1' then
        // bits are 0 BUT this actually means DCP and DRE are permitted
        assert (Slave_SBZ(resp.DCP));
        assert (Slave_SBZ(resp.DRE));

    if resp.BYPASS == '0' && req.SEC_SID == '0' then
        assert (resp.STRW != EL3);

    if resp.BYPASS == '0' && req.SEC_SID == '1' then
        assert (resp.STRW IN {EL1, EL3});

    if resp.BYPASS == '0' && req.ATST == '1' then
        assert (resp.STRW == EL1_S2);

    if resp.BYPASS == '1' && req.SSV == '1' then
        assert (resp.BP_TYPE != BP_TYPE_StreamBypassNoSSV);

    if resp.BYPASS == '1' then
```

```
        assert (Slave_SBZ(resp.OA));
    else
        integer max_output_addr = (1<<OAS_MSB(DTIProtocolState.CurrentOAS))-1;
        assert (UInt(resp.OA) <= max_output_addr);
        // This address must be the first byte in a region of size that is given
        // by the TRANS_RNG field.
        integer trans_rng_msb = TRANS_RNG_MSB(resp.TRANS_RNG);
        integer trans_rng_mask = (1 << trans_rng_msb) - 1;
        assert ((resp.OA AND trans_rng_mask<51:0>) == 0<51:0>);

    if UInt(req.IA) >= (1<<(OAS_MSB(DTIProtocolState.CurrentOAS))) then
        assert (resp.BYPASS == '0');

    if (resp.BYPASS == '1'
            && resp.BP_TYPE == BP_TYPE_GlobalBypass) then
        assert (Slave_SBZ(resp.CONT));

    // Permission checks between req and resp
    effective_InD = ((resp.INSTCFG == INSTCFG_UseIncoming && req.InD == '1')
        || resp.INSTCFG == INSTCFG_Instruction);

    effective_PnU = ((resp.PRIVCFG == PRIVCFG_UseIncoming && req.PnU == '1')
        || resp.PRIVCFG == PRIVCFG_Privileged);

    req_R = (req.PERM == R && !effective_InD) || req.PERM == RW;
    req_W = (req.PERM == W) || (req.PERM == RW);
    req_X = (req.PERM == R) && effective_InD;

    effective_NS = if resp.NSOVR == '1' then resp.NS else req.NS;

    // The DTI slave can only return a DTI_TBU_TRANS_RESP message when permission
    // is granted for the transaction that is described in the translation request.
    // Check the permissions here:
    if (resp.BYPASS == '0') then
        if effective_PnU then
            if req_R then assert(resp.ALLOW_PR == '1');
            if req_W then assert(resp.ALLOW_PW == '1');
            if req_X then assert(resp.ALLOW_PX == '1');
        else
            if req_R then assert(resp.ALLOW_UR == '1');
            if req_W then assert(resp.ALLOW_UW == '1');
            if req_X then assert(resp.ALLOW_UX == '1');
    elsif <req.SEC_SID,effective_NS> == '11' then
        if req_X then assert(resp.ALLOW_NSX == '1');

    return;

// TransFaultCheck()
// =================
// Assertions for well-formedness of fault messages, including
// restrictions based on the originating translation request.

TransFaultCheck(DTI_TBU_TRANS_FAULT fault, DTI_TBU_TRANS_REQ req)

    if req.PERM == SPEC then
        assert (fault.FAULT_TYPE != FAULT_TYPE_Abort);

    if fault.FAULT_TYPE IN {FAULT_TYPE_NonAbort, FAULT_TYPE_Abort} then
        assert (Slave_SBZ(fault.CONT));

    if fault.FAULT_TYPE IN {FAULT_TYPE_NonAbort, FAULT_TYPE_Abort} then
        assert (fault.DO_NOT_CACHE == '1');

    if req.ATST == '1' then
        assert (fault.FAULT_TYPE != FAULT_TYPE_GlobalDisabled);

    return;

// InvReqCheck()
// =============
// Assertions for well-formedness of invalidation request messages.

InvReqCheck(DTI_TBU_INV_REQ inv_req)

    if (inv_req.OPERATION IN {
            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}) then
        assert(inv_req.INC_ASET1 == '1');

    if use_VMID(inv_req.OPERATION) then
        assert(UInt(inv_req.RANGE) <= 4);

    return;
```

B.3 Invalidation

This section provides an example implementation for checking an invalidation request against cache tags and scope fields.

```
// MatchMSBs()
// ===========
// Returns TRUE if two fields match, except for the specified number
// of least-significant bits.

boolean MatchMSBs(bits(N) A, bits(N) B, integer LSBs)
    integer mask_lsbs = (1 << LSBs) - 1;
    bits(N) mask = NOT mask_lsbs<N-1:0>;
    return (A AND mask) == (B AND mask);

// MatchInvalidationGlobal()
// =========================
// Returns TRUE if an invalidation request must invalidate a global entry
// cache entry with a given tag.

boolean MatchInvalidationGlobal(DTI_TBU_INV_REQ inv_req, GlobalCacheTag global_tag)
    case inv_req.OPERATION of
        when CFGI_S_ALL
            return (global_tag.ATST == '0'
                && global_tag.SEC_SID == '1');
        when CFGI_NS_ALL
            return global_tag.SEC_SID == '0';
        when INV_ALL
            return TRUE;
    return FALSE;

// MatchInvalidationConfig()
// =========================
// Returns TRUE if an invalidation request must invalidate a configuration
// cache entry with a given tag.

boolean MatchInvalidationConfig(DTI_TBU_INV_REQ inv_req, ConfigCacheTag cfg_tag)

    if cfg_tag.SSV == '0' then
        cfg_tag.SSID = 0<19:0>;

    case inv_req.OPERATION of
        when CFGI_S_ALL
            return (cfg_tag.SEC_SID == '1');
        when CFGI_S_SID
            return (cfg_tag.SEC_SID == '1'
                && MatchMSBs(cfg_tag.SID, inv_req.SID, UInt(inv_req.RANGE)));
        when CFGI_S_SID_SSID
            return (cfg_tag.SEC_SID == '1'
                && cfg_tag.SID == inv_req.SID
                && cfg_tag.SSID == inv_req.SSID);
        when CFGI_NS_ALL
            return (cfg_tag.SEC_SID == '0');
        when CFGI_NS_SID
            return (cfg_tag.SEC_SID == '0'
                && MatchMSBs(cfg_tag.SID, inv_req.SID, UInt(inv_req.RANGE)));
        when CFGI_NS_SID_SSID
            return (cfg_tag.SEC_SID == '0'
                && cfg_tag.SID == inv_req.SID
                && cfg_tag.SSID == inv_req.SSID);
        when INV_ALL
            return TRUE;
    return FALSE;

// MatchASET()
// ===========
// Returns TRUE if the given invalidation request INC_ASET1 value and the
// given translation response ASET value would match for invalidation.

boolean MatchASET(bit inc_aset1, bit aset)
    if inc_aset1 == '0' then
        return aset == '0';
    else
        return TRUE;

// MatchInvalidationTLB()
// ======================
// Returns TRUE if an invalidation request must invalidate a TLB entry
// with a given pair of tag and scope fields.

boolean MatchInvalidationTLB(DTI_TBU_INV_REQ inv_req, TLBCacheTag tlb_tag, TLBCacheFields tlb_entry)
```

```
case inv_req.OPERATION of
    when TLBI_S_EL1_ALL
        return (tlb_tag.SEC_SID == '1'
            && tlb_tag.STRW == EL1);
    when TLBI_S_EL1_VAA
        return (tlb_tag.SEC_SID == '1'
            && tlb_tag.STRW == EL1
            && MatchMSBs(tlb_tag.IA, inv_req.VA, INVAL_RNG_MSB(tlb_entry.INVAL_RNG)));
    when TLBI_S_EL1_ASID
        return (tlb_tag.SEC_SID == '1'
            && tlb_tag.STRW == EL1
            && MatchASET(inv_req.INC_ASET1, tlb_tag.ASET)
            && (tlb_entry.GLOBAL == '0' && tlb_tag.ASID == inv_req.ASID));
    when TLBI_S_EL1_VA
        return (tlb_tag.SEC_SID == '1'
            && tlb_tag.STRW == EL1
            && MatchMSBs(tlb_tag.IA, inv_req.VA, INVAL_RNG_MSB(tlb_entry.INVAL_RNG))
            && MatchASET(inv_req.INC_ASET1, tlb_tag.ASET)
            && (tlb_entry.GLOBAL == '1' || tlb_tag.ASID == inv_req.ASID));
    when TLBI_NS_EL1_ALL
        return (tlb_tag.SEC_SID == '0'
            && tlb_tag.STRW IN {EL1, EL1_S2});
    when TLBI_NS_EL1_S1_VMID
        return (tlb_tag.SEC_SID == '0'
            && tlb_tag.STRW == EL1
            && MatchMSBs(tlb_tag.VMID, inv_req.VMID, UInt(inv_req.RANGE)));
    when TLBI_NS_EL1_S12_VMID
        return (tlb_tag.SEC_SID == '0'
            && tlb_tag.STRW IN {EL1, EL1_S2}
            && MatchMSBs(tlb_tag.VMID, inv_req.VMID, UInt(inv_req.RANGE)));
    when TLBI_NS_EL1_VAA
        return (tlb_tag.SEC_SID == '0'
            && tlb_tag.STRW == EL1
            && MatchMSBs(tlb_tag.IA, inv_req.VA, INVAL_RNG_MSB(tlb_entry.INVAL_RNG))
            && MatchMSBs(tlb_tag.VMID, inv_req.VMID, UInt(inv_req.RANGE)));
    when TLBI_NS_EL1_ASID
        return (tlb_tag.SEC_SID == '0'
            && tlb_tag.STRW == EL1
            && MatchMSBs(tlb_tag.VMID, inv_req.VMID, UInt(inv_req.RANGE))
            && MatchASET(inv_req.INC_ASET1, tlb_tag.ASET)
            && (tlb_entry.GLOBAL == '0' && tlb_tag.ASID == inv_req.ASID));
    when TLBI_NS_EL1_VA
        return (tlb_tag.SEC_SID == '0'
            && tlb_tag.STRW == EL1
            && MatchMSBs(tlb_tag.IA, inv_req.VA, INVAL_RNG_MSB(tlb_entry.INVAL_RNG))
            && MatchMSBs(tlb_tag.VMID, inv_req.VMID, UInt(inv_req.RANGE))
            && MatchASET(inv_req.INC_ASET1, tlb_tag.ASET)
            && (tlb_entry.GLOBAL == '1' || tlb_tag.ASID == inv_req.ASID));
    when TLBI_NS_EL1_S2_IPA
        return (tlb_tag.SEC_SID == '0'
            && tlb_tag.STRW == EL1_S2
            && MatchMSBs(tlb_tag.IA, inv_req.IPA, INVAL_RNG_MSB(tlb_entry.INVAL_RNG))
            && MatchMSBs(tlb_tag.VMID, inv_req.VMID, UInt(inv_req.RANGE)));
    when TLBI_NS_EL2_ALL
        return (tlb_tag.SEC_SID == '0'
            && tlb_tag.STRW == EL2);
    when TLBI_NS_EL2_VAA
        return (tlb_tag.SEC_SID == '0'
            && tlb_tag.STRW == EL2
            && MatchMSBs(tlb_tag.IA, inv_req.VA, INVAL_RNG_MSB(tlb_entry.INVAL_RNG)));
    when TLBI_NS_EL2_ASID
        return (tlb_tag.SEC_SID == '0'
            && tlb_tag.STRW == EL2
            && MatchASET(inv_req.INC_ASET1, tlb_tag.ASET)
            && (tlb_entry.GLOBAL == '0' && tlb_tag.ASID == inv_req.ASID));
    when TLBI_NS_EL2_VA
        return (tlb_tag.SEC_SID == '0'
            && tlb_tag.STRW == EL2
            && MatchMSBs(tlb_tag.IA, inv_req.VA, INVAL_RNG_MSB(tlb_entry.INVAL_RNG))
            && MatchASET(inv_req.INC_ASET1, tlb_tag.ASET)
            && (tlb_entry.GLOBAL == '1' || tlb_tag.ASID == inv_req.ASID));
    when TLBI_S_EL3_ALL
        return (tlb_tag.SEC_SID == '1'
            && tlb_tag.STRW == EL3);
    when TLBI_S_EL3_VA
        return (tlb_tag.SEC_SID == '1'
            && tlb_tag.STRW == EL3
            && MatchMSBs(tlb_tag.IA, inv_req.VA, INVAL_RNG_MSB(tlb_entry.INVAL_RNG))
            && MatchASET(inv_req.INC_ASET1, tlb_tag.ASET));
    when INV_ALL
        return TRUE;
return FALSE;
```