Arm® CoreLink™ MMU-600 System Memory Management Unit

Revision: r1p0

Technical Reference Manual

arm

100310_0100_00_en

Arm® CoreLink™ MMU-600 System Memory Management Unit

Technical Reference Manual

Release Information

Document History

Issue	Date	Confidentiality	Change
0000-00	16 December 2016	Confidential	First release for r0p0
0000-01	19 May 2017	Confidential	Second release for r0p0
0001-00	23 August 2017	Confidential	First release for r0p1
0001-01	10 November 2017	Non-Confidential	Second release for r0p1
0002-00	15 December 2017	Non-Confidential	First release for r0p2
0100-00	20 March 2018	Non-Confidential	First release for r1p0

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Additional Notices

Some material in this document is based on IEEE 754-2008 IEEE Standard for Binary Floating-Point Arithmetic. The IEEE disclaims any responsibility or liability resulting from the placement and use in the described manner.

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.

Unrestricted Access is an Arm internal classification.

Product Status

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

Web Address

http://www.arm.com

Contents
Arm® CoreLink™ MMU-600 System Memory Management Unit Technical Reference Manual

Preface

This preface introduces the *Arm® CoreLink™ MMU-600 System Memory Management Unit Technical Reference Manual*.

It contains the following:

About this book

This book is for the Arm® CoreLink™ MMU-600 System Memory Management Unit.

Product revision status

The r*m*p*n* identifier indicates the revision status of the product described in this book, for example, r*1*p*2*, where:

r*m* Identifies the major revision of the product, for example, r1.

p*n* Identifies the minor revision or modification status of the product, for example, p2.

Intended audience

This book is written for system designers, system integrators, and programmers who are designing or programming a *System-on-Chip* (SoC) that uses the MMU-600.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction
This chapter provides an overview of the MMU-600.

Chapter 2 Functional description
This chapter describes the functionality of the MMU-600.

Chapter 3 Programmers model
This chapter describes the MMU-600 programmers model.

Appendix A Signal descriptions
This appendix describes the MMU-600 external signals.

Appendix B Software initialization examples
This appendix provides examples of how software can initialize and enable the MMU-600.

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

Glossary

The Arm® Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm 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 *Arm® 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.

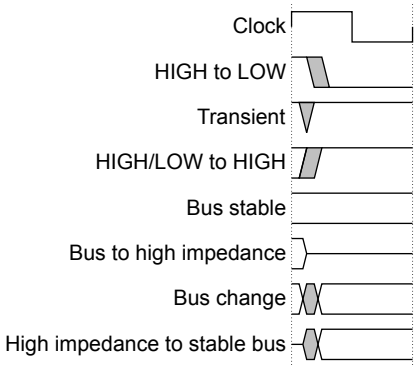

Figure 1 Key to timing diagram conventions

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

Arm publications

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

- *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3.0 and version 3.1* (ARM IHI 0070).
- *Arm® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile* (ARM DDI 0487).
- *Arm® AMBA® Distributed Translation Interface (DTI) Protocol Specification* (100225).
- *Arm® AMBA® APB Protocol Specification* (ARM IHI 0024).
- *Arm® CoreSight™ Architecture Specification* (ARM IHI 0029).
- *Arm® AMBA® AXI and ACE Protocol Specification, AXI3, AXI4, AXI5, ACE and ACE5* (ARM IHI 0022).
- *Arm® AMBA® 4 AXI4-Stream Protocol Specification* (ARM IHI 0051).
- *AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces* (ARM IHI 0068).
- *Arm® CoreLink™ LPD-500 Low Power Distributor Technical Reference Manual* (100361).
- *Arm® Server Base System Architecture* (DEN-0029).

The following confidential books are only available to licensees:

- *Arm® CoreLink™ MMU-600 System Memory Management Unit Configuration and Integration Manual* (100311).
- *Arm® CoreLink™ ADB-400 AMBA® Domain Bridge User Guide* (ARM DUI 0615).

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 CoreLink MMU-600 System Memory Management Unit Technical Reference Manual*.
- The number 100310_0100_00_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 provides an overview of the MMU-600.

It contains the following sections:

1.1 About the MMU-600

The MMU-600 is a *System*-level *Memory Management Unit* (SMMU) that translates an input address to an output address. This translation is based on address mapping and memory attribute information that is available in the MMU-600 internal registers and translation tables.

The MMU-600 implements the Arm SMMU architecture version 3.1, SMMUv3.1, as defined by the *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3.0 and version 3.1*.

An address translation from an input address to an output address is described as a stage of address translation. The MMU-600 can perform:

- Stage 1 translations that translate an input *virtual address* (VA) to an output *physical address* (PA) or *intermediate physical address* (IPA).
- Stage 2 translations that translate an input IPA to an output PA.
- Combined stage 1 and stage 2 translations that translate an input VA to an IPA, and then translate that IPA to an output PA. The MMU-600 performs translation table walks for each stage of the translation.

In addition to translating an input address to an output address, a stage of address translation also defines the memory attributes of the output address. With a two-stage translation, the stage 2 translation can modify the attributes that the stage 1 translation defines. A stage of address translation can be disabled or bypassed, and the MMU-600 can define memory attributes for disabled and bypassed stages of translation.

The MMU-600 uses inputs from the requesting master to identify a context. Configuration tables in memory tell the MMU-600 how to translate each context, such as which translation tables to use.

The MMU-600 can cache the result of a translation table lookup in a *Translation Lookaside Buffer* (TLB). It can also cache configuration tables in a configuration cache.

The MMU-600 contains the following key components:

- *Translation Buffer Units* (TBUs) that use a TLB to cache translation tables.
- A *Translation Control Unit* (TCU) that controls and manages address translations.
- *Distributed Translation Interface* (DTI) interconnect components that connect multiple TBUs to the TCU.

Related information

2.1 About the functions on page 2-22

1.2 Compliance

The MMU-600 complies with, or implements, the specifications that this section describes. This *Technical Reference Manual* (TRM) complements architecture reference manuals, architecture specifications, protocol specifications, and relevant external standards. It does not duplicate information from these sources.

1.2.1 Arm architecture

The MMU-600 implements parts of the ARMv8 *Virtual Memory System Architecture* (VMSA), as defined by the *Arm® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*. The SMMUv3 architecture describes the parts of VMSA that apply to the MMU-600.

1.2.2 SMMU architecture

The MMU-600 implements the SMMUv3.1 architecture, as defined by the *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3.0 and version 3.1*.

Related information

2.4.1 SMMUv3 support on page 2-54

1.2.3 AMBA DTI protocol

The MMU-600 implements the *Distributed Translation Interface* (DTI) protocol, as defined by the *Arm® AMBA® Distributed Translation Interface (DTI) Protocol Specification*.

The DTI interfaces use an AXI4-Stream interface, as defined by the *Arm® AMBA® 4 AXI4-Stream Protocol Specification*.

Related information

2.3.1 DTI overview on page 2-36

1.2.4 AMBA ACE5-Lite and AMBA® AXI5 protocol

The MMU-600 complies with the AMBA ACE5-Lite protocol. A variant of the TBU also supports the ACE5 protocol when used for protection only.

See the *Arm® AMBA® AXI and ACE Protocol Specification, AXI3, AXI4, AXI5, ACE and ACE5* for more information.

Related information

2.4.2 AMBA support on page 2-57

1.2.5 AMBA APB protocol

The MMU-600 complies with the AMBA APB4 protocol, as defined by the *Arm® AMBA® APB Protocol Specification*.

1.3 Features

The MMU-600 provides the following features:

- Compliance with the SMMUv3.1 architecture:
  - Support for Stage 1 translation, Stage 2 translation, and Stage 1 followed by stage 2 translation.
  - Support for ARMv8 AArch32 and AArch64 translation table formats.
  - Support for 4KB, 16KB and 64KB granule sizes in AArch64 format.
  - Support for *Page Request Interface* (PRI), as defined by SMMUv3. PRI is an optional PCIe ATS extension that enables support for unpinned memory in PCIe.
  - Masters can be stalled while a processor handles translation faults, enabling software support for demand paging.
  - Configuration tables in memory can support millions of active translation contexts.
  - Queues in memory perform MMU-600 management, no requirement to stall a processor when it accesses the MMU-600.
  - Support for *PCI Express* (PCIe) integration, including *Address Translation Services* (ATS) and *Process Address Space IDs* (PASIDs).
  - Support for *Generic Interrupt Controller* (GIC) integration, with *Message Signaled Interrupts* (MSIs) supported for common interrupt types.
  - A *Performance Monitoring Unit* (PMU) in each TBU and TCU that enables MMU-600 performance to be investigated.
  - *Reliability, Serviceability and Availability* (RAS) features for cache corruption detection and correction.
- Support for AMBA interfaces, including:
  - ACE5-Lite TBU transaction interfaces that support cache stash transactions, deallocating transactions, and cache maintenance.
  - Option to disable cache maintenance operations on a TBU, a sideband channel protection feature.
  - An architected AXI5 extension that communicates per-transaction translation stream information.
  - An ACE5-Lite + *Distributed Virtual Memory* (DVM) TCU table walk interface that enables ARMv8.2 processors to perform shared TLB invalidate operations without accessing the MMU-600 directly.
  - An ACE5 Low Power extension that enables the TCU to subscribe to DVM TLB invalidate requests on powerup and powerdown without reprogramming the DTI interconnect.
  - AMBA DTI communication between the TCU and TBUs, enabling masters to request translations and implement TBU functionality internally.
  - Support for the AMBA *Low-Power Interface* (LPI) Q-Channel so that standard controllers can control power and clock gating.
  - AXI5 **WAKEUP** signaling on all interfaces, including DTI and APB interfaces.
  - Access protection for ACE interfaces. ACE protection aligns with the restrictions that ACE5 defines for ACE usage of the Untranslated_Transactions extension.
- Support for flexible integration:
  - A configurable number of TBUs can be placed close to the masters being translated.
  - Communication between TBU and TCU over AXI4-Stream, supported using the supplied DTI interconnect components, or any other AXI4-Stream interconnect.
  - DTI interconnect components support hierarchical topologies, and control of the tradeoff between number of wires and DTI bandwidth.
- Support for high-performance translation:
  - Scalable configurable micro TLB and *Main TLB* (MTLB) in the TBU can reduce the number of translation requests to the TCU.
  - TBU direct indexing and MTLB partitioning enable the use of MTLB entries to be managed outside the TBU, improving real-time translation performance.
  - Optimization to store all architecturally defined page and block sizes, including contiguous page and block entries, as a single entry in the TBU and TCU TLBs.
  - Per-TBU prioritization in the TCU enables high-priority transaction streams to be translated before low-priority streams.

- TCU prefetch of translation tables, which can be enabled on a per-context basis, improving translation performance for real-time masters that access memory linearly.
- *Hit-Under-Miss* (HUM) support in the TBU enables transactions with different AXI IDs to be propagated out of order, when a translation is available.
- TBU detection of multiple transactions that require the same translation so that only one TBU request to the TCU is required.
- TCU detection of multiple translations that require the same table in memory so that only one TCU memory request is required.
- Multi-level, multi-stage walk caches in the TCU reduce translation cost by performing only part of the table walk process on a miss.
- A configurable number of concurrent translations in the TBU and TCU promotes high translation throughput.

1.4 Interfaces

Both the TCU and TBU support the following common interfaces:

- DTI.
- Tie-offs.
- Interrupts.
- PMU snapshot.
- Test and debug.
- LPI clock gating.
- LPI powerdown.

The TCU also supports the following interfaces:

- Programming.
- System coherency.
- *Queue and Table Walk* (QTW)/DVM.

The TBU also supports the following interfaces:

- *Transaction slave* (TBS).
- *Transaction master* (TBM).

Related information

2.2 Interfaces on page 2-28

1.5 Configurable options

The MMU-600 is highly configurable and provides configuration options for each of the main blocks.

For the TCU, you can configure:

- The size of each of the caches.
- The data width of the QTW/DVM interface.
- The number of translations that can be performed at the same time.
- The number of translation requests that can be accepted from all DTI masters.

For the TBU, you can configure:

- Write data buffer depth.
- The size of each of the caches.
- The number of transactions that can be translated at the same time.
- The number of outstanding read and write transactions that the TBM interface supports.
- The width of data, ID, user, StreamID, and SubstreamID signals on the TBS and TBM interfaces.

——— **Note** ———

Depths are specified as a discrete number of entries.

———————————

You can also configure the DTI interconnect components to meet your system requirements.

1.6 Product documentation and design flow

This section describes the MMU-600 documentation in relation to the design flow.

1.6.1 Documentation

The MMU-600 documentation is as follows:

Technical Reference Manual

The *Technical Reference Manual* (TRM) describes the functionality and the effects of functional options on the behavior of the MMU-600. It is required at all stages of the design flow. The choices that are made in the design flow can mean that some behaviors that are described in the TRM are not relevant. If you are programming the MMU-600, then contact:

- The implementer to determine:
  - The build configuration of the implementation.
  - What integration, if any, was performed before implementing the MMU-600.
- The integrator to determine the pin configuration of the device that you are using.

Configuration and Integration Manual

The *Configuration and Integration Manual* (CIM) describes:

- The available build configuration options and related issues in selecting them.
- How to integrate the MMU-600 into a SoC. This section includes describing the pins that the integrator must tie off to configure the macrocells for the required integration.
- The processes to sign off the configuration, integration, and implementation of the design.

The Arm product deliverables include reference scripts and information about using them to implement your design. Reference methodology flows that Arm supplies are example reference implementations. Contact your EDA vendor for EDA tool support.

The CIM is a confidential book that is only available to licensees.

1.6.2 Design flow

The MMU-600 is delivered as synthesizable RTL. Before it can be used in a product, it must go through the following processes:

Implementation

The implementer configures and synthesizes the RTL to produce a hard macrocell. This process might include integrating RAMs into the design.

Integration

The integrator connects the implemented design into a SoC. Integration includes connecting the design to a memory system and peripherals.

Programming

The system programmer develops the software to configure and initialize the MMU-600, and tests the required application software.

Each process is separate, and can include implementation and integration choices that affect the behavior and features of the MMU-600.

The operation of the final device depends on:

Build configuration

The implementer chooses the options that affect how the RTL source files are pre-processed. These options usually include or exclude logic that affects one or more of the following:

- Area.
- Maximum frequency.
- Features of the resulting macrocell.

Configuration inputs

The integrator configures some features of the MMU-600 by tying inputs to specific values. These configurations affect the start-up behavior before any software configuration is made.

Software configuration

The programmer configures the MMU-600 by programming particular values into registers. This configuration affects the behavior of the MMU-600.

Related information

1.2 Compliance on page 1-13
1.5 Configurable options on page 1-17

1.7 Product revisions

This section describes the differences in functionality between product revisions:

r0p0	First release.
r0p0-r0p1	The following changes apply to this release: • Modified bits in TCU_CTRL. • Modified bits in TBU_CTRL.
r0p1-r0p2	This release has no functional changes.

Chapter 2
Functional description

This chapter describes the functionality of the MMU-600.

It contains the following sections:

2.1 About the functions

The major functional blocks of the MMU-600 are the TCU, TBU, and DTI interconnect.

The following figure shows an example system that uses the MMU-600.

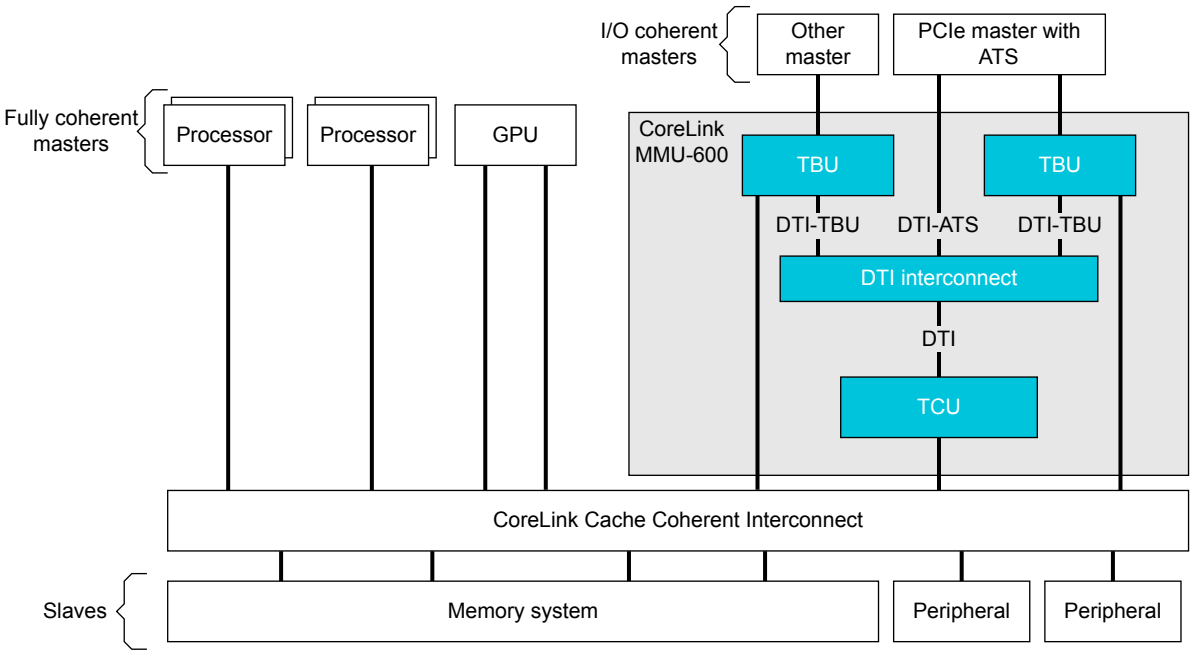

Figure 2-1 Example system with the MMU-600

The MMU-600 contains the following key components:

***Translation Buffer Unit* (TBU)**
The TBU contains *Translation Lookaside Buffers* (TLBs) that cache translation tables. The MMU-600 implements at least one TBU for each connected master, and these TBUs are local to the corresponding master.

***Translation Control Unit* (TCU)**
The TCU controls and manages the address translations. The MMU-600 implements a single TCU. In MMU-600-based systems, the AMBA DTI protocol defines the standard for communicating with the TCU.

DTI interconnect
The DTI interconnect connects multiple TBUs to the TCU.

When an MMU-600 TBU receives a transaction on the TBS interface, it looks for a matching translation in its TLBs. If it has a matching translation, it uses it to translate the transaction and outputs the transaction on the TBM interface. If it does not have a matching translation, it requests a new translation from the TCU using the DTI interface.

When the TCU receives a DTI translation request, it uses the QTW interface to perform:

- Configuration table walks, which return configuration information for the translation context.
- Translation table walks, which return translation information specific to the transaction address.

The TCU contains caches that reduce the number of configuration and translation table walks that are to be performed. Sometimes no walks are required.

When the TBU receives the translation from the TCU, it stores it in its TLBs. If the translation was successful, the TBU uses it to translate the transaction, otherwise it terminates it.

A processor controls the TCU by:

- Writing commands to a Command queue in memory.
- Receiving events from an Event queue in memory.
- Writing to its configuration registers using the programming interface.

See the *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3.0 and version 3.1* for more information about translation and how software communicates with the TCU.

This section contains the following subsections:

- *2.1.1 Translation Control Unit* on page 2-23.
- *2.1.2 Translation Buffer Unit* on page 2-25.
- *2.1.3 DTI interconnect* on page 2-26.

2.1.1 Translation Control Unit

A typical SMMUv3-based system includes a single *Translation Control Unit* TCU. The TCU is usually the largest block in the system, and performs several roles.

The TCU:

- Manages the memory queues.
- Performs translation table walks.
- Performs configuration table walks.
- Implements backup caching structures.
- Implements the SMMU programmers model.

The following figure shows the TCU.

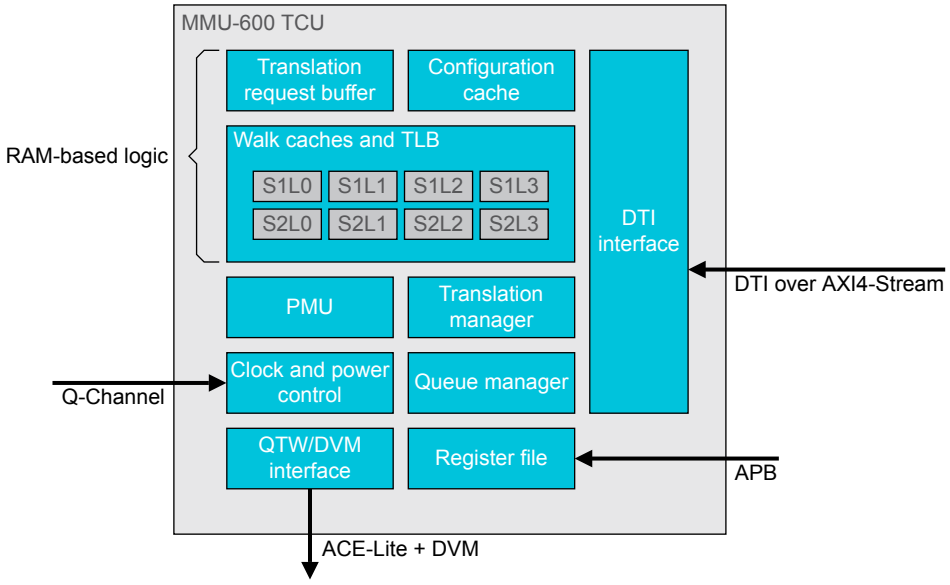

Figure 2-2 MMU-600 TCU

The TCU consists of:

Walk caches

The TCU includes separate four-way set-associative walk caches to store results of translation table walks. During MMU-600 configuration, the cache line entries are split to create separate walk caches that are reserved for:

- Stage 1 level 0 table entries.
- Stage 1 level 1 table and block entries.
- Stage 1 level 2 table and block entries.
- Stage 1 level 3 table entries.
- Stage 2 level 0 table entries.
- Stage 2 level 1 table and block entries.
- Stage 2 level 2 table and block entries.
- Stage 2 level 3 table entries.

To enable and disable the walk cache for a particular stage and level of translation, use the TCU_CTRL register. If an error occurs for a cache line entry, the TCU_ERRSTATUS register identifies the affected entry.
The walk cache is useful in cases where a translation request results in a miss in other TCU caches. A subsequent hit in the walk cache requires only a single memory access to complete the translation table walk and fetch the required descriptor.

Configuration cache

The configuration caches are 4-way set-associative cache structures that store configuration information. Each entry stores the *Context Descriptor* (CD) and *Stream Table Entry* (STE) contents for a translation context.

——— **Note** ———

The configuration cache does not cache the contents of intermediate configuration tables.

———————————

Translation manager

The translation manager manages translation requests that are in progress. All translation table walks and configuration table walks are hazard-checked to reduce the possibility of multiple transactions requesting duplicate walks.

Translation request buffer

The translation request buffer stores translation requests from TBUs when all translation manager slots are full. The translation request buffer supports more slots than the translation manager. When correctly configured, this buffer has enough space to store all translation requests that TBUs can issue simultaneously. This buffer therefore prevents the DTI interface from becoming blocked.

PMU

The PMU counts TCU performance-related events.

Clock and power control

The TCU has its own clock and power control, provided by the Q-Channel.

Queue manager

The queue manager manages all SMMUv3 Command queues and Event queues that are stored in memory.

QTW/DVM interface

The *Queue and Table Walk* (QTW)/*Distributed Virtual Memory* (DVM) interface is an ACE-Lite +DVM master interface.

Register file

The register file implements the SMMUv3 programmers model, as defined by the *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3.0 and version 3.1*.

DTI interface

The slave DTI interface uses the DTI protocol, typically over AXI4-Stream, to enable the TCU to communicate with a master component. For the MMU-600, the master component is either a TBU or a PCIe master.

Related information

2.2 Interfaces on page 2-28

2.3.8 TCU transaction handling on page 2-49

2.3.9 TCU prefetch on page 2-49

3.2 SMMU architectural registers on page 3-64

2.1.2 Translation Buffer Unit

A typical SMMUv3-based system includes multiple *Translation Buffer Units* (TBUs). Each TBU is located close to the component that it provides address translation for.

A TBU intercepts transactions and provides the required translation from a *Translation Lookaside Buffer* (TLB) if possible. If a TLB does not contain the required translation, the TBU requests translations from the TCU and then caches the translation in one of the TLBs.

The following figure shows the TBU.

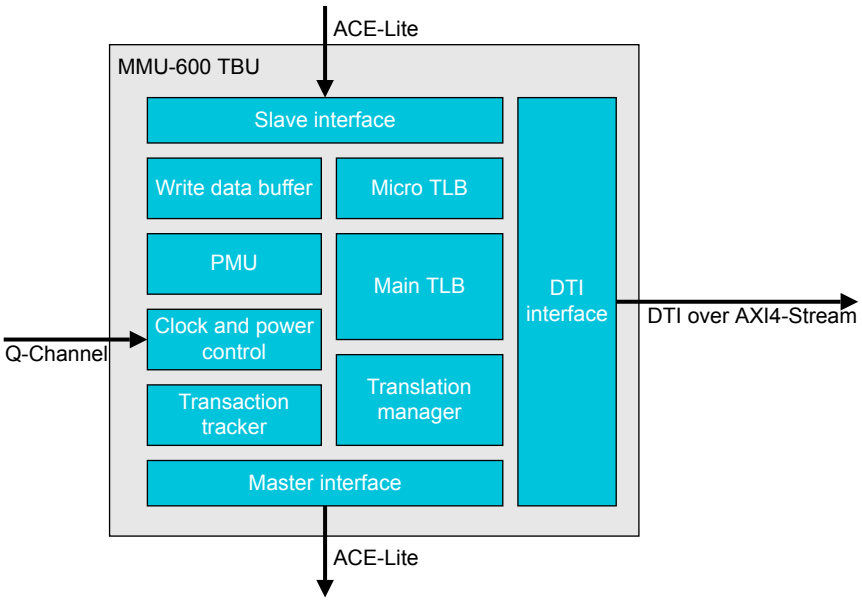

Figure 2-3 MMU-600 TBU

The TBU consists of:

Master and slave interfaces

These interfaces manage the TBS and TBM interfaces.

Micro TLB

The TBU compares incoming transactions with translations that are cached in the micro TLB before looking in the *Main TLB* (MTLB). The micro TLB is a fully associative TLB that provides configuration cache and TLB functionality. You can use a tie-off signal to configure the cache replacement policy as either round-robin or *Pseudo Least Recently Used* (PLRU).

Main TLB

Each TBU includes an optional *Main TLB* (MTLB) that caches translation table walk entries from:

- Stage 1 translations.
- Stage 2 translations.
- Stage 1 combined with stage 2 translations.

The MTLB is a configurable four-way set associative cache structure that uses a random cache replacement policy.
If multiple translation sizes are in use, a single transaction might require multiple lookups. Lookups are pipelined to permit a sustained rate of one lookup per cycle.
TBU direct indexing enables the MMU-600 to manage MTLB entries externally to the TBU. This improves the predictability of TBU performance, for bus masters that have real-time performance requirements.

Translation manager

The translation manager manages translation requests that are in progress. Each transaction occupies a translation slot until it is propagated downstream through the master interface. All transactions are hazard-checked to reduce the possibility of duplicate translation requests being sent to the TCU.
There is no restriction on the ordering of transactions with different AXI IDs. Transactions with different AXI IDs can be propagated downstream out-of-order.
All transactions with a given AXI ID value must remain ordered. The translation manager propagates such transactions when the translation is ready, provided no other transaction with the same AXI ID is already waiting.
See the *Arm® AMBA® AXI and ACE Protocol Specification, AXI3, AXI4, AXI5, ACE and ACE5* for more information about AXI transaction identifiers.

Write data buffer

The optional write data buffer enables write transactions with different AXI IDs to progress through the TBU out-of-order. It reorders the data to match the downstream transaction order.

PMU

The PMU counts TBU performance-related events.

Clock and power control

The TBU has its own clock and power control, provided by the Q-Channel.

DTI interface

The master DTI interface uses the DTI protocol, typically over AXI4-Stream, to enable the TBU to communicate with a slave component. For the MMU-600, the slave component is the TCU. Although you can implement DTI over different transport protocols, the MMU-600 interfaces use AXI4-Stream.

Transaction tracker

The transaction trackers manage outstanding read and write transactions, permitting invalidation and synchronization to take place without stalling the AXI interfaces.

Related information

2.3.4 TBU direct indexing and MTLB partitioning on page 2-46
3.2 SMMU architectural registers on page 3-64

2.1.3 DTI interconnect

The TBU and TCUs use a DTI interface to communicate. The DTI interconnect enables the DTI interface to use the AXI4-Stream transport protocol.

The DTI interconnect can connect any components that conform to the AXI4-Stream protocol, as defined by the *Arm® AMBA® Distributed Translation Interface (DTI) Protocol Specification*.

The DTI interconnect contains internal components that are hierarchically composable, that is, they can be connected in different ways to suit your system requirements. For example, within an MMU-600 system, you can use the switch component to combine the DTI interfaces of multiple TBUs into a single DTI interface. You can then connect the combined DTI interface to another DTI interconnect that is closer to the TCU. The DTI interconnect includes switch, sizer, and register slice components.

Switch

The switch connects multiple DTI masters, such as TBUs, to a DTI slave such as a TCU. The switch implements the following parallel networks:

- For TBU to TCU traffic, a network that connects multiple AXI4-Stream slave interfaces to a single AXI4-Stream master interface.
- For TCU to TBU traffic, a network that connects a single AXI4-Stream slave interface to multiple AXI4-Stream master interfaces.

Note

The switch does not store any data, and therefore does not require a Q-Channel clock-gating interface.

Sizer

The sizer connects channels that have different data widths, enabling different tradeoffs of bandwidth to area. The sizer supports conversion between any of the supported AXI4-Stream data widths:

- 1 byte.
- 4 bytes.
- 10 bytes.
- 20 bytes.

The sizer includes a Q-Channel interface to provide clock-gating control.

Register slice

Use the register slice to improve timing. The register slice includes a Q-Channel interface to provide clock-gating control.

The MMU-600 DTI interconnect components do not include a component to connect different clock and power domains. You can connect DTI interfaces in different clock and power domains by using the *Bidirectional AXI4-Stream* (BAS) configuration of the ADB-400 AMBA Domain Bridge.

Related information

2.3 Operation on page 2-36

2.2 Interfaces

The MMU-600 includes interfaces for each of the TCU, TBU, and DTI interconnect components.

The DTI interconnect consists of switch, sizer, and register slice components that can be connected separately, and therefore have their own interfaces.

The PMU snapshot interface is common to both TCU and TBU.

This section contains the following subsections:

- *2.2.1 TCU interfaces* on page 2-28.
- *2.2.2 TBU interfaces* on page 2-30.
- *2.2.3 DTI interconnect interfaces* on page 2-32.

2.2.1 TCU interfaces

The MMU-600 contains various TCU interfaces.

The following figure shows the TCU interfaces.

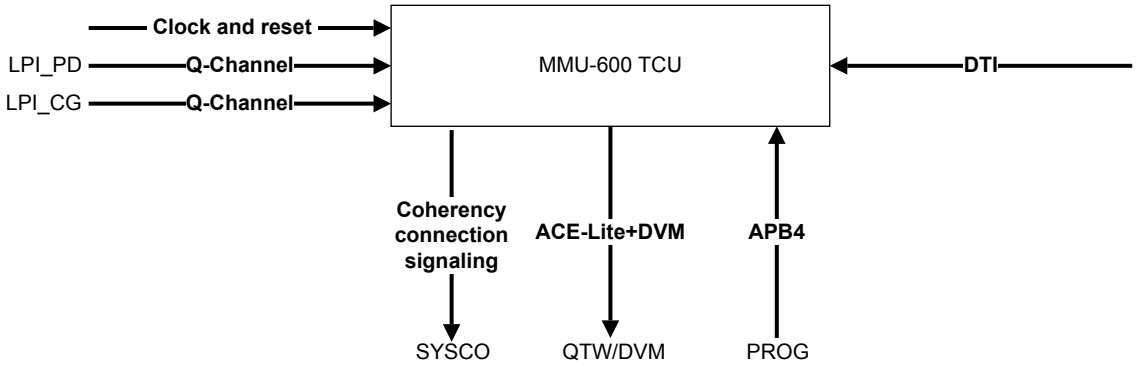

Figure 2-4 TCU interfaces

TCU Queue and Table Walk/Distributed Virtual Memory interface

The *Queue and Table Walk/Distributed Virtual Memory* (QTW/DVM) interface is an ACE-Lite+DVM master interface.

The QTW/DVM interface issues the following transaction types:

- ReadNoSnoop.
- WriteNoSnoop.
- ReadOnce.
- WriteUnique.
- DVM Complete.

The QTW/DVM interface uses the write address transaction ID signal **awid_qtw**, and the read address transaction ID signal, **arid_qtw**. The value of **awid_qtw** is always 0, and the value of **arid_qtw** depends on the transaction type. The following table shows the possible values of **arid_qtw**.

Table 2-1 Possible arid_qtw values

Transaction type	arid_qtw[n:1]	arid_qtw[0]
Translation table walk	Indicates the slot that is requesting the translation table walk	1
Command queue read	All bits = 0.	0
DVM Complete	All bits = 1.	0

To support 16-bit *Virtual Machine IDentifiers* (VMIDs), the interface provides DVMv8.1 support.

The interface does not issue cache maintenance operations or exclusive accesses.

Related information

2.3.7 Distributed Virtual Memory (DVM) messages on page 2-48

2.3.10 Error responses on page 2-50

AXI5 support on page 2-59

A.2 TCU QTW/DVM interface signals on page Appx-A-99

TCU PROG interface

The PROG interface is an AMBA APB4 slave interface. It enables software to program the MMU-600 internal registers and read the *Performance Monitoring Unit* (PMU) registers and the debug registers.

This interface runs synchronously with the other TCU interfaces.

The applicable address width for this interface depends on the value of `TCUCFG_NUM_TBU`:

- When `TCUCFG_NUM_TBU` = 14, the address width is 21 bits.
- When `TCUCFG_NUM_TBU` = 62, the address width is 23 bits.

Transactions are *Read-As-Zero, Writes Ignored* (RAZ/WI) when any of the following apply:

- An unimplemented register is accessed.
- **PSTRB[3:0]** is not `0b1111` for write transfers.
- **PPROT[1]** is not set to 0 for Secure register accesses.

See the *Arm® AMBA® APB Protocol Specification* for more information.

Related information

A.3 TCU programming interface signals on page Appx-A-102

TCU LPI_PD interface

This Q-Channel slave interface manages LPI powerdown for the TCU.

See the *AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces* for more information.

Related information

A.6 TCU LPI_PD interface signals on page Appx-A-105

TCU LPI_CG interface

This Q-Channel slave interface enables LPI clock-gating for the TCU.

See the *AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces* for more information.

Related information

A.7 TCU LPI_CG interface signals on page Appx-A-106

TCU DTI interface

The DTI interface manages communication between the TBUs and the TCU, using the DTI protocol. The DTI protocol can be conveyed over different transport layer mediums, including AXI4-Stream.

The TCU includes a slave DTI interface and each TBU includes a master DTI interface. To permit bidirectional communication, each DTI interface includes one AXI4-Stream master interface and one AXI4-Stream slave interface.

See the *Arm® AMBA® Distributed Translation Interface (DTI) Protocol Specification* and the *Arm® AMBA® 4 AXI4-Stream Protocol Specification* for more information.

Related information

2.3.1 DTI overview on page 2-36

A.8 TCU DTI interface signals on page Appx-A-107

TCU interrupt interfaces

This interface provides global, per-context, and performance interrupts.

Related information

A.9 TCU interrupt signals on page Appx-A-108

TCU SYSCO interface

The MMU-600 provides a hardware system coherency interface. This interface permits the TCU to remove itself from a coherency domain in response to an LPI request.

The SYSCO interface uses the **syscoreq** and **syscoack** handshake signals to enter or exit a coherency domain.

If the **sup_btm** signal is tied LOW:

- **syscoreq** is always driven LOW and **syscoack** is ignored.
- The TCU SYSCO interface is not used and can be left unconnected.

Related information

A.4 TCU SYSCO interface signals on page Appx-A-103

TCU tie-off signals

The TCU tie-off signals enable you to initialize various operating parameters on exit from reset state.

At reset, the value of each tie-off signal controls the respective bits in the SMMU_IDR0 Register.

Related information

A.10 TCU tie-off signals on page Appx-A-109

2.2.2 TBU interfaces

The following figure shows the TBU interfaces.

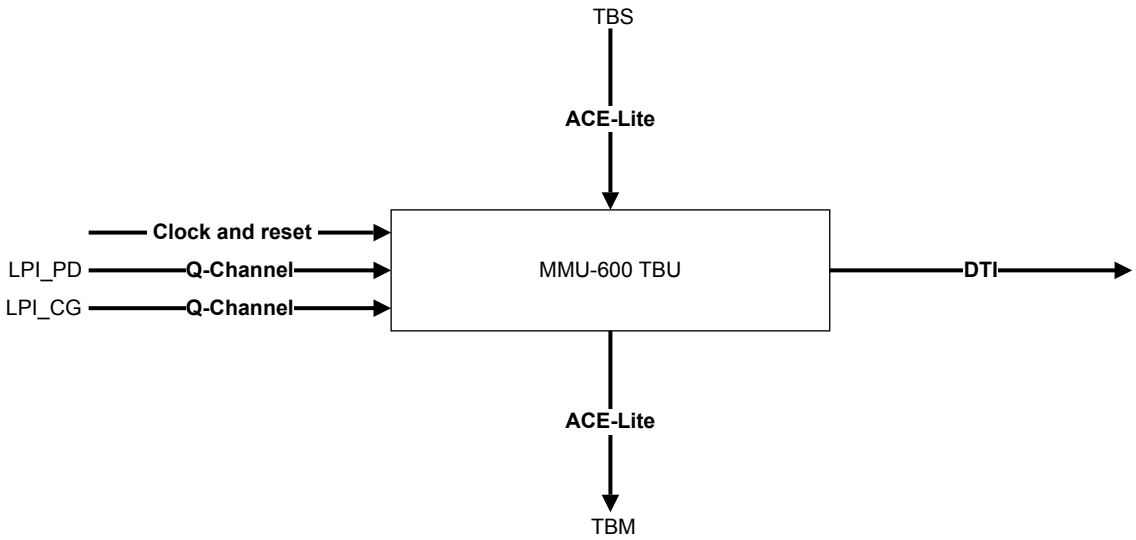

Figure 2-5 TBU interfaces

TBU TBS interface

The *transaction slave* (TBS) interface is an ACE5-Lite interface on which the TBU receives incoming untranslated memory accesses.

When the ACE TBU configuration is used, this interface is defined as an ACE interface rather than ACE-Lite.

This interface supports a 64-bit address width.

The interface implements optional signals to support the following AXI5 extensions:
- Untranslated_Transactions.
- Cache_Stash_Transactions.
- DeAllocation_Transactions.
- Wakeup_Signals.

The TBS interface supports ACE Exclusive accesses.

If a transaction is terminated in the TBU, the transaction tracker returns the transaction with the user-defined AXI **RUSER** and **BUSER** bits set to 0.

Related information

2.3.10 Error responses on page 2-50
A.12 TBU TBS interface signals on page Appx-A-111

TBU TBM interface

The TBM transaction master interface is an ACE5-Lite interface on which the TBU sends outgoing translated memory accesses.

When the ACE TBU configuration is used, this interface is defined as an ACE interface rather than ACE-Lite.

The AXI ID of a transaction on this interface is the same as the AXI ID of the corresponding transaction on the TBS interface.

This interface supports a 48-bit address width, and `TBUCFG_DATA_WIDTH` defines the data width.

This interface can issue read and write transactions until the outstanding transaction limit is reached. The MMU-600 provides parameters that permit you to configure:

- The outstanding read transactions limit.
- The outstanding write transactions limit.
- The total outstanding read and write transactions limit.

The interface implements optional signals to support the following AXI5 extensions:
- Untranslated_Transactions.
- Cache_Stash_Transactions.
- DeAllocation_Transactions.
- Wakeup_Signals.

When receiving an SLVERR or DECERR response to a downstream transaction, the TBM interface propagates the same response to the TBS interface.

The TBM interface supports ACE Exclusive accesses.

Related information

2.3.10 Error responses on page 2-50
2.4.2 AMBA support on page 2-57
A.13 TBU TBM interface signals on page Appx-A-115

TBU LPI_PD interface

This Q-Channel slave interface manages LPI powerdown for the TBU.

See the *AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces* for more information.

Related information

A.15 TBU LPI_PD interface signals on page Appx-A-120

TBU LPI_CG interface

This Q-Channel slave interface enables LPI clock-gating for the TBU.

See the *AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces* for more information.

Related information

A.16 TBU LPI_CG interface signals on page Appx-A-121

TBU DTI interface

The TBU DTI interface enables master devices with their own TLB and prefetch capability to request translations from the MMU-600. This interface uses the DTI-TBU protocol for communication between the TBU and the TCU.

The TCU includes a slave DTI interface and each TBU includes a master DTI interface. To permit bidirectional communication, each DTI interface includes one AXI4-Stream master interface and one AXI4-Stream slave interface.

See the *Arm® AMBA® Distributed Translation Interface (DTI) Protocol Specification* and the *Arm® AMBA® 4 AXI4-Stream Protocol Specification* for more information.

Related information

2.3.1 DTI overview on page 2-36

A.17 TBU DTI interface signals on page Appx-A-122

TBU interrupt interfaces

This interface provides global, per-context, and performance interrupts.

Related information

A.18 TBU interrupt signals on page Appx-A-123

TBU tie-off signals

The TBU tie-off signals enable you to initialize various operating parameters on exit from reset state.

At reset, the value of each tie-off signal controls the respective bits in the SMMU_IDR0 Register.

Related information

A.19 TBU tie-off signals on page Appx-A-124

2.2.3 DTI interconnect interfaces

The DTI interconnect includes interfaces for each of the switch, sizer, and register slice components.

DTI interconnect switch interfaces

The DTI interconnect switch component includes dedicated interfaces.

The following figure shows the DTI interconnect switch interfaces.

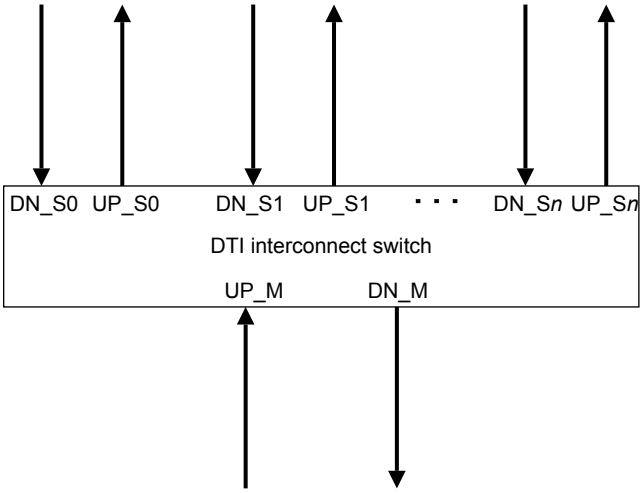

Figure 2-6 DTI interconnect switch interfaces

The following table provides more information about the switch interfaces.

Table 2-2 DTI interconnect switch interfaces

Interface	Interface type	Protocol	Description
DN_Sn	Slave	AXI4-Stream	Slave downstream interface. One DN_Sn interface is present for each slave interface.
UP_Sn	Master		Slave upstream interface. One UP_Sn interface is present for each slave interface.
DN_M	Master		Master downstream interface.
UP_M	Slave		Master upstream interface.

Note

The interconnect switch does not store any data, and therefore does not require a Q-Channel clock-gating interface.

DTI interconnect sizer interfaces

The DTI interconnect sizer component includes dedicated interfaces.

The following figure shows the DTI interconnect sizer interfaces.

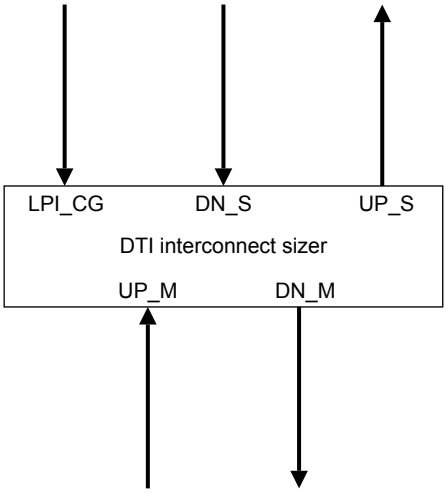

Figure 2-7 DTI interconnect sizer interfaces

The following table provides more information about the sizer interfaces.

Table 2-3 DTI interconnect sizer interfaces

Interface	Interface type	Protocol	Description
LPI_CG	Slave	Q-Channel	Clock-gating interface.
DN_S	Slave	AXI4-Stream	Slave downstream interface.
UP_S	Master		Slave upstream interface.
DN_M	Master		Master downstream interface.
UP_M	Slave		Master upstream interface.

DTI interconnect register slice interfaces

The DTI interconnect register slice component includes dedicated interfaces.

The following figure shows the DTI interconnect register slice interfaces.

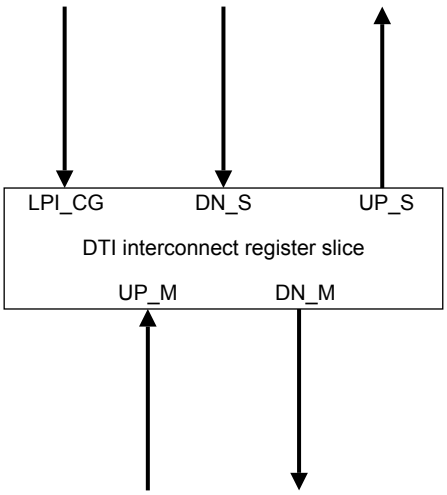

Figure 2-8 DTI interconnect register slice interfaces

The following table provides more information about the register slice interfaces.

Table 2-4 DTI interconnect register slice interfaces

Interface	Interface type	Protocol	Description
LPI_CG	Slave	Q-Channel	Clock-gating interface.
DN_S		AXI4-Stream	Slave downstream interface.
UP_S	Master		Slave upstream interface.
DN_M			Master downstream interface.
UP_M	Slave		Master upstream interface.

2.3 Operation

This section provides information about the operation of the MMU-600 features.

This section contains the following subsections:

2.3.1 DTI overview

In an MMU-600-based system, the AMBA DTI protocol defines the standard for communicating with a TCU.

The AMBA DTI protocol includes both:

- DTI-TBU protocol, for communication between a TBU and a TCU.
- DTI-ATS protocol, for communication between a PCIe Root Complex and a TCU.

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.

DTI masters and slaves communicate using defined DTI messages. The DTI protocol defines the following message groups:

- Connection and disconnection.
- Translation request.
- Invalidation and synchronization.
- Page request.
- Register access.

The DTI_TBU_CONDIS_REQ message initiates a TBU connection or disconnection handshake. The TBU uses this message to connect to the TCU. During connection, the TBU can specify the number of requested translation tokens.

The TBU uses the TOK_TRANS_REQ field to request translation tokens. The **max_tok_trans** signal defines the number of translation tokens that the TBU requests.

The TBU uses the TOK_INV_GNT field to grant invalidation tokens. The TBU grants only one invalidation token, and the TCU is only capable of issuing one invalidate message at a time.

A DTI master uses a DTI_TBU_CONDIS_REQ or a DTI_ATS_CONDIS_REQ message to initiate a connection handshake. If the master provides a **TID** value that is greater than the maximum supported **TID** that `TCUCFG_NUM_TBU` defines, the slave sends a Connect Deny message.

A translation request to the TCU where StreamID $\geq 2^{24}$ results in a fault and an SMMUv3 C_BAD_STREAMID event. If the TBU receives an invalidation request where StreamID $\geq 2^{24}$, any

comparisons with a StreamID value fail. No TLB entries are invalidated, but other effects that do not consider the supplied StreamID occur as normal.

Note

- The TBU never generates translation requests with StreamID $\geq 2^{24}$.
- The TCU never generates invalidation requests with StreamID $\geq 2^{24}$.

See the *Arm® AMBA® Distributed Translation Interface (DTI) Protocol Specification* for more information.

2.3.2 Performance Monitoring Unit

The MMU-600 includes a PMU for the TCU and a PMU for each TBU. The PMU events and counters indicate the runtime performance of the MMU-600.

The MMU-600 includes logic to gather various statistics on the operation of the MMU during runtime, using events and counters. These events, which the SMMUv3 architecture defines, provide useful information about the behavior of the MMU. You can use this information when debugging or profiling traffic.

SMMUv3 architectural performance events

Both the TCU and the TBU implement performance events that the SMMUv3 Performance Monitor extension defines.

The SMMU_PMCG_SMR0 register can filter some events so that only events with a particular StreamID are counted. This event filtering includes:

- Speculative transactions and translations.
- Transactions and translations that result in a terminated transaction or a translation fault.

The following table shows the architecturally defined MMU-600 TCU performance events.

Table 2-5 SMMUv3 performance events for the TCU

Event	Event ID	SMMU_PMCG_SMR0 filterable	Description
Clock cycle.	0x0	No	Counts clock cycles. Cycles where the clock is gated after a clock Q-Channel handshake are not counted.
Transaction.	0x1	Yes	Counts translation requests that originate from a DTI-TBU or DTI-ATS master.
TLB miss caused by incoming transaction or translation request.	0x2	Yes	Counts translation requests where the translation walks new translation table entries.
Configuration cache miss caused by transaction or translation request.	0x3	Yes	Counts translation requests where the translation walks new configuration table entries.
Translation table walk access.	0x4	Yes	Counts translation table walk accesses.
Configuration structure access.	0x5	Yes	Counts configuration table walk accesses.
PCIe ATS Translation Request received.	0x6	Yes	Counts translation requests that originate from a DTI-ATS master.

The following table shows the architecturally defined MMU-600 TBU performance events.

Table 2-6 SMMUv3 performance events for the TBU

Event	Event ID	SMMU_PMCG_SMR0 filterable	Description
Clock cycle.	0x0	No	Counts clock cycles. Cycles where the clock is gated after a clock Q-Channel handshake are not counted.
Transaction.	0x1	Yes	Counts transactions that are issued on the TBM interface.
TLB miss caused by incoming transaction or translation request.	0x2	Yes	Counts non-speculative translation requests that are issued to the TCU.
PCIe ATS Translation Request received.	0x7	Yes	Counts ATS-translated transactions that are issued on the TBM interface.

See the *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3.0 and version 3.1* for more information.

MMU-600 TCU events

The MMU-600 PMU can be configured to monitor a range of IMPLEMENTATION DEFINED TCU performance events.

The SMMU_PMCG_SMR0 register can filter some TCU performance events so that only events with a particular StreamID are counted. This event filtering includes:

- Speculative transactions and translations.
- Transactions and translations that result in a terminated transaction or a translation fault.

The following table shows the TCU performance events.

Table 2-7 MMU-600 TCU performance events

Event	Event ID	SMMU_PMCG_SMR0 filterable	Description
S1L0WC lookup	0x80	Yes	Counts translation requests that access the S1L0WC walk cache.
S1L0WC miss	0x81	Yes	Counts translation requests that access the S1L0WC walk cache and do not result in a hit.
S1L1WC lookup	0x82	Yes	Counts translation requests that access the S1L1WC walk cache.
S1L1WC miss	0x83	Yes	Counts translation requests that access the S1L1WC walk cache and do not result in a hit.
S1L2WC lookup	0x84	Yes	Counts translation requests that access the S1L2WC walk cache.
S1L2WC miss	0x85	Yes	Counts translation requests that access the S1L2WC walk cache and do not result in a hit.
S1L3WC lookup	0x86	Yes	Counts translation requests that access the S1L3WC walk cache.

Table 2-7 MMU-600 TCU performance events (continued)

Event	Event ID	SMMU_PMCG_SMR0 filterable	Description
S1L3WC miss	0x87	Yes	Counts translation requests that access the S1L3WC walk cache and do not result in a hit.
S2L0WC lookup	0x88	Yes	Counts translation requests that access the S2L0WC walk cache.
S2L0WC miss	0x89	Yes	Counts translation requests that access the S2L0WC walk cache and do not result in a hit.
S2L1WC lookup	0x8A	Yes	Counts translation requests that access the S2L1WC walk cache.
S2L1WC miss	0x8B	Yes	Counts translation requests that access the S2L1WC walk cache and do not result in a hit.
S2L2WC lookup	0x8C	Yes	Counts translation requests that access the S2L2WC walk cache.
S2L2WC miss	0x8D	Yes	Counts translation requests that access the S2L2WC walk cache and do not result in a hit.
S2L3WC lookup	0x8E	Yes	Counts translation requests that access the S2L3WC walk cache.
S2L3WC miss	0x8F	Yes	Counts translation requests that access the S2L3WC walk cache and do not result in a hit.
WC read	0x90	Yes	Counts reads from the walk cache RAMs, excluding reads that are caused by invalidation requests. **Note** A single walk cache lookup might result in multiple RAM reads. This behavior permits contiguous entries to be located.
Buffered translation	0x91	Yes	Counts translations written to the translation request buffer because all translation slots are full.
CC lookup	0x92	Yes	Counts lookups into the configuration cache.
CC read	0x93	Yes	Counts reads from the configuration cache RAMs, excluding reads that are caused by invalidation requests. **Note** A single cache lookup might result in multiple RAM reads. This behavior permits contiguous entries to be located.
CC miss	0x94	Yes	Counts lookups into the configuration cache that result in a miss.
Speculative translation	0xA0	Yes	Counts translation requests that are marked as speculative.
S1L0WC error	0xC0	No	RAS corrected error in S1L0 walk cache. This Secure event is visible only when the SMMU_PMCG_SCR.SO bit is set to 1.

Table 2-7 MMU-600 TCU performance events (continued)

Event	Event ID	SMMU_PMCG_SMR0 filterable	Description
S1L1WC error	0xC1	No	RAS corrected error in S1L1 walk cache. This Secure event is visible only when the SMMU_PMCG_SCR.SO bit is set to 1.
S1L2WC error	0xC2	No	RAS corrected error in S1L2 walk cache. This Secure event is visible only when the SMMU_PMCG_SCR.SO bit is set to 1.
S1L3WC error	0xC3	No	RAS corrected error in S1L3 walk cache. This Secure event is visible only when the SMMU_PMCG_SCR.SO bit is set to 1.
S2L0WC error	0xC4	No	RAS corrected error in S2L0 walk cache. This Secure event is visible only when the SMMU_PMCG_SCR.SO bit is set to 1.
S2L1WC error	0xC5	No	RAS corrected error in S2L1 walk cache. This Secure event is visible only when the SMMU_PMCG_SCR.SO bit is set to 1.
S2L2WC error	0xC6	No	RAS corrected error in S2L2 walk cache. This Secure event is visible only when the SMMU_PMCG_SCR.SO bit is set to 1.
S2L3WC error	0xC7	No	RAS corrected error in S2L3 walk cache. This Secure event is visible only when the SMMU_PMCG_SCR.SO bit is set to 1.
Configuration cache error	0xC8	No	RAS corrected error in configuration cache. This Secure event is visible only when the SMMU_PMCG_SCR.SO bit is set to 1.

Note

A single DTI translation request might correspond to multiple translation request events in either of the following circumstances:

- A translation results in a stall fault event and is restarted.
- If a translation results in a stall fault event because of the Event queue being full, the translation is retried when an Event queue slot becomes available.

MMU-600 TBU events

The MMU-600 PMU can be configured to monitor a range of IMPLEMENTATION DEFINED TBU performance events.

The SMMU_PMCG_SMR0 register can filter the TBU performance events so that only events with a particular StreamID are counted. This event filtering includes:

- Speculative transactions and translations.
- Transactions and translations that result in a terminated transaction or a translation fault.

The following table shows the TBU performance events.

Table 2-8 MMU-600 TBU performance events

Event	Event ID	SMMU_PMCG_SMR0 filterable	Description
Main TLB lookup	0x80	Yes	Counts Main TLB lookups.
Main TLB miss	0x81	Yes	Counts translation requests that miss in the Main TLB.
Main TLB read	0x82	Yes	Counts once per access to the Main TLB RAMs, excluding reads that invalidation requests cause. **Note** A transaction might access the Main TLB multiple times to look for different page sizes.
Micro TLB lookup	0x83	Yes	Counts micro TLB lookups.
Micro TLB miss	0x84	Yes	Counts translation requests that miss in the micro TLB.
Slots full	0x85	No	Counts once per cycle when all slots are occupied and not ready to issue transactions downstream. This Secure event is visible only when the SMMU_PMCG_SCR.SO bit is set to 1.
Out of translation tokens	0x86	No	Counts once per cycle when a translation request cannot be issued because all translation tokens are in use. This Secure event is visible only when the SMMU_PMCG_SCR.SO bit is set to 1.
Write data buffer full	0x87	No	Counts once per cycle when a transaction is blocked because the write data buffer is full. This Secure event is visible only when the SMMU_PMCG_SCR.SO bit is set to 1.
Translation request	0x88	Yes	Counts translation requests, including both speculative and non-speculative requests.
Write data uses write data buffer	0x89	Yes	Counts transactions with write data that is stored in the write data buffer.
Write data bypasses write data buffer	0x8A	Yes	Counts transactions with write data that bypasses the write data buffer.
MakeInvalid downgrade	0x8B	Yes	Counts when either: • A MakeInvalid transaction on the TBS interface is output as CleanInvalid on the TBM interface. • A ReadOnceMakeInvalid transaction on the TBS interface is output as ReadOnceCleanInvalid on the TBM interface.

Table 2-8 MMU-600 TBU performance events (continued)

Event	Event ID	SMMU_PMCG_SMR0 filterable	Description
Stash fail	0x8C	Yes	Counts when either. • A WriteUniquePtlStash or WriteUniqueFullStash transaction on TBS is output as a WriteNoSnoop or WriteUnique transaction on the TBM interface. • A StashOnceShared or StashOnceUnique transaction on the TBS interface has a valid translation, but is terminated in the TBU. **Note** A StashOnceShared or StashOnceUnique transaction that is terminated because of a StreamDisable or GlobalDisable translation response does not cause this event to count.
Main TLB error	0xC0	No	RAS corrected error in Main TLB. This Secure event is visible only when the SMMU_PMCG_SCR.SO bit is set to 1.

SMMUv3 PMU register architectural options

The SMMUv3 architecture defines the *Performance Monitor Counter Group* (PMCG) configuration register, SMMU_PMCG_CFGR. An MMU-600 implementation assumes fixed values for SMMU_PMCG_CFGR, and these values define behavioral aspects of the implementation.

The following table shows the SMMU_PMCG_CFGR register options that the MMU-600 TCU and TBU use.

Table 2-9 MMU-600 SMMU_PMCG_CFGR register architectural options

Field	Default value	Description for default value
SID_FILTER_TYPE	1	A single StreamID filter applies to all PMCG counters.
CAPTURE	1	Capture of counter values into SVRn registers is supported.
MSI	0	The counter group does not support *Message Signaled Interrupts* (MSIs).
RELOC_CTRS	1	The PMCG registers are relocated to page 1 of the PMU address map.
SIZE	0x31	The counter group implements 32-bit counters.
NCTR	0x3	The counter group includes 4 counters.

Related information

3.3 MMU-600 memory map on page 3-69

PMU snapshot interface

The *Performance Monitoring Unit* (PMU) snapshot interface is included on the TCU and on each TBU. You can use this asynchronous interface to initiate a PMU snapshot. A simultaneous snapshot of each counter register is created and copied to the respective SMMU_PMCG_SVRn register.

The PMU snapshot sequence is a 4-phase handshake. Both **pmusnapshot_req** and **pmusnapshot_ack** are LOW after reset. A snapshot occurs on the rising edge of **pmusnapshot_req**, and is equivalent to writing the value 1 to SMMU_PMCG_CAPR.CAPTURE.

The **pmusnapshot_req** signal is sampled using synchronizing registers. A register drives **pmusnapshot_ack** so that the connected component can sample the signal asynchronously.

Related information

2.3.5 Reliability, Availability, and Serviceability on page 2-47
A.5 TCU PMU snapshot interface signals on page Appx-A-104
A.14 TBU PMU snapshot interface signals on page Appx-A-119

2.3.3 ACE protection support

ACE protection support provides a TBU configuration that protects devices that implement a fully coherent ACE interface. The configuration aligns with the restrictions that ACE5 defines for ACE usage of the Untranslated_Transactions extension.

A fully coherent master receives snoop transactions from the interconnect. Such snoop transactions travel in the opposite direction from regular transactions, and backward address translation is not possible. ACE protection configuration in the TBU however provides a protection check for such masters, by requiring that the input address and output address are the same.

ACE protection supports at least 255 outstanding snoop transactions on the snoop address channel before back-pressure is applied to the channel.

The following table shows the transactions that belong to each of the main ACE transaction groups.

Table 2-10 ACE transaction groups

Group	Reads	Writes
Non-shareable and I/O coherent transactions	ReadNoSnoop ReadOnce	WriteNoSnoop WriteUnique WriteLineUnique Non-shareable WriteBack Non-shareable WriteClean Non-shareable WriteEvict
Fully coherent transactions	ReadClean ReadNotSharedDirty ReadShared ReadUnique CleanUnique MakeUnique	Shareable WriteBack Shareable WriteClean Shareable WriteEvict Evict

ACE protection places restrictions on an upstream master. These restrictions ensure that transactions from one transaction group cannot manipulate data that is read into a cache by transactions from the other transaction group. The groups of transactions can therefore behave in different ways:

- Non-shareable and IO-coherent transactions are translated using the normal rules of SMMU translation. The only exception is that changes in shareability are not permitted, because this would enable WriteNoSnoop transactions to be changed into WriteUnique transactions. Such a change could violate the deadlock avoidance rules regarding outstanding WriteBack or WriteClean transactions.
- Fully coherent transactions are only supported for translations that are suitable for ACE protection. This behavior prevents transactions from being modified or access permissions being limited during

translation. Writes and snoops are only permitted for lines already read, so the protection check is only required for reads.

A TBU that is configured for ACE protection only supports stage 2 translation, and there cannot therefore be any RAZ/WI translation results. Therefore, anything other than translations that are supported in ACE protection can result in a transaction fault.

ACE protection does not support PCIe *Address Translation Services* (ATS). When an ACE TBU configuration is used, any transaction where **armmuatst** = 1 or **awmmuatst** = 1 is terminated with an SLVERR response.

Effect of ACE protection on transaction behavior

The scope of how ACE transactions are supported varies depending on whether ACE protection support is enabled.

The following table shows which transactions are supported in different circumstances, where the stated behavior is described after the table.

Table 2-11 ACE protection support for transactions

Transaction	Behavior for ACE-Lite TBU configurations	Behavior for ACE TBU configurations
ReadNoSnoop WriteNoSnoop ReadOnce WriteUnique WriteLineUnique	Translate	Translate-NoSH
ReadClean ReadNotSharedDirty ReadShared ReadUnique CleanUnique MakeUnique	Illegal	Prot-RWX-only
Non-shareable WriteBack Non-shareable WriteClean Non-shareable WriteEvict	Illegal	Abort
Shareable WriteBack Shareable WriteClean Shareable WriteEvict Evict	Illegal	Pass-through
CleanShared CleanSharedPersist CleanInvalid MakeInvalid	Behavior depends on **cmo_disable** setting: **0** Translate. **1** Abort.	Abort

Table 2-11 ACE protection support for transactions (continued)

Transaction	Behavior for ACE-Lite TBU configurations	Behavior for ACE TBU configurations
ReadOnceCleanInvalid ReadOnceMakeInvalid WriteUniquePtlStash WriteUniqueFullStash StashOnceShared StashOnceUnique StashTranslation	Translate	Illegal
DVM Complete	Illegal	Pass-through
DVM Message	Illegal	Abort

The behaviors that the table describes have the following meanings:

Translate
The transaction is translated as normal.

Abort
The transaction is terminated with an SLVERR response.

Illegal
The transaction is defined as an AMBA protocol error on this type of interface.

Pass-through
The transaction propagates through the TBU without attribute checks or modification. The table-based hardware attributes and STE *implementation defined* auxiliary attributes **AxUSER** fields are 0.

Translate-NoSH
The transaction is translated, but terminated with an SLVERR response when any of the following apply:
- Stage 1 translation is enabled.
- The STE.MTCFG field is set to not use the incoming memory type.
- The STE.SHCFG field is set to not use the incoming shareability attribute or STE.MTCFG
- For translations where stage 2 translation is enabled, the SH field of the stage 2 translation table entry is not Non-shareable.

Prot-RWX-only
The transaction is translated, but terminated with an SLVERR response when any of the following apply:
- Any of STE.NSCFG, STE.PRIVCFG, STE.INSTCFG, or STE.MTCFG are set to not use the incoming attribute, when those fields are not otherwise ignored.
- For translations where Stage 2 translation is enabled, the MemAttr field of the Stage 2 translation table entry is not Inner and Outer Write-Back Cacheable.
- For translations where Stage 2 translation is enabled, the output address of the Stage 2 translation table entry is not the same as the input address.
- For translations where Stage 2 translation is enabled, the XN field of the Stage 2 translation table entry is not `0b00`.
- For translations where Stage 2 translation is enabled, the S2AP field of the Stage 2 translation table entry is not `0b11`.
- For Secure translations where Stage 2 translation is enabled, SMMU_S_CR0.SIF=1.

For transactions where the ACE protection behavior is Prot-RWX-only or Pass-through, the shareability and attributes are not modified. For these transactions:

- The master and slave attribute normalization rules are not used.
- The **AxDOMAIN**, **AxCACHE**, and **AxLOCK** values that are output from the TBM interface are the same as the values that are input to the TBS interface.
- On the TBM interface, the outer cacheable bit in **AxUSER** is 1 if **AxCACHE** is a cacheable type, that is, if **AxCACHE[3:2]** != `0b00`, and 0 otherwise.

Stalling faults

Ensure that stalling faults are not enabled for ACE TBUs that might enter fully coherent mode.

Stalling faults stall WriteNoSnoop and WriteUnique transactions, possibly leading to stalled Write-Back transactions, stalled snoop responses, and system deadlocks. The TBU does not know about stalling faults and therefore cannot prevent such circumstances. When a *Stream Table Entry* (STE) is used for a fully coherent master, the SMMUv3 driver must therefore:

- Set the STE.S1STALLD bit.
- Clear the STE.S2S bit.

Removing permission to access a translation table

When a translation table entry is modified and therefore invalidated, it is important to ensure that a master cannot read any modified cache lines into its coherent cache.

You can prevent a master from reading invalid cache lines by removing permission to access the affected translation table as follows:

Procedure

1. Change the Stage 2 translation tables to remove permission to access.
2. Invalidate the translation tables in the SMMU. After invalidating a translation table in the SMMU, a master cannot read the affected cache lines. However, those that the master holds in cache might still be invalid anyway.
3. Issue Clean and Invalidate to *Point of Coherency* (PoC) operations to the affected cache lines. This removes the cache lines from the GPU coherent cache.
4. Zero the data in the translation table.
5. Issue Clean to PoC operations to the affected cache lines. This step is necessary to ensure that the zeroed data is visible to non-coherent masters.
6. Change the translation tables to provide access to the new user of the table.

Related information

AXI5 support on page 2-59
Upstream ACE master restrictions on page 2-60
Avoiding deadlock when using fully coherent ACE masters on page 2-60

2.3.4 TBU direct indexing and MTLB partitioning

TBU direct indexing can help your system to meet real-time translation requirements by enabling the MMU-600 to manage *Main TLB* (MTLB) entries externally to the TBU.

Direct indexing enables real-time translation requirements to be met, as follows:

- Prefetched entries can be guaranteed not to be overwritten by different streams.
- The MTLB can be partitioned into different sets of entries that are used by different streams.

If you configure your system to not use direct indexing, you can select MTLB partitioning. MTLB partitioning has similar behavior, but only the most significant TLB index bits are provided, and the other bits are generated internally.

Direct indexing is enabled for a TBU when `TBUCFG_DIRECT_IDX` = 1.

When TBUCFG_DIRECT_IDX = 1, or when an MTLB is partitioned, **aruser_m** and **awuser_m** have more bits than the corresponding signals on the TBS interface. The following table shows the **aruser_m** and **awuser_m** extended bits.

Note

The table lists the extended bits in the order MSB first.

Table 2-12 Extended aruser_m and awuser_m bits for MTLB partitioning

Field name	Width	Description
mtlbidx	When direct indexing is enabled, the width of this field is log2(TBUCFG_MTLB_DEPTH) - 2. When direct indexing is not enabled, the width of this field is 0.	MTLB index.
mtlbway	When direct indexing is enabled, the width of this field is 2. When direct indexing is not enabled, the width of this field is 0.	MTLB way.
mtlbpart	log2(TBUCFG_MTLB_PARTS)	MTLB partition.
-	TBUCFG_AWUSER_WIDTH for **awuser_m**. TBUCFG_ARUSER_WIDTH for **aruser_m**.	Regular **AxUSER** signals.

If an MTLB is partitioned:

- The MTLB size is multiplied by TBUCFG_MTLB_PARTS.
- The mtlbpart field defines the log2(TBUCFG_MTLB_PARTS) most significant index bits.

When direct indexing is enabled for a TBU:

- Lookups and updates to the MTLB use the mtlbidx field.
- Updates to the MTLB use the way that mtlbway specifies.
- Lookups to the MTLB operate on all ways simultaneously.

To maintain system performance, Arm recommends that DVM invalidation is disabled on TBUs on which direct indexing is enabled. Disable DVM invalidation by setting the appropriate TCU_NODE_CTRLn.DIS_DVM bit.

2.3.5 Reliability, Availability, and Serviceability

Reliability, Serviceability, and Availability (RAS) features enable cache corruption to be detected and corrected, optionally generating interrupts into the system. All MMU-600 RAM-based caches support RAS error detection and correction.

The RAS Extension registers permit software to monitor the following caches for errors:

- TBU *Main TLB* (MTLB).
- TCU configuration cache.
- TCU translation table walk cache.

Within a coherent system, these caches are always clean, and there is no requirement to correct data on these caches. Any incorrect data is discarded and refetched. From an RAS standpoint, discarding and refetching counts as a corrected error.

See the *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3.0 and version 3.1* for more information.

Related information

3.8.1 TCU_ERRFR on page 3-84
3.8.2 TCU_ERRCTLR on page 3-84

3.8.3 TCU_ERRSTATUS on page 3-85
3.12.1 TBU_ERRFR on page 3-93
3.12.2 TBU_ERRCTLR on page 3-93
3.12.3 TBU_ERRSTATUS on page 3-94

2.3.6 Quality of Service

You can program the TCU with a priority level for each TBU. The priority level is applied to every translation from that TBU.

The TCU uses this priority level to:

- Arbitrate between translations that are waiting in the translation request buffer when translation manager slots become available.
- Arbitrate between translation manager slots when they access the caches and perform configuration table walks and translation table walks.
- Determine the AXI **AxQOS** value for translation table walks and configuration table walks that the TCU issues on the QTW/DVM interface.

The arbiters contain starvation avoidance mechanisms to prevent transactions from being stalled indefinitely.

The TBU does not implement any prioritization between transactions. Arm recommends that bus masters with different QoS requirements use separate TBUs for translation.

Related information

3.7.2 TCU_QOS on page 3-77
3.7.6 TCU_NODE_CTRLn on page 3-81

2.3.7 *Distributed Virtual Memory* (DVM) messages

The QTW/DVM interface supports DVM messages. The MMU-600 supports DVMv8.1.

The interface supports DVM transactions of message types TLB Invalidate and Synchronization. The interface accepts all other DVM transaction message types, and sends a snoop response, but otherwise ignores such transactions.

Tie the **sup_btm** input signal HIGH when Broadcast TLB Maintenance is supported.

You can use SMMU_CR2 and SMMU_S_CR2 to control how the QTW/DVM interface responds to TLB Invalidate operations:

- If SMMU_CR2.PTM = 1, the interface ignores Non-secure TLB Invalidate operations.
- If SMMU_S_CR2.PTM = 1, the interface ignores Secure TLB Invalidate operations.

Note

Although TLB Invalidate operations have no effect on the QTW/DVM interface when PTM = 1, the interface still returns the appropriate response.

The QTW/DVM interface might receive DVM Sync transactions without receiving a DVM TLB Invalidate transaction, or when the PTM bits have masked a TLB Invalidate. If no DVM TLB Invalidate operations have occurred since the most recent DVM Sync transaction, subsequent DVM Sync transactions result in an immediate DVM Complete transaction. This behavior ensures that the TCU does not affect system DVM performance unless TLB Invalidate operations are performed.

The DTI interface allocates the access permissions and shareability of DVM Complete transactions as follows:

- **ARPROT** = `0b000`, indicating Unprivileged, Secure, Data access.
- **ARDOMAIN** = `0b01`, indicating Inner Shareable.

For a DVM Operation or DVM Sync request on the **AC** channel, the snoop response signal **CRRESP[4:0]** is always set to `0b00000`.

2.3.8 TCU transaction handling

The transaction width, burst length, and transfer size that the TCU supports depend on the transaction type.

The following table shows the TCU support for read transactions.

Table 2-13 TCU support for read transactions

Transaction type	Transaction width, bits	ARID[n:1]	ARID[0]
Stage 1 Stream table lookup	64	`TCUCFG_PTW_SLOTS`	1
Stream table lookup	256	`TCUCFG_PTW_SLOTS`	1
Translation table lookup	64	`TCUCFG_PTW_SLOTS`	1
Command queue read	128	All 0	0
DVM Complete	-	All 1	0

DVM Complete transactions are always one beat of full data width.

Command queue reads and DVM complete transactions are independent of translation slots. Therefore, the maximum number of read transactions that the TCU can issue at any time is `TCUCFG_PTW_SLOTS` + 2.

The following table shows the TCU support for write transactions.

Table 2-14 TCU support for write transactions

Transaction type	Transaction width, bits	AWID
Event queue write	256	0
PRI queue write	128	0
Message Signaled Interrupt (MSI)	32	0

Only one write transaction can be outstanding at a time.

All read and write transactions are aligned to the transaction size.

2.3.9 TCU prefetch

TCU prefetch enables the TCU to prefetch translations on a per-context basis, improving translation performance for real-time masters that access memory linearly. Software can request a TCU prefetch of the next translation table to be accessed, when it is required.

Prefetched translations are placed in the TCU walk caches. When the TBU requires the prefetched translation, it is passed from the TCU to the TBU.

Bits [121:120] of the STE are IMPLEMENTATION DEFINED in SMMUv3, and have the following meanings for the MMU-600:

`0b00` Prefetch disabled.

`0b01` Reserved.

`0b10` Prefetch forwards.

`0b11` Prefetch backwards.

2.3.10 Error responses

AMBA defines external AXI slave error, SLVERR, and external AXI decode error, DECERR. The MMU-600 error response behavior depends on the interface.

The TCU QTW/DVM interface treats SLVERR and DECERR identically, as an abort.

When terminating a transaction, the TBS interface generates a SLVERR response.

If the TBU TBM interface receives a SLVERR or DECERR response to a downstream transaction, it propagates the same abort type to the TBS interface.

2.3.11 Conversion between ACE-Lite and ARMv8 attributes

The SMMUv3 architecture defines attributes in terms of the ARMv8 architecture. The MMU-600 components are therefore required to perform conversion between ACE-Lite and ARMv8 attributes.

The TBU must convert:

- ACE-Lite attributes to ARMv8 attributes when it receives transactions on the *Transaction Slave* (TBS) interface.
- ARMv8 attributes to ACE-Lite attributes when it outputs transactions on the *Transaction Master* (TBM) interface.

The TCU must convert ARMv8 attributes to ACE-Lite attributes when it outputs transactions on the QTW/DVM interface.

Slave interface memory type attribute handling

The memory attributes that apply to the TBS interface are contained in the **AxCACHE** and **AxDOMAIN** signals.

The following table shows the ACE-Lite to ARMv8 attribute conversions that the TBU TBS interface performs.

Table 2-15 MMU-600 ACE-Lite to ARMv8 memory attribute conversions

AxCACHE attribute	AxDOMAIN attribute	ARMv8 memory attribute	ARMv8 shareability
Device Non-bufferable	System	Device-nGnRnE	Outer Shareable
Device Bufferable	System	Device-nGnRE	Outer Shareable
Normal Non-cacheable Bufferable Normal Non-cacheable Non-bufferable Write-Through No Allocate Write-Through Read-allocate Write-Through Write-Allocate Write-Through Read and Write-Allocate	Any	Normal Inner Non-cacheable Outer Non-cacheable	Outer Shareable
Write-Back No Allocate Write-Back Read-Allocate Write-Back Write-Allocate Write-Back Read Allocate Write-Allocate	Non-shareable Inner Shareable Outer Shareable	Normal Inner Write-Back Outer Write-Back	Non-shareable Non-shareable Outer Shareable

Note

- WriteBack transactions are always treated as non-transient.
- The ARMv8-A Read-Allocate and Write-Allocate hints are the same as the hints that the **AxCACHE** Write-Back type provides.
- The TBU TBS interface converts instruction writes into data writes. That is, it treats **AWPROT[2]** as 0.

Master interface memory type attribute handling

The memory attributes that apply to the TBM and the QTW/DVM interfaces are contained in the **AxCACHE** and **AxDOMAIN** signals.

In addition, the TBU TBM interface can use the **AxLOCK** signal to indicate an Exclusive access. The QTW/DVM interface does not use the **AxLOCK** signal.

On the TBU TBM interface, a bit on **AxUSER** indicates whether the memory type before the conversion is Outer Cacheable.

The following table shows the ARMv8 to ACE-Lite attribute conversions that the master interfaces perform.

Table 2-16 MMU-600 ARMv8 to ACE-Lite memory attribute conversions

ARMv8 memory attribute	AxCACHE attribute	AxDOMAIN attribute	AxLOCK attribute	AxUSER Outer Cacheable
Device-nGnRnE	Device Non-bufferable.	System.	As *Transaction Slave* (TBS) **AxLOCK** value	0
Device-GRE Device-nGRE Device-nGRE	Device Bufferable.	System.	As TBS **AxLOCK** value	0
Normal Inner Non-cacheable Outer Non-cacheable Normal Inner Write-Through Outer Non-cacheable Normal Inner Write-Back Outer Non-cacheable	Normal Non-cacheable Bufferable.	System.	As TBS **AxLOCK** value	0

Table 2-16 MMU-600 ARMv8 to ACE-Lite memory attribute conversions (continued)

ARMv8 memory attribute	AxCACHE attribute	AxDOMAIN attribute	AxLOCK attribute	AxUSER Outer Cacheable
Normal Inner Non-cacheable Outer Write-Through Normal Inner Write-Through Outer Write-Through Normal Inner Write-Back Outer Write-Through Normal Inner Non-cacheable Outer Write-Back Normal Inner Write-Through Outer Write-Back	Normal Non-cacheable Bufferable.	System.	As TBS **AxLOCK** value	1
Normal Inner Write-Back Outer Write-Back	Write-Back No Allocate Write-Back Read-Allocate. Write-Back Write-Allocate. Write-Back Read and Write-Allocate.	If **AxBURST** == FIXED, Non-shareable. If **AxBURST** != FIXED, the attribute reflects the ARMv8 shareability: • Non-shareable. • Inner Shareable. • Outer Shareable.	0	1

2.3.12 AXI USER bits defined by the MMU-600 TBU

The TBU TBM interface **AxUSER** signals, **aruser_m** and **awuser_m**, have 13 bits more than the corresponding signals on the TBS interface. These extra bits are output in higher-order bits of **aruser_m** and **awuser_m**.

The following table shows the MMU-600-defined **aruser_m** and **awuser_m** bits, where *w* represents the AXI USER bus width that `TBUCFG_AxUSER_WIDTH` defines.

Table 2-17 MMU-600 defined aruser_m and awuser_m bits

Bit position	Value
[*w*+12]	Outer Cacheable.
[*w*+11:n+8]	The *Stream Table Entry* (STE) defines the attributes.
[*w*+7:n+4]	The IMPLEMENTATION DEFINED stage 2 hardware attributes.
[*w*+3:n]	The IMPLEMENTATION DEFINED stage 1 hardware attributes.

Bits [119:116] of the STE are IMPLEMENTATION DEFINED in SMMUv3. When the TCU sends a DTI translation response message to a TBU, it outputs these bits in the

DTI_TBU_TRANS_RESP.CTXTATTR field. The MMU-600 TBU outputs these bits as STE-defined attributes.

The TCU DTI_TBU_TRANS_RESP response also includes S1HWATTR[3:0] and S2HWATTR[3:0] fields. These fields provide the IMPLEMENTATION DEFINED hardware attributes for each stage of translation. The TBU reports these fields using awuser_m and aruser_m.

The S1HWATTR and S2HWATTR fields are calculated as follows:

S1HWATTR

S1HWATTR[*n*] is equal to bit[*n*+59] of the stage 1 translation table final-level descriptor when both of the following conditions apply:

- SMMUv3 permits the bit to have an IMPLEMENTATION DEFINED hardware use.
- SMMUv3 does not permit bit[*n*+59] of the stage 2 translation table final-level descriptor to have an IMPLEMENTATION DEFINED hardware use.

Otherwise, S1HWATTR[*n*] = 0.

S2HWATTR

S2HWATTR[*n*] is equal to bit[*n*+59] of the stage 2 translation table final-level descriptor when SMMUv3 permits that bit to have an IMPLEMENTATION DEFINED hardware use. Otherwise, S2HWATTR[*n*] = 0.

Arm recommends that systems always use the value of S1HWATTR[n] | S2HWATTR[n], that is:

- The value of the corresponding stage 2 final-level descriptor bit, if it is enabled for hardware use and stage 2 translation is enabled.
- The value of the corresponding stage 1 final-level descriptor bit, if it is enabled for hardware use and stage 1 translation is enabled.
- Otherwise, 0.

Related information

Master interface memory type attribute handling on page 2-51

2.4 Constraints and limitations of use

Certain usage constraints and limitations apply to the MMU-600.

Unless otherwise specified:

- An IMPLEMENTATION DEFINED field in a structure that the MMU-600 generates is 0.
- An IMPLEMENTATION DEFINED field in a structure that the MMU-600 reads is ignored.

This section contains the following subsections:

2.4.1 SMMUv3 support

The MMU-600 does not implement, or require, certain SMMUv3 functionality.

The SMMUv3 architectural registers include a set of ID registers that indicate the SMMUv3 features that the MMU-600 implements. The following table shows the SMMUv3 ID register values that the MMU-600 uses.

——— **Note** ———

The values in this table are not configurable except for values that are specified in **bold**.

———————————

Table 2-18 MMU-600 SMMUv3 ID register architectural options

Register	Field	Value	Description for value
SMMU_IDR0	S2P	1	Stage 2 translations are supported.
	S1P	1	Stage 1 translations are supported.
	TTF	`0b11`	Both AArch32 Long-descriptor and AArch64 translation tables are supported.
	COHACC	**sup_cohacc**	Coherent access to translations, structure, and queues is supported.
	BTM	**sup_btm**	Broadcast TLB maintenance is supported.
	HTTU[1:0]	`0b00`	Updates of the Dirty state and Access flag are not supported.
	DORMHINT	0	Dormant hint is not supported.
	HYP	1	Hypervisor stage 1 context is supported.
	ATS	1	PCIe Root Complex ATS is supported.
	NS1ATS	1	Stage 1-only ATS is not supported.
	ASID16	1	16-bit ASID is supported.
	MSI	1	*Message Signaled Interrupts* (MSIs) are supported.
	SEV	**sup_sev**	SMMU and system support for the generation of events.
	ATOS	0	Address translation operations are not supported.
	PRI	1	PCIe *Page Request Interface* (PRI) is supported.
	VMW	1	VMID wildcard-matching is supported for TLB invalidation.
	VMID16	1	16-bit VMIDs are supported.
	CD2L	1	2-level *Context Descriptor* (CD) tables are supported.
	VATOS	0	Virtual ATOS page interface is not supported.
	TTENDIAN	`0b00`	Mixed-endian translation walks are supported.
	STALL_MODEL	{0, SMMU_S_CR0.NSSTALLD}	Stall model and Terminate model are both supported, unless the Secure world disables Non-secure stalling.
	TERM_MODEL	0	Terminated transactions can terminate with either RAZ/WI behavior or abort.
	ST_LEVEL	`0b01`	2-level Stream tables are supported.
SMMU_IDR1	SIDSIZE	`0b11000`	24-bit stream IDs are supported.
	SSIDSIZE	`0b10100`	20-bit substream IDs are supported.
	PRIQS	`0b10011`	2^{19} PRI queue entries are supported.
	EVENTQS	`0b10011`	2^{19} Event queue entries are supported.
	CMDQS	`0b10011`	2^{19} Command queue entries are supported.
	ATTR_PERMS_OVR	1	Incoming permission attributes can be overridden.
	ATTR_TYPES_OVR	1	Incoming memory attributes can be overridden.
	REL	0	Base addresses are not fixed.
	QUEUES_PRESET	0	The queue base addresses are not fixed.
	TABLES_PRESET	0	The table base addresses are not fixed.

Table 2-18 MMU-600 SMMUv3 ID register architectural options (continued)

Register	Field	Value	Description for value
SMMU_IDR2	BA_VATOS	0	No VATOS page is present.
SMMU_IDR3	HAD	1	Hierarchical Attribute Disable is supported.
	PBHA	1	Page-Based Hardware Attributes are supported.
	XNX	1	EL0/EL1 execute control distinction at stage 2 is supported for both AArch64 and AArch32 stage 2 translation tables.
	PPS	1	If the request has a *Process Address Space ID* (PASID), the PASID is included in PRI queue overflow auto-generated responses. The STE.PPAR field is not checked and is treated as 1.
SMMU_IDR4	IMPDEF	0	No IMPLEMENTATION DEFINED features apply.
SMMU_IDR5	OAS	**sup_oas**	The size of the physical address that is output from the SMMU.
	GRAN4K	1	4KB translation granule is supported.
	GRAN16K	1	16KB translation granule is supported.
	GRAN64K	1	64KB translation granule is supported.
	VAX	`0b00`	Virtual addresses of 48 bits per CD.TTBx are supported.
	STALL_MAX	`TCUCFG_XLATE_SLOTS`	Maximum number of outstanding stalled transactions that the SMMU supports.
SMMU_IIDR	Implementer	`0x43B`	Arm implementation.
	Revision	MAX(`0x2`, **ecorevnum**)	Minor revision is p2. **Note** **ecorevnum** is not configurable.
	Variant	0	Product variant, or major revision is r0.
	ProductID	`0x483`	Arm ID.
SMMU_AIDR	ArchMinorRev	`0b0001`	Architecture minor revision is SMMUv3.1.
	ArchMajorRev	`0b0000`	Architecture major revision is SMMUv3.
SMMU_S_IDR0	MSI	1	Secure MSIs are supported.
	STALL_MODEL	`0b00`	Stall model and Terminate model are both supported.
SMMU_S_IDR1	S_SIDSIZE	`0b11000`	24-bit Secure stream IDs are supported.
	SECURE_IMPL	1	Security implemented.
SMMU_S_IDR3	SAMS	1	Secure *Address Translation Services* (ATS) maintenance is not implemented.

In an MMU-600-based system, the SFM_ERR global error cannot occur, because *Service Failure Mode* (SFM) is not required.

The MMU-600 accepts but does not act on the following SMMUv3 Prefetch commands:

CMD_PREFETCH_CONFIG
Prefetch configuration. This command prefetches the required configuration for a StreamID.

CMD_PREFETCH_ADDR
Prefetch address. This command prefetches configuration and TLB entries for an address range.

The MMU-600 does not generate any of the following SMMUv3 events, because they are not required:

F_UUT
Unsupported Upstream Transaction.

F_TLB_CONFLICT
TLB conflict.

F_CFG_CONFLICT
Configuration cache conflict.

E_PAGE_REQUEST
Speculative page request hint.

IMPDEF_EVENTn
IMPLEMENTATION DEFINED event allocation.

Note

F_TLB_CONFLICT and F_CFG_CONFLICT are not required because the MMU-600 caches include logic to ensure that only one entry can match at a time. If multiple cache entries match a transaction or translation request, only one entry is used and the others are ignored.

The MMU-600 never merges events. The STE.MEV field is ignored.

The TBU ignores the STE.ALLOCCFG field that the TCU communicates to the TBU in the ALLOCCFG field of the DTI_TBU_TRANS_RESP message.

The TCU **sup_oas[2:0]** signal must not be set to `0b110`. If this value is used, the TCU treats it as `0b101`, that is, 48 bits. The TBU supports a 48-bit PA size. The MMU-600 TBU and TCU cannot be used with other components that implement DTI and are configured for a 52-bit PA size.

Related information
3.2 SMMU architectural registers on page 3-64

2.4.2 AMBA support

Certain behavior applies to how the MMU-600 implements its ACE-Lite interfaces.

TBU support for ACE-Lite transactions

The MMU-600 TBU supports many ACE-Lite transaction types, and handles these transactions in certain ways. Typically, when propagating downstream transactions on the TBU TBM interface, the MMU-600 uses the same transaction type that the upstream master presents to the TBU TBS interface.

If the shareability domain of a downstream WriteLineUnique transaction is not Inner Shareable or Outer Shareable, the MMU-600 outputs the transaction as WriteNoSnoop. That is, **AWSNOOP** = `0b0000`. The **AWDOMAIN** signal indicates the shareability domain of write transactions.

Transactions that can result in a translation fault

In an MMU-600 system, some transactions can result in a translation fault, and certain behavior is associated with such transactions.

The MMU-600 treats the following transactions as ordinary reads when calculating translation faults:

- CleanShared.
- CleanInvalid.
- MakeInvalid.
- CleanSharedPersist.

- ReadOnceMakeInvalid.
- ReadOnceCleanInvalid.

Therefore, these transactions might require either read permission or execute permission at the appropriate privilege level.

The MMU-600 treats the following transactions as ordinary writes when calculating translation faults:
- WriteUniquePtlStash.
- WriteUniqueFullStash.

Therefore, these transactions require write permission at the appropriate privilege level.

CleanShared, CleanInvalid, MakeInvalid, and CleanSharedPersist transactions do not have a memory type. The input transaction and output transaction memory type and allocation hints are ignored and replaced by Normal, Inner Write-Back, Outer Write-Back, Read Allocate, Write Allocate. This behavior means that the **ARDOMAIN** output on the TBM interface is never System Shareable for these transactions, because they are never Non-cacheable or Device.

The MMU-600 treats transactions that pass the translation fault check as follows:

MakeInvalid transactions
The MMU-600 converts MakeInvalid transactions to CleanInvalid transactions, unless the translation also grants write permission and *Destructive Read Enable* (DRE) permission.

ReadOnceMakeInvalid and ReadOnceCleanInvalid transactions
The MMU-600 outputs ReadOnceMakeInvalid transactions as ReadOnceCleanInvalid transactions, unless the translation also granted write permission and DRE permission.
If the final transaction attributes on the TBU TBM interface are not Inner Shareable Write-Back or Outer Shareable Write-Back, the MMU-600 converts ReadOnceMakeInvalid and ReadOnceCleanInvalid transactions into ordinary reads.

WriteUniquePtlStash and WriteUniqueFullStash transactions
If they pass the translation fault check, the MMU-600 converts WriteUniquePtlStash and WriteUniqueFullStash transactions to ordinary write transactions if either:
- The translation did not grant *Directed Cache Prefetch* (DCP) permission.
- The final transaction attributes on the TBU TBM interface are not Inner Shareable or Outer Shareable Write-Back.

If such a conversion occurs, **AWSTASH*** is driven as 0.

Transactions that cannot result in a translation fault

In an MMU-600 system, certain transactions cannot result in a translation fault, and certain behavior is associated with such transactions.

The following transactions never result in a translation fault:

- StashOnceShared.
- StashOnceUnique.
- StashTranslation.

If any of these transactions require a translation request to the TCU, the MMU-600 issues a speculative translation request on the DTI interconnect. StashOnceShared and StashOnceUnique transactions are terminated in the TBU, with a **BRESP** value of OKAY, when any of the following cases apply:
- The translation did not grant *Directed Cache Prefetch* (DCP) permission.
- The final transaction attributes on the TBM interface are not Inner Shareable or Outer Shareable Write-Back.
- The translation did not grant any of read, write, or execute permission at the appropriate privilege level.

Note

Only one of these permissions is required for the stash transaction to be permitted.

Note

A **BRESP** value of OKAY indicates transaction success. The MMU-600 always generates this value when a StashOnceShared or a StashOnceUnique transaction is terminated in the TBU. This behavior applies even when a StreamDisable or GlobalDisable translation response causes the transaction to be terminated.

The MMU-600 never propagates StashTranslation transactions downstream, and uses StashTranslation only to prefetch TLB contents. The MMU-600 always terminates StashTranslation transactions with a **BRESP** value of OKAY, even if no translation could be stored in the TLB.

The TBU ignores **AWPROT[0]** and **AWPROT[2]** for StashTranslation transactions, because they do not affect speculative translation requests.

AXI5 support

The AXI5 protocol includes extensions that are not included in previous AXI versions. The *Arm® AMBA® AXI and ACE Protocol Specification, AXI3, AXI4, AXI5, ACE and ACE5* defines these extensions.

The following table shows whether individual TCU and TBU interfaces support the AXI5 extensions.

Table 2-19 TCU and TBU interface support for AXI5 extensions

AXI5 extension	QTW/DVM	TBU TBS	TBU TBM
DVM_v8.1	Yes	-	-
Wakeup_Signals	Yes	Yes	Yes
Atomic_Transactions	-	-	-
Coherency_Connection_Signals	Yes	-	-
Cache_Stash_Transactions	-	Yes	Yes
DeAllocation_Transactions	-	Yes	Yes
Untranslated_Transactions	-	Yes	Yes
Poison	-	-	-
Check_Type	-	-	-
QoS_Accept	-	-	-
Trace_Signals	-	-	-
Loopback_Signals	-	-	-
NSAccess_Identifiers	-	-	-
Persist_CMO	-	Yes	Yes

Upstream ACE master restrictions

The ACE5 protocol places requirements on upstream ACE masters that support the Untranslated_Transactions extension.

For the ACE TBU configuration to be used with an upstream master that implements ACE5, the following requirements apply:

- Snoop transactions return an invalid cache state response unless they correspond to a Shareable cache line that has previously been modified with a successful ReadClean, ReadNotSharedDirty, ReadShared, ReadUnique, CleanUnique, or MakeUnique transaction. In particular:
  - Data that a ReadNoSnoop or ReadOnce transaction returns is either not cached, or cached in such a way that snoops of that cache line return an invalid cache state response.
  - If cache lines are allocated silently due to non-shared writes, then snoops to those lines also return an invalid cache state response.
- Shareable Write-Back, WriteClean, WriteEvict and Evict transactions always correspond to a Shareable cache line that has previously been modified with a successful ReadClean, ReadNotSharedDirty, ReadShared, ReadUnique, CleanUnique, or MakeUnique transaction. In particular, data that a ReadNoSnoop or ReadOnce transaction returns is either not cached, or written back with a WriteNoSnoop, WriteUnique, or WriteLineUnique transaction.

In addition, cache maintenance operations are not supported for upstream ACE masters.

Avoiding deadlock when using fully coherent ACE masters

You can take certain steps to prevent deadlock in systems where an ACE TBU configuration is used with a fully coherent ACE master.

The ACE protocol requires that certain write transactions, including WriteNoSnoop, must progress to any address without requiring any pending snoop transactions to progress. If a TBU is used with a fully coherent ACE master, WriteNoSnoop transactions might require the TCU to issue translation table walks. If these table walks depend on snoop transactions, then this ACE protocol requirement is not met. You can ensure that this requirement is met by ensuring that TCU transactions do not pass through a coherent interconnect.

Alternatively, you might be able to use a coherent interconnect such as the Arm CoreLink CCI-500 or CCI-550. Both of these interconnects ensure that interfaces that never issue snoop transactions are never blocked by snoop transactions. You can use CCI-500 or CCI-550 provided:

- All the TCU transactions are configured to use Device or Non-cacheable memory types, so that the TCU transactions never result in snoop transactions being performed.
- The TCU does not share an interconnect interface with any masters that can cause snoop transactions, so that the TCU transactions cannot be blocked by snoop transactions.

Chapter 3
Programmers model

This chapter describes the MMU-600 programmers model.

It contains the following sections:

3.1 About the programmers model

This section provides general information about the MMU-600 register properties.

The following information applies to the MMU-600 registers:

- The base address is not fixed, and can be different for any particular system implementation. The offset of each register from the base address is fixed.
- Do not attempt to access reserved or unused address locations. Attempting to access these locations can result in UNPREDICTABLE behavior.
- Unless otherwise stated in the accompanying text:
  - Do not modify undefined register bits.
  - Ignore undefined register bits on reads.
  - All register bits are reset to 0 by a system or powerup reset.
- Access type is described as follows:

  RW Read and write.

  RO Read only.

  WO Write only.

  RAZ Read as zero.

  WI Writes ignored.
- Bit positions that are described as reserved are:
  - In an RW register, RAZ/WI.
  - In an RO register, RAZ.
  - In a WO register, WI.

The MMU-600 registers are accessed using the PROG APB4 slave interface on the TCU, and cannot be accessed directly through any other slave interfaces.

Some registers are 64 bits, but the PROG APB4 interface is 32 bits. Because software accesses 64-bit registers 32 bits at a time, such accesses are not guaranteed to be 64-bit atomic. This behavior does not cause problems for software, because the SMMUv3 architecture does not require 64-bit atomic access to any registers.

The programmers model contains separate TBU and TCU regions for internal control, RAS, and identification registers. Accesses to unmapped or reserved registers are RAZ/WI. Non-secure accesses to Secure registers are RAZ/WI. The MMU-600 implements the identification register scheme that the SMMUv3 architecture defines.

The MMU-600 implements all the *Performance Monitor Counter Group* (PMCG) registers that the SMMUv3 architecture defines, except for:

- SMMU_PMCG_IRQ_CFG0.
- SMMU_PMCG_IRQ_CFG1.
- SMMU_PMCG_IRQ_CFG2.
- SMMU_PMCG_IRQ_STATUS.

The MMU-600 does not implement the following SMMUv3 architectural registers, and accesses to these locations are RAZ/WI:

- SMMU_IDR4.
- SMMU_STATUSR.
- SMMU_AGBPA.
- SMMU_GATOS_*.
- SMMU_S_IDR4.
- SMMU_S_AGBPA.
- SMMU_S_GATOS_*.
- SMMU_VATOS_*.

See the *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3.0 and version 3.1* for more information about the SMMU architectural registers.

3.2 SMMU architectural registers

The MMU-600 implements many of the SMMU architectural registers, as defined by the *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3.0 and version 3.1*.

The following table lists the SMMUv3 architectural registers that the MMU-600 implements.

Note

All writable register fields reset to 0 unless the SMMU architecture specifies otherwise.

Table 3-1 SMMUv3 architectural registers

Register	Name	Description
SMMU_S_IDR0 - SMMU_S_IDR3	SMMU Secure feature Identification Registers	Provides information about the Secure features that the SMMU implementation supports.
SMMU_S_CR0	Secure global Control Register 0	Provides global configuration of the Secure SMMU.
SMMU_S_CR0ACK	Secure global Control Register 0 update Acknowledge	Provides acknowledgement of completion of updates to SMMU_S_CR0.
SMMU_S_CR1 SMMU_S_CR2	Secure global Control Registers	Provides the controls for Secure table and queue access attributes.
SMMU_S_INIT	Secure Initialization control register	Provides a control to invalidate all Secure SMMU caching on system initialization.
SMMU_S_GBPA	Secure Global Bypass Attribute register	Controls the global bypass attributes that are used for transactions from Secure streams when the MMU is disabled.
SMMU_S_IRQ_CTRL	Secure Interrupt Control register	Contains enables for SMMU interrupts.
SMMU_S_IRQ_CTRLACK	Secure Interrupt Control register update Acknowledge	Provides acknowledgement of the completion of updates to SMMU_S_IRQ_CTRL.
SMMU_S_GERROR	Secure Global Error status register	Provides information on Secure global programming interface errors.
SMMU_S_GERRORN	Secure Global Error Acknowledgement register	Contains the acknowledgement fields for SMMU_S_GERROR errors.
SMMU_S_GERROR_IRQ_CFG0 - SMMU_S_GERROR_IRQ_CFG2	Secure Global Error IRQ Configuration register	Contains the Secure MSI address configuration for the GERROR IRQ.
SMMU_S_STRTAB_BASE	Secure Stream Table Base address register	Contains the base address and attributes for the Secure Stream table.
SMMU_S_STRTAB_BASE_CFG	Secure Stream Table Base Configuration register	Contains configuration fields for the Secure Stream table.
SMMU_S_CMDQ_BASE	Secure Command queue Base address register	Contains the base address and attributes for the Secure Command queue.

Table 3-1 SMMUv3 architectural registers (continued)

Register	Name	Description
SMMU_S_CMDQ_PROD	Secure Command queue Producer index register	Contains the Secure Command queue index for writes by the producer.
SMMU_S_CMDQ_CONS	Secure Command queue Consumer index register	Contains the Secure Command queue index for reads by the consumer.
SMMU_S_EVENTQ_BASE	Secure Event queue Base address register	Contains the base address and attributes for the Secure Event queue.
SMMU_S_EVENTQ_PROD	Secure Event queue Producer index register	Contains the Secure Event queue index for writes by the producer.
SMMU_S_EVENTQ_CONS	Secure Event queue Consumer index register	Contains the Secure Event queue index for reads by the consumer.
SMMU_S_EVENTQ_IRQ_CFG0 - SMMU_S_EVENTQ_IRQ_CFG2	Secure Event queue IRQ Configuration registers	Contains the MSI address configuration for the Secure Event queue IRQ.
SMMU_IDR0 - SMMU_IDR3 SMMU_IDR5	SMMU feature Identification Registers	Provides information about the features that the SMMU implementation supports.
SMMU_IIDR	Implementation Identification Register	Provides implementer, part, and revision information for the SMMU implementation.
SMMU_AIDR	Architecture Identification Register	Identifies the SMMU architecture version to which the implementation conforms.
SMMU_CR0	Non-secure global Control Register 0	Provides the controls for the global configuration of the Non-secure SMMU.
SMMU_CR0ACK	Non-secure global Control Register 0 update Acknowledge register	Provides acknowledgement of completion of updates to SMMU_CR0.
SMMU_CR1	Non-secure global Control Register 1	Provides the controls for Non-secure table and queue access attributes.
SMMU_CR2	Non-secure global Control Register 2	Provides the controls for the configuration of the global Non-secure features.
SMMU_GBPA	Non-secure Global Bypass Attribute register	Controls the global bypass attributes that are used for transactions from Non-secure streams when the MMU is disabled.
SMMU_IRQ_CTRL	Non-secure Interrupt Control register	Provides IRQ enable flags for edge-triggered wired outputs, if implemented, and MSI writes, if implemented.
SMMU_IRQ_CTRLACK	Non-secure Interrupt Control register update Acknowledge register	Provides acknowledgement of the completion of updates to SMMU_IRQ_CTRL.
SMMU_GERROR	Non-secure Global Error status register	Provides information about Non-secure global programming interface errors.

Table 3-1 SMMUv3 architectural registers (continued)

Register	Name	Description
SMMU_GERRORN	Non-secure Global Error acknowledgement register	Contains the acknowledgement fields for SMMU_GERROR errors.
SMMU_GERROR_IRQ_CFG0	Non-secure Global Error IRQ Configuration register 0	Contains the MSI address configuration for the GERROR IRQ.
SMMU_GERROR_IRQ_CFG1	Non-secure Global Error IRQ Configuration register 1	Contains the MSI payload configuration for the GERROR IRQ.
SMMU_GERROR_IRQ_CFG2	Non-secure Global Error IRQ Configuration register 2	Contains the MSI attribute configuration for the GERROR IRQ.
SMMU_STRTAB_BASE	Non-secure Stream Table Base address register	Contains the base address and attributes for the Non-secure Stream table.
SMMU_STRTAB_BASE_CFG	Non-secure Stream Table Configuration register	Contains configuration fields for the Non-secure Stream table.
SMMU_CMDQ_BASE	Non-secure Command queue Base address register	Contains the base address and attributes for the Non-secure Command queue.
SMMU_CMDQ_PROD	Non-secure Command queue Producer index register	Contains the Non-secure Command queue index for writes by the producer.
SMMU_CMDQ_CONS	Non-secure Command queue Consumer index register	Contains the Non-secure Command queue index for reads by the consumer.
SMMU_EVENTQ_BASE	Non-secure Event queue Base address register	Contains the base address and attributes for the Non-secure Event queue.
SMMU_EVENTQ_PROD	Non-secure Event queue Producer index register	Contains the Non-secure Event queue index for writes by the producer.
SMMU_EVENTQ_CONS	Non-secure Event queue Consumer index register	Contains the Non-secure Event queue index for reads by the consumer.
SMMU_EVENTQ_IRQ_CFG0	Non-secure Event queue IRQ Configuration register 0	Contains the MSI address configuration for the Event queue IRQ.
SMMU_EVENTQ_IRQ_CFG1	Non-secure Event queue IRQ Configuration register 1	Contains the MSI payload configuration for the Event queue IRQ.
SMMU_EVENTQ_IRQ_CFG2	Non-secure Event queue IRQ Configuration register 2	Contains the MSI attribute configuration for the Event queue IRQ.
SMMU_PRIQ_BASE	Non-secure PRI queue Base address register	Contains the base address and attributes for the Non-secure PRI queue.
SMMU_PRIQ_PROD	Non-secure PRI queue Producer index register	Contains the Non-secure PRI queue index for writes by the producer.
SMMU_PRIQ_CONS	Non-secure PRI queue Consumer index register	Contains the Non-secure PRI queue index for reads by the consumer.

Table 3-1 SMMUv3 architectural registers (continued)

Register	Name	Description
SMMU_PRIQ_IRQ_CFG0	Non-secure PRI queue IRQ Configuration register 0	Contains the MSI address configuration for the PRI queue IRQ.
SMMU_PRIQ_IRQ_CFG1	Non-secure PRI queue IRQ Configuration register 1	Contains the MSI payload configuration for the PRI queue IRQ.
SMMU_PRIQ_IRQ_CFG2	Non-secure PRI queue IRQ Configuration register 2	Contains the MSI attribute configuration for the PRI queue IRQ.

The MMU-600 implements an SMMUv3 *Performance Monitor Counter Group* (PMCG) in the TCU and in each TBU. The following table lists the registers that the MMU-600 implements in each PMCG.

Table 3-2 SMMUv3 PMCG registers

Register	Name	Description
SMMU_PMCG_EVCNTR0 - SMMU_PMCG_EVCNTR3	SMMU PMCG Event Counter registers	Contains the values of the event counters.
SMMU_PMCG_EVTYPER0 - SMMU_PMCG_EVTYPER3	SMMU PMCG Event Type configuration registers	Configures the events that the corresponding counter counts.
SMMU_PMCG_SVR0 - SMMU_PMCG_SVR3	SMMU PMCG Shadow Value Registers	Contains the shadow value of the corresponding event counter.
SMMU_PMCG_SMR0	SMMU PMCG Stream Match filter Register	Configures the stream match filter for the corresponding event counter.
SMMU_PMCG_CNTENSET0	SMMU PMCG Counter Enable Set register	Provides the set mechanism for the counter enables.
SMMU_PMCG_CNTENCLR0	SMMU PMCG Counter Enable Clear register	Provides the clear mechanism for the counter enables.
SMMU_PMCG_INTENSET0	SMMU PMCG Interrupt contribution Enable Set register	Provides the set mechanism for the counter interrupt contribution enables.
SMMU_PMCG_INTENCLR0	SMMU PMCG Interrupt contribution Enable Clear register	Provides the clear mechanism for the counter interrupt enables.
SMMU_PMCG_OVSCLR0	SMMU PMCG Overflow Status Clear register	Provides the clear mechanism for the overflow status bits and provides read access to the overflow status bit values.
SMMU_PMCG_OVSSET0	SMMU PMCG Overflow Status Set register	Provides the set mechanism for the overflow status bits and provides read access to the overflow status bit values.
SMMU_PMCG_CAPR	SMMU PMCG Counter shadow value Capture Register	Controls the counter shadow value capture mechanism.

Table 3-2 SMMUv3 PMCG registers (continued)

Register	Name	Description
SMMU_PMCG_SCR	SMMU PMCG Secure Control Register	Secure Control Register.
SMMU_PMCG_CFGR	SMMU PMCG Configuration information Register	Provides information about the PMCG implementation.
SMMU_PMCG_CR	SMMU PMCG Control Register	Contains the Performance Monitor control flags.
SMMU_PMCG_CEID0 - SMMU_PMCG_CEID1	SMMU PMCG Common Event ID registers	Contains the lower and upper 64 bits of the Common Event identification bitmap.
SMMU_PMCG_IRQ_CTRL	SMMU PMCG IRQ enable register	Contains the Performance Monitors IRQ enable.
SMMU_PMCG_IRQ_CTRLACK	SMMU PMCG IRQ enable Acknowledge register	Provides acknowledgement of the completion of updates to SMMU_PMCG_IRQ_CTRL.
SMMU_PMCG_AIDR	SMMU PMCG Architecture Identification Register	Provides the Performance Monitor Architecture Identification.
SMMU_PMCG_ID_REGS	ID registers	IMPLEMENTATION DEFINED.
SMMU_PMCG_PMAUTHSTATUS	PMU Authentication Status register	Performance Monitor authentication status.
SMMU_PMCG_PMDEVARCH	PMU Device Architecture register	Performance Monitor architecture identifier.
SMMU_PMCG_PMDEVTYPE	PMU Device Type register	Performance Monitor device type.

3.3 MMU-600 memory map

The MMU-600 memory map contains all registers.

The following table shows the MMU-600 memory map with the maximum number of implemented TBUs.

Table 3-3 MMU-600 memory map

Address range	Description
0x000000 - 0x03FFFC	TCU registers.
0x040000 - 0x05FFFC	TBU0 registers.
0x060000 - 0x07FFFC	TBU1 registers
0x080000 - 0x09FFFC	TBU2 registers.
.	.
.	.
.	.
0x7C0000 - 0x7DFFFC	TBU60 registers.
0x7E0000 - 0x7FFFFC	TBU61 registers.

Note

All TBU and TCU register addresses in this manual are described relative to the beginning of the respective address range for the component.

The following table shows the MMU-600 TCU memory map.

Table 3-4 MMU-600 TCU memory map

Address	Description
0x00000 - 0x0FFFC	TCU registers, page 0, including: • SMMUv3 registers, page 0. • TCU *Performance Monitor Counter Group* (PMCG) registers, page 0, starting at offset 0x02000. • TCU IMPLEMENTATION DEFINED registers.
0x10000 - 0x1FFFC	TCU registers, page 1. This address range contains the SMMUv3 registers, page 1.
0x20000 - 0x2FFFC	TCU registers, page 2. This address range contains the TCU PMCG registers, page 1, starting at offset 0x22000.
0x30000 - 0x3FFFC	Reserved.

The following table shows the MMU-600 TBU memory map.

Table 3-5 MMU-600 TBU memory map

Address	Description
`0x00000` - `0x0FFFC`	TBU registers, page 0, including: • TBU PMCG registers, page 0, starting at offset `0x02000`. • TBU IMPLEMENTATION DEFINED registers.
`0x10000` - `0x1FFFC`	TBU registers, page 1. This address range contains the TBU PMCG registers, page 1, starting at offset `0x12000`.

3.4 Register summary

The register summary lists all MMU-600 registers and some key characteristics.

TBU identification register summary

The following table shows the TBU identification registers in offset order from the base memory address.

Table 3-6 TBU identification register summary

Offset	Name	Type	Description
0x00FD0	SMMU_PIDR4	RO	*3.9 TBU Component and Peripheral ID Registers* on page 3-89.
0x00FD4	SMMU_PIDR5	RO	
0x00FD8	SMMU_PIDR6	RO	
0x00FDC	SMMU_PIDR7	RO	
0x00FE0	SMMU_PIDR0	RO	
0x00FE4	SMMU_PIDR1	RO	
0x00FE8	SMMU_PIDR2	RO	
0x00FEC	SMMU_PIDR3	RO	
0x00FF0	SMMU_CIDR0	RO	
0x00FF4	SMMU_CIDR1	RO	
0x00FF8	SMMU_CIDR2	RO	
0x00FFC	SMMU_CIDR3	RO	

TBU RAS register summary

The following table shows the TBU *Reliability, Availability, and Serviceability* (RAS) registers in offset order from the base memory address.

Table 3-7 TBU RAS register summary

Offset	Name	Type	Description
0x08E80	TBU_ERRFR	RO	*3.12.1 TBU_ERRFR* on page 3-93.
0x08E88	TBU_ERRCTLR	RW	*3.12.2 TBU_ERRCTLR* on page 3-93.
0x08E90	TBU_ERRSTATUS	RW	*3.12.3 TBU_ERRSTATUS* on page 3-94 .
0x08EC0	TBU_ERRGEN	RW	*3.12.4 TBU_ERRGEN* on page 3-95.

TBU microarchitectural register summary

The following table shows the TBU microarchitectural registers in offset order from the base memory address.

Table 3-8 TBU microarchitectural register summary

Offset	Name	Type	Description
0x08E00	TBU_CTRL	RW	*3.11.1 TBU_CTRL* on page 3-91.
0x08E18	TBU_SCR	RW	*3.11.2 TBU_SCR* on page 3-91.

TCU identification register summary

The following table shows the TCU identification registers in offset order from the base memory address.

Table 3-9 TCU identification register summary

Offset	Name	Type	Description
0x00FD0	SMMU_PIDR4	RO	*3.5 TCU Component and Peripheral ID Registers* on page 3-74.
0x00FD4	SMMU_PIDR5	RO	
0x00FD8	SMMU_PIDR6	RO	
0x00FDC	SMMU_PIDR7	RO	
0x00FE0	SMMU_PIDR0	RO	
0x00FE4	SMMU_PIDR1	RO	
0x00FE8	SMMU_PIDR2	RO	
0x00FEC	SMMU_PIDR3	RO	
0x00FF0	SMMU_CIDR0	RO	
0x00FF4	SMMU_CIDR1	RO	
0x00FF8	SMMU_CIDR2	RO	
0x00FFC	SMMU_CIDR3	RO	

TCU and TBU PMU identification register summary

The TCU and the TBU use the same PMU identification registers. The following table shows the TCU and TBU PMU identification registers in offset order from the base memory address.

Table 3-10 TCU and TBU PMU identification register summary

Offset	Name	Type	Description
0x02FB8	SMMU_PMCG_PMAUTHSTATUS	RO	*3.6 TCU PMU Component and Peripheral ID Registers* on page 3-75. *3.10 TBU PMU Component and Peripheral ID Registers* on page 3-90.
0x02FD0	SMMU_PMCG_PIDR4	RO	
0x02FD4	SMMU_PMCG_PIDR5	RO	
0x02FD8	SMMU_PMCG_PIDR6	RO	
0x02FDC	SMMU_PMCG_PIDR7	RO	
0x02FE0	SMMU_PMCG_PIDR0	RO	
0x02FE4	SMMU_PMCG_PIDR1	RO	
0x02FE8	SMMU_PMCG_PIDR2	RO	
0x02FEC	SMMU_PMCG_PIDR3	RO	
0x02FF0	SMMU_PMCG_CIDR0	RO	
0x02FF4	SMMU_PMCG_CIDR1	RO	
0x02FF8	SMMU_PMCG_CIDR2	RO	
0x02FFC	SMMU_PMCG_CIDR3	RO	

TCU RAS register summary

The following table shows the TCU RAS registers in offset order from the base memory address.

Table 3-11 TCU RAS register summary

Offset	Name	Type	Description
0x08E80	TCU_ERRFR	RO	*3.8.1 TCU_ERRFR* on page 3-84.
0x08E88	TCU_ERRCTLR	RW	*3.8.2 TCU_ERRCTLR* on page 3-84.
0x08E90	TCU_ERRSTATUS	RW	*3.8.3 TCU_ERRSTATUS* on page 3-85.
0x08EC0	TCU_ERRGEN	RW	*3.8.4 TCU_ERRGEN* on page 3-87.

TCU microarchitectural register summary

The following table shows the TCU microarchitectural registers in offset order from the base memory address.

Table 3-12 TCU microarchitectural register summary

Offset	Name	Type	Description
0x08E00	TCU_CTRL	RW	*3.7.1 TCU_CTRL* on page 3-76.
0x08E04	TCU_QOS	RW	*3.7.2 TCU_QOS* on page 3-77.
0x08E08	TCU_CFG	RO	*3.7.3 TCU_CFG* on page 3-79.
0x08E10	TCU_STATUS	RO	*3.7.4 TCU_STATUS* on page 3-79.
0x08E18	TCU_SCR	RW	*3.7.5 TCU_SCR* on page 3-80.
0x09000 - 0x093FC	TCU_NODE_CTRL*n*	RW	*3.7.6 TCU_NODE_CTRLn* on page 3-81.
0x09400 - 0x097FC	TCU_NODE_STATUS*n*	RO	*3.7.7 TCU_NODE_STATUSn* on page 3-82.

3.5 TCU Component and Peripheral ID Registers

The component and peripheral identity registers comply with the format that the Arm CoreLink and CoreSight components use, and recommended in the SMMUv3 architecture. They provide key information about the MMU-600 hardware, including the product and associated revision number. They also identify Arm as the designer of the SMMU.

These registers are all read-only. Each field defines a single byte in the least significant 8 bits, and the most significant 24 bits are reserved. The least significant 8 bits of the four Component ID registers form a single 32-bit conceptual ID register. In a similar way, the defined fields of the eight Peripheral ID registers form a conceptual 64-bit ID register.

Table 3-13 TCU Component and Peripheral ID registers bit assignments

Register	Offset	Bits	Value	Function
SMMU_PIDR4	0x00FD0	[7:4]	0x0	4KB region count.
		[3:0]	0x4	JEP106 continuation code for Arm.
SMMU_PIDR5	0x00FD4	[7:0]	0x00	Reserved.
SMMU_PIDR6	0x00FD8	[7:0]	0x00	Reserved.
SMMU_PIDR7	0x00FDC	[7:0]	0x00	Reserved.
SMMU_PIDR0	0x00FE0	[7:0]	0x83	Part number[7:0].
SMMU_PIDR1	0x00FE4	[7:4]	0xB	JEP106 ID code[3:0] for Arm.
		[3:0]	0x4	Part number[11:8].
SMMU_PIDR2	0x00FE8	[7:4]	0x1	MMU-600 major revision. The value 0x1 indicates major product revision r1.
		[3]	0x1	The component uses a manufacturer identity code that JEDEC allocates, according to the JEP106 specification.
		[2:0]	0x3	JEP106 ID code[6:4] for Arm.
SMMU_PIDR3	0x00FEC	[7:4]	MAX[0x0,ecorevnum]	MMU-600 minor revision. The value 0x0 indicates minor product revision p0.
		[3:0]	0x0	CMOD. This field is not used.
SMMU_CIDR0	0x00FF0	[7:0]	0x0D	Preamble.
SMMU_CIDR1	0x00FF4	[7:0]	0xF0	
SMMU_CIDR2	0x00FF8	[7:0]	0x05	
SMMU_CIDR3	0x00FFC	[7:0]	0xB1	

3.6 TCU PMU Component and Peripheral ID Registers

The component and peripheral identity registers comply with the format that Arm CoreLink and CoreSight components use, and recommended in the SMMUv3 architecture. They provide key information about the MMU-600 hardware, including the product and associated revision number. They also identify Arm as the designer of the SMMU.

These registers are all read-only. Each field defines a single byte in the least significant 8 bits, and the most significant 24 bits are reserved. The least significant 8 bits of the four Component ID registers form a single 32-bit conceptual ID register. In a similar way, the defined fields of the eight Peripheral ID registers form a conceptual 64-bit ID register.

Table 3-14 TCU PMU Component and Peripheral ID registers bit assignments

Register	Offset	Bits	Value	Function
SMMU_PMCG_PMAUTHSTATUS	0x02FB8	[7:0]	0x00	No authentication interface is implemented.
SMMU_PMCG_PIDR4	0x02FD0	[7:4]	0x0	4KB region count.
		[3:0]	0x4	JEP106 continuation code for Arm.
SMMU_PMCG_PIDR5	0x02FD4	[7:0]	0x00	Reserved.
SMMU_PMCG_PIDR6	0x02FD8	[7:0]	0x00	Reserved.
SMMU_PMCG_PIDR7	0x02FDC	[7:0]	0x00	Reserved.
SMMU_PMCG_PIDR0	0x02FE0	[7:0]	0x83	Part number[7:0].
SMMU_PMCG_PIDR1	0x02FE4	[7:4]	0xB	JEP106 ID code[3:0] for Arm.
		[3:0]	0x4	Part number[11:8].
SMMU_PMCG_PIDR2	0x02FE8	[7:4]	0x1	MMU-600 revision. The value 0x1 indicates major product revision r1.
		[3]	0x1	The component uses a manufacturer identity code that JEDEC allocates, according to the JEP106 specification.
		[2:0]	0x3	JEP106 ID code[6:4] for Arm.
SMMU_PMCG_PIDR3	0x02FEC	[7:4]	MAX[0x0,ecorevnum]	MMU-600 minor revision. The value 0x0 indicates minor product revision p0.
		[3:0]	0x0	CMOD. This field is not used.
SMMU_PMCG_CIDR0	0x02FF0	[7:0]	0x0D	Preamble.
SMMU_PMCG_CIDR1	0x02FF4	[7:0]	0x90	
SMMU_PMCG_CIDR2	0x02FF8	[7:0]	0x05	
SMMU_PMCG_CIDR3	0x02FFC	[7:0]	0xB1	

3.7 TCU microarchitectural registers

You can set the TCU microarchitectural registers at boot time to optimize TCU behavior for your system. Arm recommends the default settings for most systems.

This section contains the following subsections:

3.7.1 TCU_CTRL

The TCU Control register disables TCU features. You can disable individual walk caches to improve performance in some systems if the hit rate of the individual walk cache is very low. Do not modify the AUX bits unless directed to do so by Arm.

The TCU_CTRL characteristics are:

Usage constraints

When TCU_SCR.NS_UARCH = 0, Non-secure accesses to this register are RAZ/WI.

Writes to this register are possible only when both SMMU_CR0.SMMUEN = 0 and SMMU_S_CR0.SMMUEN = 0. Writes at other times are ignored.

After modifying this register, software must issue an INV_ALL operation using the SMMU_S_INIT register, before it sets SMMUEN to 1. Failure to issue an INV_ALL operation results in UNPREDICTABLE behavior.

Configurations

This register exists in all TCU configurations.

Attributes

Offset `0x08E00`

Type RW

Reset `0x00000000`

Width 32

The following figure shows the bit assignments.

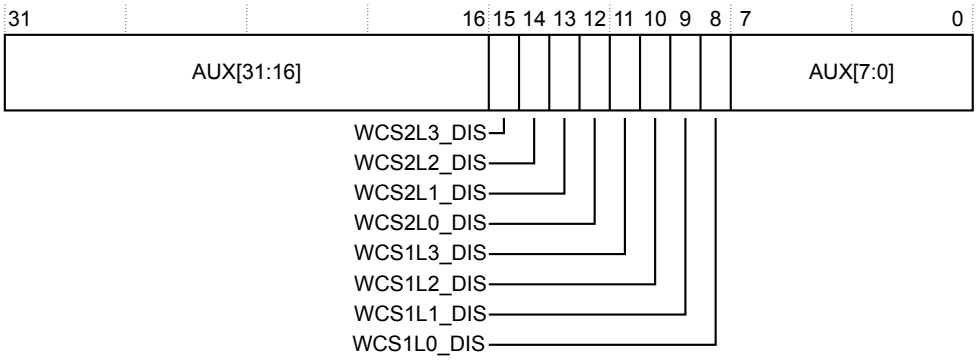

Figure 3-1 TCU_CTRL register bit assignments

The following table shows the bit assignments.

Table 3-15 TCU_CTRL register bit assignments

Bits	Name	Function
[31:16]	AUX[31:16]	Leave each of these bits as 0.
[15]	WCS2L3_DIS	Walk cache disable. **0** Stage 2 level 3 walk cache is enabled. **1** Stage 2 level 3 walk cache is disabled.
[14]	WCS2L2_DIS	Walk cache disable. **0** Stage 2 level 2 walk cache is enabled. **1** Stage 2 level 2 walk cache is disabled.
[13]	WCS2L1_DIS	Walk cache disable. **0** Stage 2 level 1 walk cache is enabled. **1** Stage 2 level 1 walk cache is disabled.
[12]	WCS2L0_DIS	Walk cache disable. **0** Stage 2 level 0 walk cache is enabled. **1** Stage 2 level 0 walk cache is disabled.
[11]	WCS1L3_DIS	Walk cache disable. **0** Stage 1 level 3 walk cache is enabled. **1** Stage 1 level 3 walk cache is disabled.
[10]	WCS1L2_DIS	Walk cache disable. **0** Stage 1 level 2 walk cache is enabled. **1** Stage 1 level 2 walk cache is disabled.
[9]	WCS1L1_DIS	Walk cache disable. **0** Stage 1 level 1 walk cache is enabled. **1** Stage 1 level 1 walk cache is disabled.
[8]	WCS1L0_DIS	Walk cache disable. **0** Stage 1 level 0 walk cache is enabled. **1** Stage 1 level 0 walk cache is disabled.
[7:0]	AUX[7:0]	Leave each of these bits as 0.

3.7.2 TCU_QOS

The TCU *Quality of Service* (QoS) register specifies **AxQOS** values for each transaction type that is issued on the QTW/DVM interface. The MMU-600 does not use this value internally, but a downstream interconnect can use the value to control how it prioritizes transactions.

The **AxQOS** value that is associated with each transaction does not take account of other transactions that are blocked behind it. For example, although higher priority translations are normally progressed

before lower priority translations, a low priority translation table walk might prevent the TCU from issuing a translation table walk with a higher priority.

The TCU_QOS characteristics are:

Usage constraints

When TCU_SCR.NS_UARCH = 0, Non-secure accesses to this register are RAZ/WI.

Writes to this register are possible only when both SMMU_CR0.SMMUEN = 0 and SMMU_S_CR0.SMMUEN = 0. Writes at other times are ignored.

After modifying this register, software must issue an INV_ALL operation using the SMMU_S_INIT register, before it sets SMMUEN to 1. Failure to issue an INV_ALL operation results in UNPREDICTABLE behavior.

Configurations

This register exists in all TCU configurations.

Attributes

Offset `0x08E04`

Type RW

Reset `0x00000000`

Width 32

The following figure shows the bit assignments.

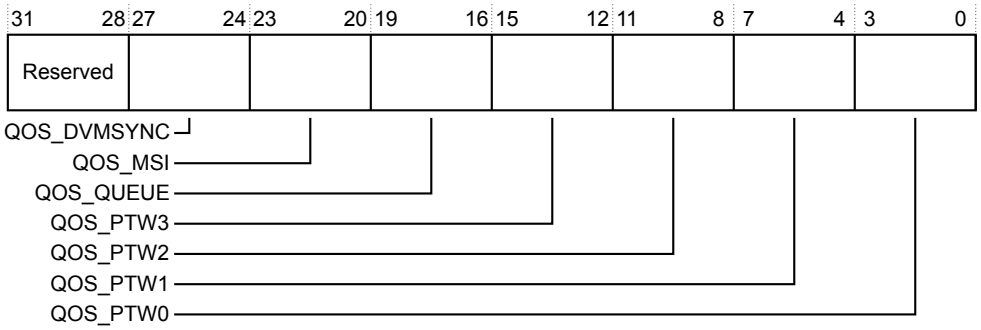

Figure 3-2 TCU_QOS register bit assignments

The following table shows the bit assignments.

Table 3-16 TCU_QOS register bit assignments

Bits	Name	Function
[31:28]	-	Reserved.
[27:24]	QOS_DVMSYNC	The **AxQOS** value that is used for DVM Sync Completion messages.
[23:20]	QOS_MSI	The **AxQOS** value that is used for MSIs.
[19:16]	QOS_QUEUE	The **AxQOS** value that is used for queue accesses.
[15:12]	QOS_PTW3	The **AxQOS** value that is used for translation table walks for translations where TCU_NODE_CTRLn.PRIORITY = 3 for the requesting node.
[11:8]	QOS_PTW2	The **AxQOS** value that is used for translation table walks for translations where TCU_NODE_CTRLn.PRIORITY = 2 for the requesting node.

Table 3-16 TCU_QOS register bit assignments (continued)

Bits	Name	Function
[7:4]	QOS_PTW1	The **AxQOS** value that is used for translation table walks for translations where TCU_NODE_CTRLn.PRIORITY = 1 for the requesting node.
[3:0]	QOS_PTW0	The **AxQOS** value that is used for translation table walks for translations where TCU_NODE_CTRLn.PRIORITY = 0 for the requesting node.

3.7.3 TCU_CFG

This is the TCU Configuration Information register.

Its characteristics are:

Usage constraints

When TCU_SCR.NS_UARCH = 0, Non-secure accesses to this register are RAZ.

Configurations

This register exists in all TCU configurations.

Attributes

Offset `0x08E08`

Type RO

Reset See register bit assignments.

Width 32

The following figure shows the bit assignments.

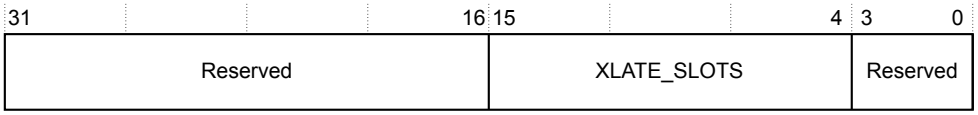

Figure 3-3 TCU_CFG register bit assignments

The following table shows the bit assignments.

Table 3-17 TCU_CFG register bit assignments

Bits	Name	Function
[31:16]	-	Reserved.
[15:4]	XLATE_SLOTS	The number of translation slots that are available for sharing between all nodes. The reset value of this field is `TCUCFG_XLATE_SLOTS`.
[3:0]	-	Reserved.

3.7.4 TCU_STATUS

This is the TCU Status Information register.

Its characteristics are:

Usage constraints

When TCU_SCR.NS_UARCH = 0, Non-secure accesses to this register are RAZ.

Configurations

This register exists in all TCU configurations.

Attributes

Offset 0x08E10

Type RO

Reset 0x00000000

Width 32

The following figure shows the bit assignments.

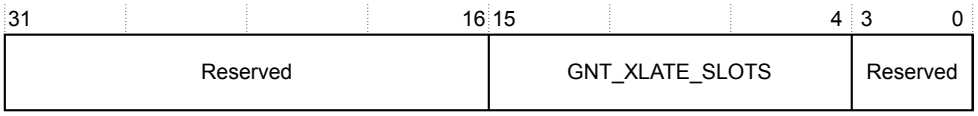

Figure 3-4 TCU_STATUS register bit assignments

The following table shows the bit assignments.

Table 3-18 TCU_STATUS register bit assignments

Bits	Name	Function
[31:16]	-	Reserved.
[15:4]	GNT_XLATE_SLOTS	This is the number of translation slots that are currently allocated to connected nodes. You can use this value for debugging purposes.
[3:0]	-	Reserved.

3.7.5 TCU_SCR

The TCU Secure Control register controls whether Non-secure software is permitted to access each TCU register group.

The TCU_SCR characteristics are:

Usage constraints

Non-secure accesses to this register are RAZ/WI.

This register does not control Secure access to the MMU-600 PMU registers. To control Secure PMU register access, use the SMMU_PMCG_SCR register.

Configurations

This register exists in all TCU configurations.

Attributes

Offset 0x08E18

Type RW

Reset See register bit assignments.

Width 32

The following figure shows the bit assignments.

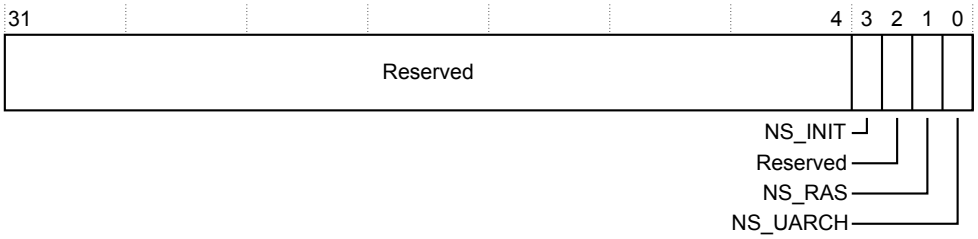

Figure 3-5 TCU_SCR register bit assignments

The following table shows the bit assignments.

Table 3-19 TCU_UARCH_TCU_SCR register bit assignments

Bits	Name	Function
[31:4]	-	Reserved
[3]	NS_INIT	Non-secure register access to SMMU_S_INIT. When this bit is set to 0, Non-secure accesses to the SMMU_S_INIT register are RAZ/WI. The **sec_override** input signal defines the reset value of this bit.
[2]	-	Reserved
[1]	NS_RAS	Non-secure register access is permitted for RAS registers. When this bit is set to 0, Non-secure accesses to register addresses `0x08E80`–`0x08EC0` are RAZ/WI. The **sec_override** input signal defines the reset value of this bit.
[0]	NS_UARCH	Non-secure register access is permitted for MMU-600 registers. When this bit is set to 0, Non-secure accesses to register addresses `0x08E00`–`0x08E7C` and `0x09000`–`0x093FC` are RAZ/WI. The **sec_override** input signal defines the reset value of this bit. If your implementation might use Secure translation, Arm recommends setting this bit to 0.

3.7.6 TCU_NODE_CTRL*n*

Each TCU Node Control register controls how the TCU communicates with a single node. A node is a DTI master that is typically either a TBU or a PCIe Root Complex that implements ATS.

The TCU_NODE_CTRL*n* characteristics are:

Usage constraints

The DIS_DMV bit can be used for TBU nodes, but is ignored for ATS nodes.

When TCU_SCR.NS_UARCH = 0, Non-secure accesses to this register are RAZ/WI.

Writes to this register are possible only when both SMMU_CR0.SMMUEN = 0 and SMMU_S_CR0.SMMUEN = 0. Writes at other times are ignored.

After modifying this register, software must issue an INV_ALL operation using the SMMU_S_INIT register, before it sets SMMUEN to 1. Failure to issue an INV_ALL operation results in UNPREDICTABLE behavior.

Configurations

The value of the `TCUCFG_NUM_TBU` configuration parameter defines *n*, that is, the number of TCU_NODE_CTRL registers that are implemented. Each register has an address width of 4 bytes, therefore the offset of a register *n* is:

`0x09000` + (4 × *n*)

Attributes

Offset 0x09000-0x093FC

Type RW

Reset 0x00000000

Width 32

The following figure shows the bit assignments.

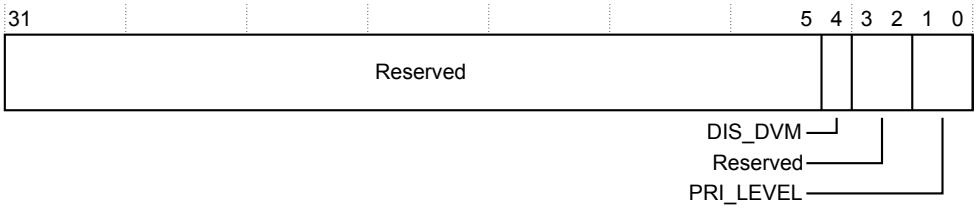

Figure 3-6 TCU_NODE_CTRL register bit assignments

The following table shows the bit assignments.

Table 3-20 TCU_NODE_CTRL register bit assignments

Bits	Name	Function
[31:5]	-	Reserved.
[4]	DIS_DVM	Disable DVM. When this bit is set to 1, the corresponding node does not participate in DVM invalidation. Set this bit to 1 to improve performance if the node is slow to respond to invalidations issued over DTI. **Note** This bit is ignored for connected DTI-ATS masters, because they never participate in DVM invalidation.
[3:2]	-	Reserved.
[1:0]	PRI_LEVEL	Priority level. This field indicates the priority level of the corresponding node. Translation requests from a node with a higher priority level are normally progressed before requests from a node with a lower priority level.

3.7.7 TCU_NODE_STATUS*n*

Each TCU Node Status register provides the status of a DTI master. A node is a DTI master that is typically either a TBU or a PCIe Root Complex that implements ATS.

The TCU_NODE_STATUS*n* characteristics are:

Usage constraints

This register indicates the status of the corresponding node only when the node is connected.

When TCU_SCR.NS_UARCH = 0, Non-secure accesses to this register are RAZ.

Configurations

The value of the TCUCFG_NUM_TBU configuration parameter defines the number of TCU_NODE_CTRL registers that are implemented. Each register has an address width of 4 bytes, therefore the offset of a register *n* is:

0x09400 + (4 × *n*)

Attributes

Offset 0x09400-0x097FC

Type RO

Reset 0x00000000

Width 32

The following figure shows the bit assignments.

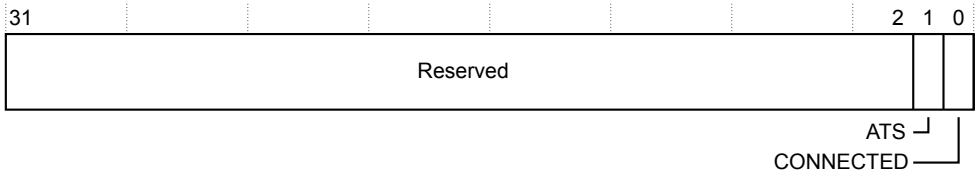

Figure 3-7 TCU_NODE_STATUS register bit assignments

The following table shows the bit assignments.

Table 3-21 TCU_NODE_STATUS register bit assignments

Bits	Name	Function
[31:2]	-	Reserved.
[1]	ATS	ATS implemented. 0 When this bit is set to 0, the corresponding node is a TBU that is connected to the TCU using the DTI-TBU protocol. 1 When this bit is set to 1, the corresponding node is a PCIe Root Complex that supports ATS, and is connected to the TCU using the DTI-ATS protocol.
[0]	CONNECTED	DTI link is connected. 0 When this bit is set to 0, the DTI link for the corresponding node is not connected. 1 When this bit is set to 1, the DTI link for the corresponding node is connected. If a DTI link is not connected, accesses to TBU registers are RAZ/WI. However, the state might change between reading this register and attempting to access the TBU.

3.8 TCU RAS registers

The MMU-600 includes TCU registers that are related to *Reliability, Availability, and Serviceability* (RAS).

This section contains the following subsections:

3.8.1 TCU_ERRFR

This is the TCU Error Feature register. Use this register to discover how the TCU handles errors.

The TCU_ERRFR characteristics are:

Usage constraints

This register is read-only. When TCU_SCR.NS_RAS = 0, Non-secure accesses to this register are RAZ.

Configurations

This register exists in all TCU configurations.

Attributes

Offset `0x08E80`

Type RO

Reset `0x00000082`

Width 32

The following figure shows the bit assignments.

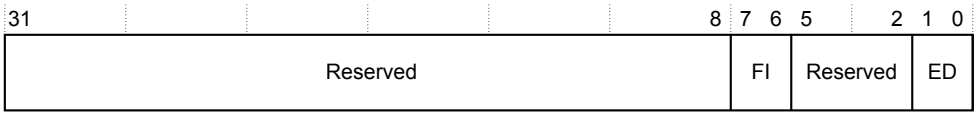

Figure 3-8 TCU_ERRFR register bit assignments

The following table shows the bit assignments.

Table 3-22 TCU_ERRFR register bit assignments

Bits	Name	Function
[31:8]	-	Reserved
[7:6]	FI	The value `0b10` indicates that the fault handling interrupt is controllable.
[5:2]	-	Reserved
[1:0]	ED	The value `0b01` indicates that TCU error detection is always enabled.

3.8.2 TCU_ERRCTLR

This is the TCU Error Control register. Use this register to enable fault handling interrupts.

The TCU_ERRCTLR characteristics are:

Usage constraints

When TCU_SCR.NS_RAS = 0, Non-secure accesses to this register are RAZ/WI.

Configurations

This register exists in all MMU-600 configurations.

Attributes

Offset `0x08E88`

Type RW

Reset `0x00000000`

Width 32

The following figure shows the bit assignments.

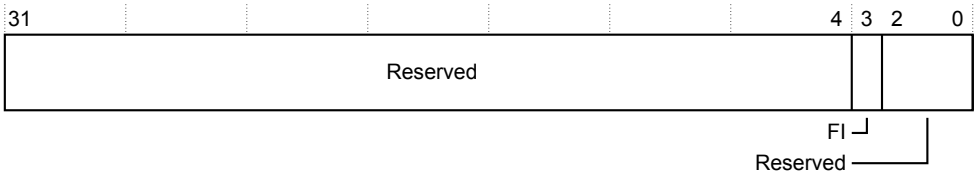

Figure 3-9 TCU_ERRCTLR register bit assignments

The following table shows the bit assignments.

Table 3-23 TCU_ERRCTLR register bit assignments

Bits	Name	Function
[31:4]	-	Reserved
[3]	FI	Enable fault handling interrupts: **0** An interrupt is generated on **ras_irpt** when a fault occurs. **1** No interrupt is generated when a fault occurs.
[2:0]	-	Reserved

3.8.3 TCU_ERRSTATUS

This is the TCU Error Record Primary Syndrome register. Use this register to find out whether different types of error have occurred on the TCU.

The TCU_ERRSTATUS characteristics are:

Usage constraints

When TCU_SCR.NS_RAS = 0, Non-secure accesses to this register are RAZ/WI.

To prevent race conditions, under certain circumstances, writes to some bits in this register are ignored. Typically, these writes are ignored when software has not yet observed a new error.

Configurations

This register exists in all TCU configurations.

Attributes

Offset `0x08E90`

Type RW

Reset 0x00000000

Width 32

The following figure shows the bit assignments.

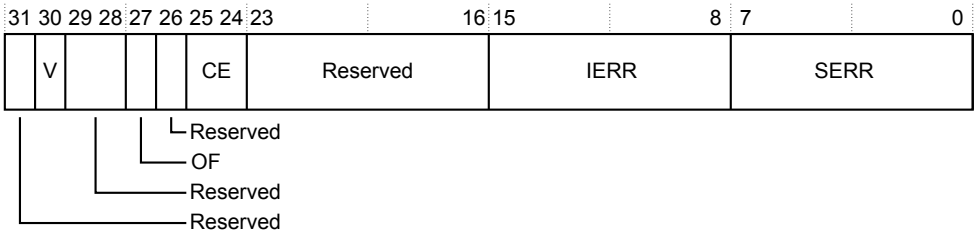

Figure 3-10 TCU_ERRSTATUS register bit assignments

The following table shows the bit assignments.

Table 3-24 TCU_ERRSTATUS register bit assignments

Bits	Name	Function
[31]	-	Reserved
[30]	V	Register valid. This bit is set to 1 to indicate that at least one RAS error was recorded. Clear this bit by writing a 1 to it. If CE is not 0b00 and is not being cleared, the write is ignored. A write of 0 is ignored.
[29:28]	-	Reserved
[27]	OF	Overflow. This bit is set to 1 to indicate that multiple correctable errors were recorded. That is, at least one correctable error was recorded when CE != 0b00. Clear this bit by writing a 1 to it. A write of 0 is ignored.
[26]	-	Reserved
[25:24]	CE	Correctable Error. This field is set to 0b10 to indicate that a corrected error occurred. Clear this field by writing 0b11 to it. If OF is set to 1 and is not being cleared, the write is ignored. A write of any value other than 0b11 is ignored.
[23:16]	-	Reserved

Table 3-24 TCU_ERRSTATUS register bit assignments (continued)

Bits	Name	Function
[15:8]	IERR	IMPLEMENTATION DEFINED error code. When SERR is not set to 0, this field indicates the source of the error, as follows: `0x00` Stage 1, level 0 walk cache. `0x01` Stage 1, level 1 walk cache. `0x02` Stage 1, level 2 walk cache. `0x03` Stage 1, level 3 walk cache. `0x04` Stage 2, level 0 walk cache. `0x05` Stage 2, level 1 walk cache. `0x06` Stage 2, level 2 walk cache. `0x07` Stage 2, level 3 walk cache. `0x08` Configuration cache. Writes to this field are ignored.
[7:0]	SERR	Error code. This read-only field provides information about the earliest unacknowledged correctable error, as follows: `0x00` No error. This code occurs when CE = `0b00`. `0x07` Tag corrupted. This code can occur when CE != `0b00`. `0x08` Data corrupted. This code can occur when CE != `0b00`.

3.8.4 TCU_ERRGEN

This is the TCU Error Generation Register. Use this register to generate tag parity errors, for example when testing error-handling software.

The TCU_ERRGEN characteristics are:

Usage constraints

When TCU_SCR.NS_RAS = 0, Non-secure accesses to this register are RAZ/WI.

Configurations

This register exists in all TCU configurations.

Attributes

Offset `0x08EC0`

Type RW

Reset `0x00000000`

Width 64

The following figure shows the bit assignments.

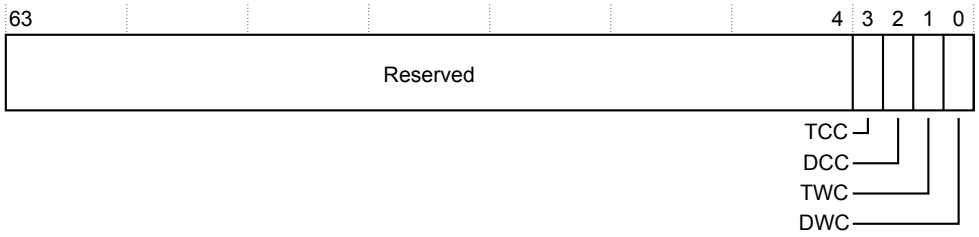

Figure 3-11 TCU_ERRGEN register bit assignments

The following table shows the bit assignments.

Table 3-25 TCU_ERRGEN register bit assignments

Bits	Name	Function
[63:4]	-	Reserved.
[3]	TCC	Configuration cache tag parity error. **0** No tag parity error is written to the configuration cache. **1** Entries that are written to the configuration cache include a tag parity error. A fault occurs when the entry is used.
[2]	DCC	Configuration cache data parity error. **0** No data parity error is written to the configuration cache. **1** Entries that are written to the configuration cache include a data parity error. A fault occurs when the entry is used. **Note** Tag parity errors mask data parity errors. When testing data parity error functionality, ensure that software does not set this bit and the TCC bit at the same time.
[1]	TWC	Walk cache tag parity error. **0** No tag parity error is written to the walk cache. **1** Entries that are written to the walk cache include a tag parity error. A fault occurs when the entry is used.
[0]	DWC	Walk cache data parity error. **0** No data parity error is written to the walk cache. **1** Entries that are written to the walk cache include a data parity error. A fault occurs when the entry is used. **Note** Tag parity errors mask data parity errors. When testing data parity error functionality, ensure that software does not set this bit and the TWC bit at the same time.

3.9 TBU Component and Peripheral ID Registers

The component and peripheral identity registers comply with the format that Arm CoreLink and CoreSight components use, and that the SMMUv3 architecture recommends. They provide key information about the MMU-600 hardware, including the product and associated revision number. They also identify Arm as the designer of the SMMU.

These registers are all read-only. Each field defines a single byte in the least significant 8 bits, and the most significant 24 bits are reserved. The least significant 8 bits of the four Component ID registers form a single 32-bit conceptual ID register. In a similar way, the defined fields of the eight Peripheral ID registers form a conceptual 64-bit ID register.

Table 3-26 TBU Component and Peripheral ID registers bit assignments

Register	Offset	Bits	Value	Function
SMMU_PIDR4	0x00FD0	[7:4]	0x0	4KB region count.
		[3:0]	0x4	JEP106 continuation code for Arm.
SMMU_PIDR5	0x00FD4	[7:0]	0x00	Reserved.
SMMU_PIDR6	0x00FD8	[7:0]	0x00	Reserved.
SMMU_PIDR7	0x00FDC	[7:0]	0x00	Reserved.
SMMU_PIDR0	0x00FE0	[7:0]	0x84	Part number[7:0].
SMMU_PIDR1	0x00FE4	[7:4]	0xB	JEP106 ID code[3:0] for Arm.
		[3:0]	0x4	Part number[11:8].
SMMU_PIDR2	0x00FE8	[7:4]	0x1	MMU-600 major revision. The value 0x1 indicates major product revision r1.
		[3]	0x1	The component uses a manufacturer identity code that JEDEC allocates, according to the JEP106 specification.
		[2:0]	0x3	JEP106 ID code[6:4] for Arm.
SMMU_PIDR3	0x00FEC	[7:4]	MAX[0x0,ecorevnum]	MMU-600 minor revision. The value 0x0 indicates minor product revision p0.
		[3:0]	0x0	CMOD. This field is not used.
SMMU_CIDR0	0x00FF0	[7:0]	0x0D	Preamble.
SMMU_CIDR1	0x00FF4	[7:0]	0xF0	
SMMU_CIDR2	0x00FF8	[7:0]	0x05	
SMMU_CIDR3	0x00FFC	[7:0]	0xB1	

3.10 TBU PMU Component and Peripheral ID Registers

The component and peripheral identity registers comply with the format used by Arm CoreLink and CoreSight components and recommended in the SMMUv3 architecture. They provide key information about the MMU-600 hardware, including the product and associated revision number. They also identify Arm as the designer of the SMMU.

These registers are all read-only. Each field defines a single byte in the least significant 8 bits, and the most significant 24 bits are reserved. The least significant 8 bits of the four Component ID registers form a single 32-bit conceptual ID register. In a similar way, the defined fields of the eight Peripheral ID registers form a conceptual 64-bit ID register.

Table 3-27 TBU PMU Component and Peripheral ID registers bit assignments

Register	Offset	Bits	Value	Function
SMMU_PMCG_PMAUTHSTATUS	0x02FB8	[7:0]	0x00	No authentication interface is implemented.
SMMU_PMCG_PIDR4	0x02FD0	[7:4]	0x0	4KB region count.
		[3:0]	0x4	JEP106 continuation code for Arm.
SMMU_PMCG_PIDR5	0x02FD4	[7:0]	0x00	Reserved.
SMMU_PMCG_PIDR6	0x02FD8	[7:0]	0x00	Reserved.
SMMU_PMCG_PIDR7	0x02FDC	[7:0]	0x00	Reserved.
SMMU_PMCG_PIDR0	0x02FE0	[7:0]	0x83	Part number[7:0].
SMMU_PMCG_PIDR1	0x02FE4	[7:4]	0xB	JEP106 ID code[3:0] for Arm.
		[3:0]	0x4	Part number[11:8].
SMMU_PMCG_PIDR2	0x02FE8	[7:4]	0x1	MMU-600 major revision. The value 0x1 indicates major product revision r1.
		[3]	0x1	The component uses a manufacturer identity code that JEDEC allocates, according to the JEP106 specification.
		[2:0]	0x3	JEP106 ID code[6:4] for Arm.
SMMU_PMCG_PIDR3	0x02FEC	[7:4]	MAX[0x0,ecorevnum]	MMU-600 minor revision. The value 0x0 indicates minor product revision p0.
		[3:0]	0x0	CMOD. This field is not used.
SMMU_PMCG_CIDR0	0x02FF0	[7:0]	0x0D	Preamble.
SMMU_PMCG_CIDR1	0x02FF4	[7:0]	0x90	
SMMU_PMCG_CIDR2	0x02FF8	[7:0]	0x05	
SMMU_PMCG_CIDR3	0x02FFC	[7:0]	0xB1	

3.11 TBU microarchitectural registers

You can set the TBU microarchitectural registers at boot time to optimize TBU behavior for your system. Arm recommends the default settings for most systems.

This section contains the following subsections:

3.11.1 TBU_CTRL

This register disables TBU features. Do not modify the bits in this register unless directed to do so by Arm.

Its characteristics are:

Usage constraints

There are no usage constraints.

Configurations

This register exists in all TBU configurations.

Attributes

Offset `0x08E00`

Type RW

Reset `0x00000000`

Width 32

The following figure shows the bit assignments.

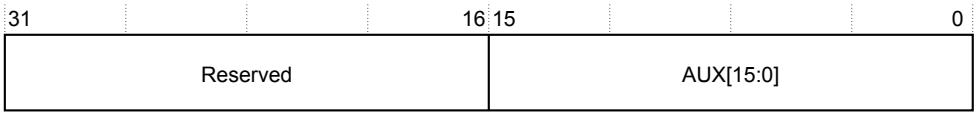

Figure 3-12 TBU_CTRL register bit assignments

The following table shows the bit assignments.

Table 3-28 TBU_CTRL register bit assignments

Bits	Name	Function
[31:16]	-	Reserved.
[15:0]	AUX[15:0]	Leave each of these bits as 0.

3.11.2 TBU_SCR

The TBU Secure Control register controls whether Non-secure software is permitted to access the TBU registers.

Its characteristics are:

Usage constraints

This register is accessible only by Secure software. Non-secure accesses to this register are RAZ/WI.

Configurations

This register exists in all TBU configurations.

Attributes

Offset `0x08E18`

Type RW

Reset `0x00000000`

Width 32

The following figure shows the bit assignments.

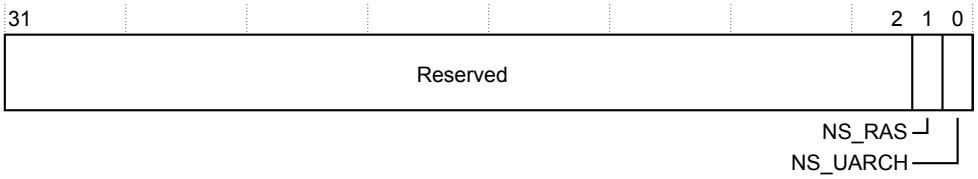

Figure 3-13 TBU_SCR register bit assignments

The following table shows the bit assignments.

Table 3-29 TBU_SCR register bit assignments

Bits	Name	Function
[31:2]	-	Reserved.
[1]	NS_RAS	Non-secure register access to RAS registers. `0` When this bit is set to 0, Non-secure accesses to register addresses `0x08E80`–`0x08EC0` are RAZ/WI. `1` When this bit is set to 1, Non-secure access to RAS registers is permitted.
[0]	NS_UARCH	Non-secure register access to TBU_CTRL. `0` When this bit is set to 0, Non-secure accesses to TBU_CTRL are RAZ/WI. `1` When this bit is set to 1, Non-secure accesses to TBU_CTRL are permitted.

3.12 TBU RAS registers

The MMU-600 includes TBU registers that are related to *Reliability, Availability, and Serviceability* (RAS).

This section contains the following subsections:

3.12.1 TBU_ERRFR

This is the TBU Error Feature register. Use this register to discover how the TBU handles errors.

The TBU_ERRFR characteristics are:

Usage constraints

This register is read-only. When TBU_SCR.NS_RAS = 0, Non-secure accesses to this register are RAZ.

Configurations

This register exists in all TBU configurations.

Attributes

Offset `0x08E80`

Type RO

Reset `0x00000081`

Width 32

The following figure shows the bit assignments.

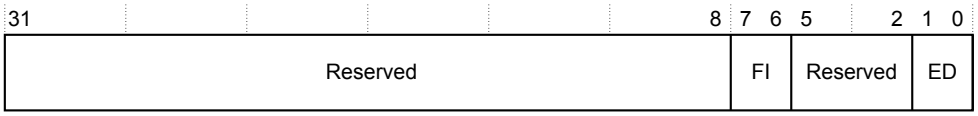

Figure 3-14 TBU_ERRFR register bit assignments

The following table shows the bit assignments.

Table 3-30 TBU_ERRFR register bit assignments

Bits	Name	Function
[31:8]	-	Reserved
[7:6]	FI	The value `0b10` indicates that the fault handling interrupt is controllable.
[5:2]	-	Reserved
[1:0]	ED	The value `0b01` indicates that TBU error detection is always enabled.

3.12.2 TBU_ERRCTLR

This is the TBU Error Control register. Use this register to enable fault handling interrupts.

The TBU_ERRCTLR characteristics are:

Usage constraints

When TBU_SCR.NS_RAS = 0, Non-secure accesses to this register are RAZ/WI.

Configurations

This register exists in all MMU-600 configurations. An instance of this register exists for each implemented TBU.

Attributes

Offset 0x08E88

Type RW

Reset 0x00000000

Width 32

The following figure shows the bit assignments.

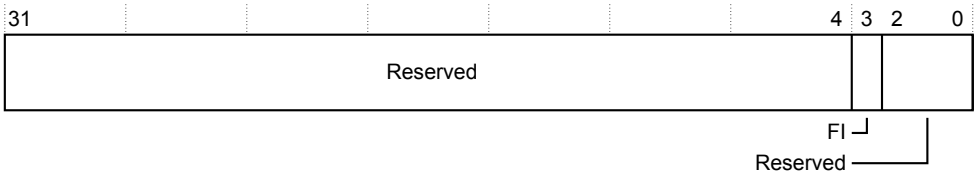

Figure 3-15 TBU_ERRCTLR register bit assignments

The following table shows the bit assignments.

Table 3-31 TBU_ERRCTLR register bit assignments

Bits	Name	Function
[31:4]	-	Reserved
[3]	FI	Set this bit to 1 to enable fault handling interrupts for the TBU.
[2:0]	-	Reserved

3.12.3 TBU_ERRSTATUS

This is the TBU Error Record Primary Syndrome register. Use this register to find out whether different types of error have occurred on the TBU.

The TBU_ERRSTATUS characteristics are:

Usage constraints

When TBU_SCR.NS_RAS = 0, Non-secure accesses to this register are RAZ/WI. To prevent race conditions, under certain circumstances, writes to some bits in this register are ignored. Typically, these writes are ignored when a new error has not yet been observed by software.

Configurations

This register exists in all MMU-600 configurations. An instance of this register exists for each implemented TBU.

Attributes

Offset 0x08E90

Type RW

Reset 0x00000000

Width 32

The following figure shows the bit assignments.

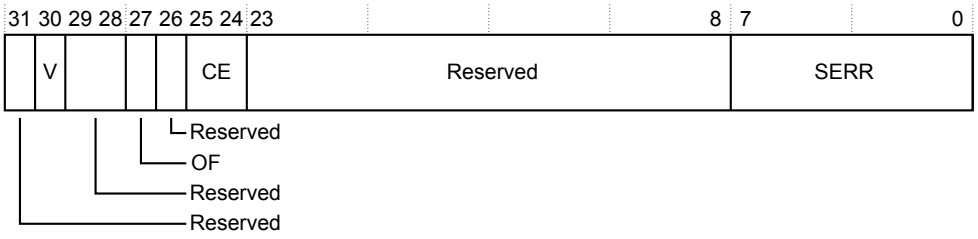

Figure 3-16 TBU_ERRSTATUS register bit assignments

The following table shows the bit assignments.

Table 3-32 TBU_ERRSTATUS register bit assignments

Bits	Name	Function
[31]	-	Reserved
[30]	V	Register valid. This bit is set to 1 to indicate that at least one RAS error was recorded. Clear this bit by writing a 1 to it. If CE is not `0b00` and is not being cleared, the write is ignored. A write of 0 is ignored.
[29:28]	-	Reserved
[27]	OF	Overflow. This bit is set to 1 to indicate that multiple correctable errors were recorded. That is, at least one correctable error was recorded when CE != `0b00`. Clear this bit by writing a 1 to it. A write of 0 is ignored.
[26]	-	Reserved
[25:24]	CE	Correctable Error. This field is set to `0b10` to indicate that a corrected error occurred. Clear this field by writing `0b11` to it. If OF is set to 1 and is not being cleared, the write is ignored. A write of any value other than `0b11` is ignored.
[23:8]	-	Reserved
[7:0]	SERR	Error code. This field provides information about the earliest unacknowledged correctable error, as follows: `0x00` No error. This code occurs when CE = `0b00`. `0x07` Main TLB tag is corrupted. This code can occur when CE != `0b00`. `0x08` Main TLB data is corrupted. This code can occur when CE != `0b00`. Writes to this field are ignored.

3.12.4 TBU_ERRGEN

This is the TBU Error Generation register. Use this register to generate tag parity errors. You might want to generate errors in certain cases, such as when testing error-handling software.

The TBU_ERRGEN characteristics are:

Usage constraints

When TBU_SCR.NS_RAS = 0, Non-secure accesses to this register are RAZ/WI.

Configurations

This register exists in all TBU configurations.

Attributes

Offset 0x08EC0

Type RW

Reset 0x00000000

Width 64

The following figure shows the bit assignments.

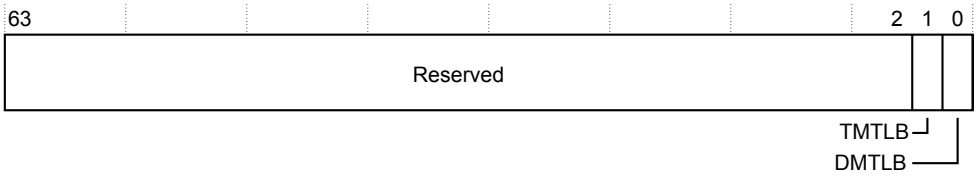

Figure 3-17 TBU_ERRGEN register bit assignments

The following table shows the bit assignments.

Table 3-33 TBU_ERRGEN register bit assignments

Bits	Name	Function
[63:2]	-	Reserved.
[1]	TMTLB	Main TLB tag parity error. **0** No tag parity error is written to the Main TLB. **1** Entries that are written to the Main TLB include a tag parity error. A fault occurs when the entry is used.
[0]	DMTLB	Main TLB data parity error. **0** No data parity error is written to the Main TLB. **1** Entries that are written to the Main TLB include a data parity error. A fault occurs when the entry is used. **Note** Tag parity errors mask data parity errors. When testing data parity error functionality, ensure that software does not set this bit and the TMTLB bit at the same time.

Appendix A
Signal descriptions

This appendix describes the MMU-600 external signals.

It contains the following sections:

A.1 Clock and reset signals

The MMU-600 uses a single set of standard clock and reset signals.

The following table shows the clock and reset signals.

Table A-1 Clock and reset signals

Signal	Direction	Description
aclk	Input	Global clock.
aresetn	Input	Global reset.

A.2 TCU QTW/DVM interface signals

The TCU QTW/DVM interface signals are based on the AMBA ACE5-Lite signals. See the *Arm® AMBA® AXI and ACE Protocol Specification, AXI3, AXI4, AXI5, ACE and ACE5* for more information about these signals.

The following table shows the TCU QTW/DVM interface signals.

Table A-2 TCU QTW/DVM interface signals

Signal	Direction	Description
acaddr_qtw	Input	Snoop address.
acprot_qtw	Input	Snoop protection type.
acready_qtw	Output	Snoop address ready.
acsnoop_qtw	Input	Snoop transaction type.
acvalid_qtw	Input	Snoop address valid.
arid_qtw	Output	Read address ID.
araddr_qtw	Output	Read address.
arburst_qtw	Output	Burst type.
arcache_qtw	Output	Memory type.
ardomain_qtw	Output	Shareability domain.
arlen_qtw	Output	Burst length.
arlock_qtw	Output	Lock type.
arprot_qtw	Output	Protection type.
arqos_qtw	Output	QoS identifier.
arready_qtw	Input	Read address ready.
arregion_qtw	Output	Region identifier.
arsize_qtw	Output	Burst size.
arsnoop_qtw	Output	Transaction type.
arvalid_qtw	Output	Read address valid.
awid_qtw	Output	Write address ID.
awaddr_qtw	Output	Write address.
awburst_qtw	Output	Burst type.
awcache_qtw	Output	Memory type.

Table A-2 TCU QTW/DVM interface signals (continued)

Signal	Direction	Description
awdomain_qtw	Output	Shareability domain.
awlen_qtw	Output	Burst length.
awlock_qtw	Output	Lock type.
awprot_qtw	Output	Protection type.
awqos_qtw	Output	QoS identifier.
awready_qtw	Input	Write address ready.
awregion_qtw	Output	Region identifier.
awsize_qtw	Output	Burst size.
awsnoop_qtw	Output	Transaction type.
awvalid_qtw	Output	Write address valid.
crready_qtw	Input	Snoop response ready.
crresp_qtw	Output	Snoop response.
crvalid_qtw	Output	Snoop response valid.
rid_qtw	Input	Read data ID.
rdata_qtw	Input	Read data.
rlast_qtw	Input	Read last.
rready_qtw	Output	Read ready.
rresp_qtw	Input	Read response.
rvalid_qtw	Input	Read valid.
wdata_qtw	Output	Write data.
wlast_qtw	Output	Write last.
wready_qtw	Input	Write ready.
wstrb_qtw	Output	Write strobe.
wvalid_qtw	Output	Write valid.
bid_qtw	Input	Response ID.
bready_qtw	Output	Response ready.
bresp_qtw	Input	Write response.

Table A-2 TCU QTW/DVM interface signals (continued)

Signal	Direction	Description
bvalid_qtw	Input	Write response valid.
awakeup_qtw	Output	Wakeup.
acwakeup_qtw	Input	Snoop wakeup.
acvmidext_qtw	Input	Snoop Extended *Virtual Machine IDentifier* (VMID).

A.3 TCU programming interface signals

The TCU programming interface signals are based on the AMBA APB4 signals. See the *Arm® AMBA® APB Protocol Specification* for more information about these signals.

The following table shows the TCU programming interface signals.

Table A-3 TCU programming interface signals

Signal	Direction	Description
paddr_prog	Input	Peripheral address.
psel_prog	Input	Peripheral select.
penable_prog	Input	Enable for transfer.
pwrite_prog	Input	Write transaction indicator.
pprot_prog	Input	Protection type.
pwdata_prog	Input	Write data.
pstrb_prog	Input	Write data strobe.
pslverr_prog	Output	Error response.
prdata_prog	Output	Read data.
pready_prog	Output	Transfer ready.
pwakeup_prog	Input	Interface wakeup.

A.4 TCU SYSCO interface signals

The following table shows the TCU SYSCO interface signals.

Table A-4 TCU SYSCO interface signals

Signal	Direction	Description
syscoreq	Output	System coherency request. This output transitions: **HIGH** To indicate that the master is requesting to enter the coherency domain. **LOW** To indicate that the master is requesting to exit the coherency domain.
syscoack	Input	System coherency acknowledge. This input transitions to the same level as **syscoreq** when the request to enter or exit the coherency domain is complete.

See the *Arm® AMBA® AXI and ACE Protocol Specification, AXI3, AXI4, AXI5, ACE and ACE5* for more information about these signals.

A.5 TCU PMU snapshot interface signals

The following table shows the TCU PMU snapshot interface signals.

Table A-5 TCU PMU snapshot interface signals

Signal	Direction	Description
pmusnapshot_req	Input	PMU snapshot request. The PMU snapshot occurs on the rising edge of **pmusnapshot_req**.
pmusnapshot_ack	Output	PMU snapshot acknowledge. The TCU uses this signal to acknowledge that the PMU snapshot has occurred. This signal is LOW after reset.

A.6 TCU LPI_PD interface signals

The following table shows the TCU LPI_PD interface signals.

Table A-6 TCU LPI_PD interface signals

Signal	Direction	Description
qactive_pd	Output	Component active.
qreqn_pd	Input	Quiescence request.
qacceptn_pd	Output	Quiescence accept.
qdeny_pd	Output	Quiescence deny.

See the *AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces* for more information about these signals.

A.7 TCU LPI_CG interface signals

The following table shows the TCU LPI_CG interface signals.

Table A-7 TCU LPI_CG interface signals

Signal	Direction	Description
qactive_cg	Output	Component active.
qreqn_cg	Input	Quiescence request.
qacceptn_cg	Output	Quiescence accept.
qdeny_cg	Output	Quiescence deny.

See the *AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces* for more information about these signals.

A.8 TCU DTI interface signals

The following table shows the TCU DTI interface signals.

Table A-8 TCU DTI interface signals

Signal	Direction	Description
tvalid_dti_dn	Master to slave.	Flow control signal.
tready_dti_dn	Slave to master.	Flow control signal.
tdata_dti_dn	Master to slave.	Message data signal.
tid_dti_dn	Master to slave.	Identifies the master that initiated the message.
tlast_dti_dn	Master to slave.	Indicates the last cycle of a message.
tkeep_dti_dn	Master to slave.	This signal indicates valid bytes.
tvalid_dti_up	Slave to master.	Flow control signal.
tready_dti_up	Master to slave.	Flow control signal.
tdata_dti_up	Slave to master.	Message data signal.
tdest_dti_up	Slave to master.	Identifies the master that is receiving the message.
tlast_dti_up	Slave to master.	Indicates the last cycle of a message.
tkeep_dti_up	Slave to master.	Indicates valid bytes.
twakeup_dti_up	Slave to master.	Wakeup signal.
twakeup_dti_dn	Master to slave.	Wakeup signal.

See the *Arm® AMBA® 4 AXI4-Stream Protocol Specification* for more information about the DTI signals.

See the *Arm® AMBA® Distributed Translation Interface (DTI) Protocol Specification* for more information about DTI protocol messages.

A.9 TCU interrupt signals

The TCU interrupt signals are edge-triggered. The interrupt controller must detect the rising edge of these signals.

The TCU can also output the Secure and Non-secure Event queue, SYNC complete commands, and global interrupts as *Message Signaled Interrupts* (MSIs) on the QTW/DVM interface. If the system supports capturing MSIs from the TCU, there is no requirement to connect the corresponding interrupt signals in this interface.

The following table shows the TCU interrupt signals.

Table A-9 TCU interrupt interface signals

Signal	Direction	Description
event_q_irpt_s	Output	Event queue, Secure interrupt. Asserts a Secure interrupt to indicate that the Event queue is not empty or has overflowed.
event_q_irpt_ns	Output	Event queue, Non-secure interrupt. Asserts a Non-secure interrupt to indicate that the Event queue is not empty or has overflowed.
cmd_sync_irpt_ns	Output	SYNC complete, Non-secure interrupt. Asserts a Non-secure interrupt to indicate that the `CMD_SYNC` command is complete.
cmd_sync_irpt_s	Output	SYNC complete, Secure interrupt. Asserts a Secure interrupt to indicate that the `CMD_SYNC` command is complete.
global_irpt_ns	Output	Asserts a global Non-secure interrupt.
global_irpt_s	Output	Asserts a global Secure interrupt.
ras_irpt	Output	Asserts a *Reliability, Availability, and Serviceability* (RAS) interrupt. ——— **Note** ——— The MMU-600 cannot output RAS interrupts as MSIs. This output must be connected to an interrupt controller. ———
pmu_irpt	Output	Asserts a PMU interrupt. ——— **Note** ——— The MMU-600 cannot output PMU interrupts as MSIs. This output must be connected to an interrupt controller. ———
evento	Output	Event output for connection to processors. This signal is asserted for one cycle to indicate an event that enables processors to wake up from WFE low-power state.
pri_q_irpt_ns	Output	Asserts a *Page Request Interface* (PRI) queue interrupt.

A.10 TCU tie-off signals

The TCU tie-off signals are sampled between exiting reset and the LPI_PD interface first entering the Q_RUN state. Ensure that the value of these signals does not change when the LPI_PD interface is in the Q_STOPPED or Q_EXIT state for the first time after exiting reset.

The following table shows the TCU tie-off signals.

Table A-10 TCU tie-off signals

Signal	Direction	Description
sup_cohacc	Input	This signal indicates whether the QTW interface is I/O-coherent. Tie HIGH when the TCU is connected to a coherent interconnect.
sup_btm	Input	This signal indicates whether the Broadcast TLB Maintenance is supported. Tie HIGH when the TCU is connected to an interconnect that supports DVM.
sup_sev	Input	This signal indicates whether the Send Event mechanism is supported. Tie HIGH when **evento** is connected.
sup_oas[2:0]	Input	Output address size supported. The encodings for this input are: `0b000` 32 bits. `0b001` 36 bits. `0b010` 40 bits. `0b011` 42 bits. `0b100` 44 bits. `0b101` 48 bits. You must not use other encodings, including `0b110` that SMMUv3.1 defines to indicate 52-bit addresses. They are treated as `0b101`.
sec_override	Input	When HIGH, certain registers are accessible to Non-secure accesses from reset, as the TCU_SCR register settings describe.
ecorevnum[3:0]	Input	Tie this signal to 0 unless directed otherwise by Arm.

See the *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3.0* for more information about the SMMUv3 ID signals.

A.11 TCU and TBU test and debug signals

The test and debug signals are common to the TCU and TBU.

The following table shows the test and debug signals.

Table A-11 Test and debug signals

Signal	Direction	Description
dftcgen	Input	Clock gate enable. To enable architectural clock gates for the **aclk** clock, set this signal HIGH during scan shift.
dftrstdisable	Input	Reset disable. To disable reset, set this signal HIGH during scan shift.
dftramhold	Input	Preserve RAM state. To preserve the state of the RAMs and their connected registers, set this signal HIGH during scan shift.
mbistresetn	Input	MBIST mode reset. This active-LOW signal is encoded as follows: **0** Reset MBIST functional logic. **1** Normal operation.
mbistreq	Input	MBIST test request. This signal is encoded as follows: **0** Normal operation. **1** Enable MBIST testing.

A.12 TBU TBS interface signals

The TBU TBS interface signals are based on the AMBA ACE5-Lite signals.

The following table shows the TBU TBS interface signals.

Table A-12 TBU TBS interface signals

Signal	Direction	Description
aclk	Input	Clock input.
acaddr_s[47:0]	Output	Snoop address. This signal is present for ACE TBU configurations only.
acprot_s[2:0]	Output	Snoop protection type. This signal is present for ACE TBU configurations only.
acready_s	Input	Snoop address ready. This signal is present for ACE TBU configurations only.
acsnoop_s[3:0]	Output	Snoop transaction type. This signal is present for ACE TBU configurations only.
acvalid_s	Output	Snoop address valid. This signal is present for ACE TBU configurations only.
acvmidext_s[3:0]	Output	Snoop address *Virtual Machine IDentifier* (VMID) Extension. This signal is present for ACE TBU configurations only.
acwakeup_s	Output	Wakeup signal. This signal is present for ACE TBU configurations only.
araddr_s	Input	Read address.
arburst_s	Input	Burst type.
arcache_s	Input	Memory type.
ardomain_s	Input	Shareability domain.
aresetn	Input	Active-LOW reset signal.
arid_s	Input	Read address ID.
arlen_s	Input	Burst length.
arlock_s	Input	Lock type.
arprot_s	Input	Protection type.
arqos_s	Input	*Quality of Service* (QoS).
arready_s	Output	Read address ready.

Table A-12 TBU TBS interface signals (continued)

Signal	Direction	Description
arregion_s	Input	Region identifier.
arsize_s	Input	Burst size.
armmussid_s	Input	These signals indicate the StreamID, SubstreamID, and ATS translated status of the originating transaction. These signals are defined by the AXI5 Untranslated_Transactions extension.
armmusid_s	Input	
armmussidv_s	Input	
armmusecsid_s	Input	
armmuatst_s	Input	
arvalid_s	Input	Read address valid.
awaddr_s	Input	Write address.
awburst_s	Input	Burst type.
awcache_s	Input	Memory type.
awdomain_s	Input	Shareability domain.
awid_s	Input	Write address ID.
awlen_s	Input	Burst length.
awlock_s	Input	Lock type.
awprot_s	Input	Protection type.
awqos_s	Input	QoS.
awready_s	Output	Write address ready.
awregion_s	Input	Region identifier.
awsize_s	Input	Burst size.
awmmussid_s	Input	These signals indicate the StreamID, SubstreamID, and ATS translated status of the originating transaction. These signals are defined by the AXI5 Untranslated_Transactions extension.
awmmusid_s		
awmmussidv_s		
awmmusecsid_s		
awmmuatst_s		
awvalid_s	Input	Write address valid.

Table A-12 TBU TBS interface signals (continued)

Signal	Direction	Description
awunique_s	Input	Line is permitted to be held in a Unique state. This signal is present for ACE TBU configurations only.
bid_s	Output	Response ID.
bready_s	Input	Response ready.
bresp_s	Output	Write response.
bvalid_s	Output	Write response valid.
cddata_s[TBUCFG_DATA_WIDTH-1:0]	Output	Snoop data. This signal is present for ACE TBU configurations only.
cdlast_s	Output	Last data transfer of a snoop transaction. This signal is present for ACE TBU configurations only.
cdready_s	Input	Snoop data ready. This signal is present for ACE TBU configurations only.
cdvalid_s	Output	Snoop data valid. This signal is present for ACE TBU configurations only.
crready_s	Input	Snoop response ready. This signal is present for ACE TBU configurations only.
crresp_s[4:0]	Output	Snoop response. This signal is present for ACE TBU configurations only.
crvalid_s	Output	Snoop response valid. This signal is present for ACE TBU configurations only.
rack_s	Input	Read acknowledge. This signal is present for ACE TBU configurations only.
rdata_s	Output	Read data.
rid_s	Output	Read ID.
rlast_s	Output	Read last.
rready_s	Input	Read ready.
rresp_s[3:2]	Output	Read response. This signal is present for ACE TBU configurations only.
rvalid_s	Output	Read valid.

Table A-12 TBU TBS interface signals (continued)

Signal	Direction	Description
wack_s	Input	Write acknowledge. This signal is present for ACE TBU configurations only.
wdata_s	Input	Write data.
wlast_s	Input	Write last.
wready_s	Output	Write ready.
wstrb_s	Input	Write strobes.
wvalid_s	Input	Write valid.
aruser_s	Input	Read address (AR) channel user signal.
awuser_s	Input	Write address (AW) channel user signal.
wuser_s	Input	Write data (W) channel user signal.
ruser_s	Output	Read data (R) channel user signal.
buser_s	Output	Write response (B) channel user signal.
awakeup_s	Input	Wakeup signal.
arsnoop_s	Input	Transaction type of read transaction. This signal is not present for ACE TBU configurations.
awsnoop_s[3]	Input	Transaction type of write transaction.
awstashnid_s[10:0]	Input	These signals are defined by the AXI5 Cache_Stash_Transactions extension. If TBUCFG_STASH = 0, these signals are ignored. These signals are not present for ACE TBU configurations.
awstashniden_s	Input	
awstashlpid_s[4:0]	Input	
awstashlpiden_s	Input	

A.13 TBU TBM interface signals

The TBU TBM interface signals are based on the AMBA ACE5-Lite signals.

The following table shows the TBU TBM interface signals.

Table A-13 TBU TBM interface signals

Signal	Direction	Description
aclk	Input	Clock input.
acaddr_m[47:0]	Input	Snoop address. This signal is present for ACE TBU configurations only.
acprot_m[2:0]	Input	Snoop protection type. This signal is present for ACE TBU configurations only.
acready_m	Output	Snoop address ready. This signal is present for ACE TBU configurations only.
acsnoop_m[3:0]	Input	Snoop transaction type. This signal is present for ACE TBU configurations only.
acvalid_m	Input	Snoop address valid. This signal is present for ACE TBU configurations only.
acvmidext_m[3:0]	Input	Snoop address *Virtual Machine IDentifier* (VMID) Extension. This signal is present for ACE TBU configurations only.
acwakeup_m	Input	Wakeup signal. This signal is present for ACE TBU configurations only.
araddr_m	Output	Read address.
arburst_m	Output	Burst type.
arcache_m	Output	Memory type.
ardomain_m	Output	Shareability domain.
aresetn	Input	Active-LOW reset signal.
arid_m	Output	Read address ID.
arlen_m	Output	Burst length.
arlock_m	Output	Lock type.
arprot_m	Output	Protection type.
arqos_m	Output	*Quality of Service* (QoS).
arready_m	Input	Read address ready.

Table A-13 TBU TBM interface signals (continued)

Signal	Direction	Description
arregion_m	Output	Region identifier.
arsize_m	Output	Burst size.
armmusid_m	Output	These signals indicate the StreamID of the originating transaction.
armmusecsid_m	Output	
arvalid_m	Output	Read address valid.
awaddr_m	Output	Write address.
awburst_m	Output	Burst type.
awcache_m	Output	Memory type.
awdomain_m	Output	Shareability domain.
awid_m	Output	Write address ID.
awlen_m	Output	Burst length.
awlock_m	Output	Lock type.
awprot_m	Output	Protection type.
awqos_m	Output	QoS.
awready_m	Input	Write address ready.
awregion_m	Output	Region identifier.
awsize_m	Output	Burst size.
awmmusid_m	Output	These signals indicate the StreamID of the originating transaction. The *Generic Interrupt Controller* (GIC) uses these signals to determine the DeviceID of MSIs that originate from upstream masters.
awmmusecsid_m	Output	
awvalid_m	Output	Write address valid.
awunique_m	Output	Line is permitted to be held in a Unique state. This signal is present for ACE TBU configurations only.
bid_m	Input	Response ID.
bready_m	Output	Response ready.
bresp_m	Input	Write response.
bvalid_m	Input	Write response valid.

Table A-13 TBU TBM interface signals (continued)

Signal	Direction	Description
rack_m	Output	Read acknowledge. This signal is present for ACE TBU configurations only.
rdata_m	Input	Read data.
rid_m	Input	Read ID.
rlast_m	Input	Read last.
rready_m	Output	Read ready.
rresp_m[3:2]	Input	Read response. This signal is present for ACE TBU configurations only.
rvalid_m	Input	Read valid.
wack_m	Output	Write acknowledge. This signal is present for ACE TBU configurations only.
wdata_m	Output	Write data.
wlast_m	Output	Write last.
wready_m	Input	Write ready.
wstrb_m	Output	Write strobes.
wvalid_m	Output	Write valid.
aruser_m	Output	Read address (AR) channel user signal.
awuser_m	Output	Write address (AW) channel user signal.
wuser_m	Output	Write data (W) channel user signal.
ruser_m	Input	Read data (R) channel user signal.
buser_m	Input	Write response (B) channel user signal.
awakeup_m	Output	Wakeup signal.
arsnoop_m	Output	Transaction type of read transaction.
awsnoop_m[3]	Output	Transaction type of write transaction. This signal is not present for ACE TBU configurations.

Table A-13 TBU TBM interface signals (continued)

Signal	Direction	Description
awstashnid_m[10:0]	Output	These signals are defined by the AXI5 Cache_Stash_Transactions extension. If TBUCFG_STASH = 0, these signals are ignored. These signals are not present for ACE TBU configurations.
awstashniden_m	Output	
awstashlpid_m[4:0]	Output	
awstashlpiden_m	Output	
cddata_m[TBUCFG_DATA_WIDTH-1:0]	Input	Snoop data. This signal is present for ACE TBU configurations only.
cdlast_m	Input	Last data transfer of a snoop transaction. This signal is present for ACE TBU configurations only.
cdready_m	Output	Snoop data ready. This signal is present for ACE TBU configurations only.
cdvalid_m	Input	Snoop data valid. This signal is present for ACE TBU configurations only.
crready_m	Output	Snoop response ready. This signal is present for ACE TBU configurations only.
crresp_m[4:0]	Input	Snoop response. This signal is present for ACE TBU configurations only.
crvalid	Input	Snoop response valid. This signal is present for ACE TBU configurations only.

A.14 TBU PMU snapshot interface signals

The following table shows the TBU PMU snapshot interface signals.

Table A-14 TBU PMU snapshot interface signals

Signal	Direction	Description
pmusnapshot_req	Input	PMU snapshot request. The PMU snapshot occurs on the rising edge of **pmusnapshot_req**.
pmusnapshot_ack	Output	PMU snapshot acknowledge. The TBU uses this signal to acknowledge that the PMU snapshot has occurred. This signal is LOW after reset.

A.15 TBU LPI_PD interface signals

The following table shows the TBU LPI_PD interface signals.

Table A-15 TBU LPI_PD interface signals

Signal	Direction	Description
qactive_pd	Output	Component active.
qreqn_pd	Input	Quiescence request.
qaccept_pd	Output	Quiescence accept.
qdeny_pd	Output	Quiescence deny.

See the *AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces* for more information about these signals.

A.16 TBU LPI_CG interface signals

The following table shows the TBU LPI_CG interface signals.

Table A-16 TBU LPI_CG interface signals

Signal	Direction	Description
qactive_cg	Output	Component active.
qreqn_cg	Input	Quiescence request.
qaccept_cg	Output	Quiescence accept.
qdeny_cg	Output	Quiescence deny.

See the *AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces* for more information about these signals.

A.17 TBU DTI interface signals

The following table shows the TBU DTI interface signals.

Table A-17 TBU DTI interface signals

Signal	Direction	Description
tvalid_dti_dn	Master to slave.	Flow control signal.
tready_dti_dn	Slave to master.	Flow control signal.
tdata_dti_dn	Master to slave.	Message data signal.
tlast_dti_dn	Master to slave.	Indicates the last cycle of a message.
tkeep_dti_dn	Master to slave.	Indicates valid bytes.
tvalid_dti_up	Slave to master.	Flow control signal.
tready_dti_up	Master to slave.	Flow control signal.
tdata_dti_up	Slave to master.	Message data signal.
tlast_dti_up	Slave to master.	Indicates the last cycle of a message.
tkeep_dti_up	Slave to master.	Indicates valid bytes.
twakeup_dti_up	Slave to master.	Wakeup signal.
twakeup_dti_dn	Master to slave.	Wakeup signal.

See the *Arm® AMBA® 4 AXI4-Stream Protocol Specification* for more information about the DTI signals.

See the *Arm® AMBA® Distributed Translation Interface (DTI) Protocol Specification* for more information about DTI protocol messages.

A.18 TBU interrupt signals

The TBU interrupt signals are edge-triggered. The interrupt controller must detect the rising edge of these signals.

The MMU-600 TBU cannot output these interrupts as *Message Signaled Interrupts* (MSIs). These signals must be connected to an interrupt controller.

The following table shows the TBU interrupt signals.

Table A-18 TBU interrupt signals

Signal	Direction	Description
ras_irpt	Output	RAS interrupt.
pmu_irpt	Output	PMU interrupt.

A.19 TBU tie-off signals

The TBU tie-off signals are sampled between exiting reset and the LPI_PD interface first entering the Q_RUN state. Ensure that the value of these signals does not change when the LPI_PD interface is in the Q_STOPPED or Q_EXIT state for the first time after exiting reset.

The following table shows the TBU tie-off signals.

Table A-19 TBU tie-off signals

Signal	Direction	Description
ns_sid_high[23:TBUCFG_SID_WIDTH]	Input	Provides the high-order StreamID bits for all transactions with a Non-secure StreamID that pass through the TBU.
s_sid_high[23:TBUCFG_SID_WIDTH]	Input	Provides the high-order StreamID bits for all transactions with a Secure StreamID that pass through the TBU.
max_tok_trans[log2(TBUCFG_XLATE_SLOTS)-1:0]	Input	Indicates the number of DTI translation tokens to request when connecting to the TCU, minus 1.
pcie_mode	Input	You must tie this signal HIGH when the TBU is connected to a PCIe interface. When this signal is HIGH, the TBU behaves as if the PCIe 'No Snoop' property is applied to transactions downstream of the SMMU, as long as the PCIe interface outputs transactions with the following AXI memory types: • Normal Non-Cacheable Bufferable, when 'No Snoop' is set for the transaction. • Write-Back, when 'No Snoop' is not set for the transaction. This TBU behavior is a requirement of the *Arm Server Base System Architecture*. If this signal is HIGH, the attributes of TBS interface transactions are always combined with the translation attributes, even if stage 1 translation is enabled. That is, the transaction attributes are always calculated as if the DTI_TBU_TRANS_RESP.STRW field is EL1-S2, regardless of the actual STRW value. If this signal is HIGH, the input attribute and shareability override information in the ATTR_OVR field of the DTI_TBU_TRANS_RESP message is ignored. For SMMUv3, PCIe masters do not support this feature.
sec_override	Input	When HIGH, certain registers are accessible to Non-secure accesses from reset, as the TCU_SCR register settings describe.
ecorevnum[3:0]	Input	Tie this signal to 0 unless directed otherwise by Arm.

Table A-19 TBU tie-off signals (continued)

Signal	Direction	Description
utlb_roundrobin	Input	Defines the Micro TLB entry replacement policy. When LOW, the Micro TLB uses a *Pseudo Least Recently Used* (PLRU) replacement policy. This policy typically provides the best average performance. However, when multiple translations are prefetched using a StashTranslation transaction, they might evict each other. When HIGH, the Micro TLB uses a round-robin replacement policy. With this policy, you can prefetch multiple translations using a StashTranslation transaction without evictions occurring, as long as the Micro TLB size is not exceeded. Tie this signal HIGH if a real-time upstream master prefetches translations and you want to avoid transactions evicting each other. Otherwise, tie this signal LOW.
cmo_disable	Input	Tie this signal HIGH to disable cache maintenance operations. When this signal is HIGH, the following transactions are always aborted with an SLVERR response: • CleanInvalid. • CleanShared. • CleanSharedPersist. • MakeInvalid. Cache maintenance operations can sometimes break the requirements of limited sideband channel communication, such as when a master component accesses protected content. You can disable cache maintenance operations in such cases. Cache maintenance operations are always disabled for ACE interfaces. This signal is therefore not present when the connected interface is configured as an ACE interface. **Note** For ACE TBU configurations, this signal is not present, and is treated as 1.

Related information

3.7.5 TCU_SCR on page 3-80

A.20 DTI interconnect switch signals

The DTI interconnect switch provides signals for each of its interfaces.

The switch provides one DN_S*n* slave downstream interface per slave interface. The following table shows the DN_S*n* signals.

Table A-20 DTI interconnect switch DN_S*n* interface signals

Signal	Direction	Description
tvalid_dti_dn_s*n*	Slave to master.	Flow control signal.
tready_dti_dn_s*n*	Master to slave.	Flow control signal.
tdata_dti_dn_s*n*	Slave to master.	Message data signal.
tid_dti_dn_s*n*	Slave to master.	Indicates the master that initiated the message.
tlast_dti_dn_s*n*	Slave to master.	Indicates the last cycle of a message.
tkeep_dti_dn_s*n*	Slave to master.	Indicates valid bytes.
twakeup_dti_up_s*n*	Slave to master.	Wakeup signal.

The switch provides one UP_S*n* slave upstream interface per slave interface. The following table shows the UP_S*n* signals.

Table A-21 DTI interconnect switch UP_S*n* interface signals

Signal	Direction	Description
tvalid_dti_up_s*n*	Master to slave.	Flow control signal.
tready_dti_up_s*n*	Slave to master.	Flow control signal.
tdata_dti_up_s*n*	Master to slave.	Message data signal.
tdest_dti_up_s*n*	Master to slave.	Indicates the master that initiated the message.
tlast_dti_up_s*n*	Master to slave.	Indicates the last cycle of a message.
tkeep_dti_up_s*n*	Master to slave.	Indicates valid bytes.
twakeup_dti_up_s*n*	Master to slave.	Wakeup signal.

The switch provides a DN_M master downstream interface. The following table shows the DN_M signals.

Table A-22 DTI interconnect switch DN_M interface signals

Signal	Direction	Description
tvalid_dti_dn_m	Slave to master.	Flow control signal.
tready_dti_dn_m	Master to slave.	Flow control signal.

Table A-22 DTI interconnect switch DN_M interface signals (continued)

Signal	Direction	Description
tdata_dti_dn_m	Slave to master.	Message data signal.
tid_dti_dn_m	Slave to master.	Indicates the master that initiated the message.
tlast_dti_dn_m	Slave to master.	Indicates the last cycle of a message.
tkeep_dti_dn_m	Slave to master.	Indicates valid bytes.
twakeup_dti_dn_m	Slave to master.	Wakeup signal.

The switch provides an UP_M master upstream interface. The following table shows the UP_M signals.

Table A-23 DTI interconnect switch UP_M interface signals

Signal	Direction	Description
tvalid_dti_up_m	Master to slave.	Flow control signal.
tready_dti_up_m	Slave to master.	Flow control signal.
tdata_dti_up_m	Master to slave.	Message data signal.
tdest_dti_up_m	Master to slave.	Indicates the master that initiated the message.
tlast_dti_up_m	Master to slave.	Indicates the last cycle of a message.
tkeep_dti_up_m	Master to slave.	Indicates valid bytes.
twakeup_dti_up_m	Slave to master.	Wakeup signal.

A.21 DTI interconnect sizer signals

The DTI interconnect sizer provides signals for each of its interfaces.

The sizer provides an LPI_CG clock gating interface. The following table shows the LPI_CG signals.

Table A-24 DTI interconnect sizer LPI_CG interface signals

Signal	Direction	Description
qaccept_cg	Output.	Quiescence accept.
qactive_cg	Output.	Component active.
qdeny_cg	Output.	Quiescence deny.
qreqn_cg	Input.	Quiescence request.

The sizer provides a DN_S slave downstream interface. The following table shows the DN_S signals.

Table A-25 DTI interconnect sizer DN_S interface signals

Signal	Direction	Description
tvalid_dti_dn_s	Slave to master.	Flow control signal.
tready_dti_dn_s	Master to slave.	Flow control signal.
tdata_dti_dn_s	Slave to master.	Message data signal.
tid_dti_dn_s	Slave to master.	Indicates the master that initiated the message.
tlast_dti_dn_s	Slave to master.	Indicates the last cycle of a message.
tkeep_dti_dn_s	Slave to master.	Indicates valid bytes.
twakeup_dti_dn_s	Slave to master.	Wakeup signal.

The sizer provides an UP_S slave upstream interface. The following table shows the UP_S signals.

Table A-26 DTI interconnect sizer UP_S interface signals

Signal	Direction	Description
tvalid_dti_up_s	Master to slave.	Flow control signal.
tready_dti_up_s	Slave to master.	Flow control signal.
tdata_dti_up_s	Master to slave.	Message data signal.
tdest_dti_up_s	Master to slave.	Indicates the master that initiated the message.
tlast_dti_up_s	Master to slave.	Indicates the last cycle of a message.

Table A-26 DTI interconnect sizer UP_S interface signals (continued)

Signal	Direction	Description
tkeep_dti_up_s	Master to slave.	Indicates valid bytes.
twakeup_dti_up_s	Master to slave.	Wakeup signal.

The sizer provides a DN_M master downstream interface. The following table shows the DN_M signals.

Table A-27 DTI interconnect sizer DN_M interface signals

Signal	Direction	Description
tvalid_dti_dn_m	Slave to master.	Flow control signal.
tready_dti_dn_m	Master to slave.	Flow control signal.
tdata_dti_dn_m	Slave to master.	Message data signal.
tid_dti_dn_m	Slave to master.	Indicates the master that initiated the message.
tlast_dti_dn_m	Slave to master.	Indicates the last cycle of a message.
tkeep_dti_dn_m	Slave to master.	Indicates valid bytes.
twakeup_dti_dn_m	Slave to master.	Wakeup signal.

The sizer provides an UP_M master upstream interface. The following table shows the UP_M signals.

Table A-28 DTI interconnect sizer UP_M interface signals

Signal	Direction	Description
tvalid_dti_up_m	Master to slave.	Flow control signal.
tready_dti_up_m	Slave to master.	Flow control signal.
tdata_dti_up_m	Master to slave.	Message data signal.
tdest_dti_up_m	Master to slave.	Indicates the master that initiated the message.
tlast_dti_up_m	Master to slave.	Indicates the last cycle of a message.
tkeep_dti_up_m	Master to slave.	Indicates valid bytes.
twakeup_dti_up_m	Slave to master.	Wakeup signal.

A.22 DTI interconnect register slice signals

The DTI interconnect register slice provides signals for each of its interfaces.

The register slice provides an LPI_CG clock gating interface. The following table shows the LPI_CG signals.

Table A-29 DTI interconnect register slice LPI_CG interface signals

Signal	Direction	Description
qaccept_cg	Output.	Quiescence accept.
qactive_cg	Output.	Component active.
qdeny_cg	Output.	Quiescence deny.
qreqn_cg	Input.	Quiescence request.

The register slice provides a DN_S slave downstream interface. The following table shows the DN_S signals.

Table A-30 DTI interconnect register slice DN_S interface signals

Signal	Direction	Description
tvalid_dti_dn_s	Slave to master.	Flow control signal.
tready_dti_dn_s	Master to slave.	Flow control signal.
tdata_dti_dn_s	Slave to master.	Message data signal.
tid_dti_dn_s	Slave to master.	Indicates the master that initiated the message.
tlast_dti_dn_s	Slave to master.	Indicates the last cycle of a message.
tkeep_dti_dn_s	Slave to master.	Indicates valid bytes.

The register slice provides an UP_S slave upstream interface. The following table shows the UP_S signals.

Table A-31 DTI interconnect register slice UP_S interface signals

Signal	Direction	Description
tvalid_dti_up_s	Master to slave.	Flow control signal.
tready_dti_up_s	Slave to master.	Flow control signal.
tdata_dti_up_s	Master to slave.	Message data signal.
tdest_dti_up_s	Master to slave.	Indicates the master that initiated the message.
tlast_dti_up_s	Master to slave.	Indicates the last cycle of a message.
tkeep_dti_up_s	Master to slave.	Indicates valid bytes.

The register slice provides a DN_M master downstream interface. The following table shows the DN_M signals.

Table A-32 DTI interconnect register slice DN_M interface signals

Signal	Direction	Description
tvalid_dti_dn_m	Slave to master.	Flow control signal.
tready_dti_dn_m	Master to slave.	Flow control signal.
tdata_dti_dn_m	Slave to master.	Message data signal.
tid_dti_dn_m	Slave to master.	Indicates the master that initiated the message.
tlast_dti_dn_m	Slave to master.	Indicates the last cycle of a message.
tkeep_dti_dn_m	Slave to master.	Indicates valid bytes.

The register slice provides an UP_M master upstream interface. The following table shows the UP_M signals.

Table A-33 DTI interconnect register slice UP_M interface signals

Signal	Direction	Description
tvalid_dti_up_m	Master to slave.	Flow control signal.
tready_dti_up_m	Slave to master.	Flow control signal.
tdata_dti_up_m	Master to slave.	Message data signal.
tdest_dti_up_m	Master to slave.	Indicates the master that initiated the message.
tlast_dti_up_m	Master to slave.	Indicates the last cycle of a message.
tkeep_dti_up_m	Master to slave.	Indicates valid bytes.

Appendix B
Software initialization examples

This appendix provides examples of how software can initialize and enable the MMU-600.

It contains the following sections:

B.1 Initializing the SMMU

Software must initialize the MMU-600 before you can use it.

The MMU-600 supports Secure and Non-secure translation worlds. This section defines how to initialize Non-secure translation. The procedures for initializing Secure translation are similar, and require you to access the corresponding MMU-600 Secure registers.

Note

This section does not describe how to create translation tables. See the *Arm® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile* for more information.

See the *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3.0 and version 3.1* for more information about MMU-600 initialization.

This section contains the following subsections:

B.1.1 Allocating the Command queue

The MMU-600 uses the Command queue to receive commands. Software must allocate memory for the Command queue and configure the appropriate registers in the SMMU.

To allocate the Command queue, ensure that your software performs the following steps:

Procedure

1. Allocate memory for the Command queue.
2. Configure the Command queue size and base address by writing to the SMMU_CMDQ_BASE register.

   Note

   The queue size can affect how many bits of the SMMU_CMDQ_CONS and SMMU_CMDQ_PROD indices are writeable. It is therefore important that you perform this step before writing to SMMU_CMDQ_CONS and SMMU_CMDQ_PROD.

3. Set the queue read index in SMMU_CMDQ_CONS and the queue write index in SMMU_CMDQ_PROD to 0.

   Note

   Setting the queue read index and the queue write index to the same value indicates that the queue is empty.

B.1.2 Allocating the Event queue

The MMU-600 uses the Event queue to signal events. Software must allocate memory for the Event queue and configure the appropriate registers in the MMU.

To allocate the Event queue, ensure that your software performs the following steps:

Procedure

1. Allocate memory for the Event queue.
2. Configure the Event queue size and base address by writing to the SMMU_EVENTQ_BASE register.

   Note

   The queue size can affect how many bits of the SMMU_EVENTQ_CONS and SMMU_EVENTQ_PROD indices are writeable. It is therefore important that you perform this step before writing to SMMU_EVENTQ_CONS and SMMU_EVENTQ_PROD.

3. Set the queue read index in SMMU_EVENTQ_CONS and the queue write index in SMMU_EVENTQ_PROD to 0.

   Note

   Setting the queue read index and the queue write index to the same value indicates that the queue is empty.

B.1.3 Configuring the Stream table

The Stream table is a configuration structure in memory that uses a *Context Descriptor* (CD) to locate translation data for a transaction. Software must allocate memory for the Stream table, configure the table format, and populate the table with *Stream Table Entries* (STEs).

To configure the Stream table, ensure that your software performs the following steps:

Procedure

1. Allocate memory for the Stream table.
2. Configure the format and size of the Stream table by writing to SMMU_STRTAB_BASE_CFG.
3. Configure the base address for the Stream table by writing to SMMU_STRTAB_BASE.
4. Prevent uninitialized memory being interpreted as a valid configuration by setting STE.V = 0 for each STE to mark it as invalid.
5. Ensure that written data is observable to the SMMU by performing a *Data Synchronization Barrier* (DSB) operation.

   If SMMU_IDR0.COHACC = 0, the system does not support coherent access to memory for the TCU. In such cases, additional steps might be required to ensure that the written data is observable to the SMMU.

B.1.4 Initializing the Command queue

Software must initialize the Command queue by enabling it and checking that the enable operation is complete.

To initialize the Command queue, ensure that your software performs the following steps:

Procedure

1. Enable the Command queue by setting the SMMU_S_CR0.CMDQEN bit to 1.
2. Check that the enable operation is complete by polling SMMU_S_CR0ACK until CMDQEN reads as 1.

B.1.5 Initializing the Event queue

Software must initialize the Event queue by enabling it and checking that the enable operation is complete.

To initialize the Event queue, ensure that your software performs the following steps:

Procedure

1. Enable the Event queue by setting the SMMU_S_CR0.EVENTQEN bit to 1.
2. Check that the enable operation is complete by polling SMMU_S_CR0ACK until EVENTQEN reads as 1.

B.1.6 Invalidating TLBs and configuration caches

Before use, the MMU-600 TLBs and configuration cache structures must be invalidated by issuing commands to the Command queue. Alternatively, Secure software can invalidate all TLBs and caches with a single write.

To invalidate TLB entries, ensure that your software issues the appropriate command for the translation context. To invalidate:

- TLB entries for Non-secure EL1 contexts, issue `CMD_TLBI_NSNH_ALL`.
- TLB entries for EL2 contexts, issue `CMD_TLBI_EL2_ALL`.
- TLB entries for EL3 contexts, issue `CMD_TLBI_EL3_ALL`.
- TLB entries for Secure EL1 contexts, issue `CMD_TLBI_NH_ALL`.

——— **Note** ———

Commands to invalidate Secure TLB entries can only be issued through the Secure Command queue. For a system that implements two security states, Secure software must issue the appropriate command to the Secure Command queue for the first TLB invalidation. If your system does not use Secure software, you can permit Non-secure software to access SMMU_S_INIT by using either **sec_override** or the TCU_SCR register.

———————————

To invalidate both the TCU configuration cache and the TBU combined configuration cache and TLB, issue the `CMD_CFGI_ALL` command.

To force all previous commands to complete, issue `CMD_SYNC`.

To invalidate all configuration caches and TLB entries for all translation regimes and security states, ensure that Secure software:

1. Sets SMMU_S_INIT.INV_ALL to 1. The SMMU sets SMMU_S_INIT.INV_ALL to 0 after the invalidation completes.
2. Polls SMMU_S_INIT.INV_ALL to check it is set to 0 before continuing the SMMU configuration.

See the *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3.0* for more information about issuing commands to the Command queue.

B.1.7 Creating a basic Context Descriptor

A *Context Descriptor* (CD) is a data structure in system memory. A CD defines how Stage 1 translation is performed. The SubstreamID is used to select the CD.

To create a CD, ensure that your software performs the following steps:

1. Allocate 64 bytes of memory for the CD.
2. Configure the CD fields according to the information in the following table.

Table B-1 Configuring the CD

Field	Description
AA64	Translation table format: **0** AArch32. **1** AArch64.
EPD0	Enable translations for TTB0 by setting EPD0 to 0.

Table B-1 Configuring the CD (continued)

Field	Description
TTB0	Base address of translation table 0.
TG0	Translation granule size for TTB0 when CD.AA64 = 1.
IR0 OR0	Cacheability attribute to use for translation table walks to TTB0: **00** Non-cacheable. **01** Write-Back Cacheable, Read-Allocate Write-Allocate. **10** Write-through Cacheable, Read-Allocate.
SH0	Shareability of translation table walks to TTB0: **00** Non-shareable. **01** Outer Shareable. **10** Inner Shareable.
EPD1	If the StreamWorld supports split address spaces, enable table walks for TTB1.
ENDI	The endianness for the translation tables.
IPS	The IPA size when CD.AA64 = 1.
ASET	Defines whether the ASID values are shared with the ASID values of an Arm processor. ——— **Note** ——— If you expect this context to receive broadcast TLB invalidation commands from a PE, set ASET to 0. ———
V	Valid CD. This field must be set to 1.

B.1.8 Creating a Stream Table Entry

Each *Stream Table Entry* (STE) configures how Stage 2 translation is performed, and how the *Context Descriptor* (CD) table can be found. The StreamID is used to select an STE.

To create an STE, ensure that your software performs the following steps:

1. Allocate 64 bytes of memory for the STE.
2. Set the STE.Config field as required for Stage 1 translation, Stage 2 translation, or translation bypass:

   `0b000` No traffic can pass through the MMU. An abort is returned.

   `0b100` Stage 1 and Stage 2 bypass.

   `0b101` Stage 1 translation Stage 2 bypass.

   `0b110` Stage 1 bypass Stage 2 translation.

   `0b111` Stage 1 and Stage 2 translation.
3. If Stage 1 translation is enabled, you can set the following fields:

   STE.S1CDMax Controls whether STE.S1ContextPtr points to a single CD or a CD table.

   STE.S1Fmt If STE.S1CDMax > 0, configures the format of the CD table.

   STE. S1ContextPtr Contains a pointer to either a CD or a CD table. If Stage 2 translation is enabled, this pointer is an *intermediate physical address* (IPA), otherwise it is an untranslated *physical address* PA.
4. If Stage 2 translation is enabled, you can set the following fields:

   STE.S2TTB Points to the Stage 2 translation table base address.

STE.S2PS	Contains the PA size of the stage 2 PA range.
STE.S2AA64	Indicates whether the Stage 2 tables are AArch32 or AArch64 format.
STE.S3ENDI	Set this field to the required endianness for the stage 2 translation tables.
STE.S2AFFD	Disable Access Flag faults for Stage 2 translation.
STE.S2TG	`0b00`: 4KB. `0b01`: 64KB. `0b10`: 16KB.
STE.S2IR0 and STE.S2OR0	`0b00`: Non-cacheable. `0b01`: Write-Back Cacheable, Read-Allocate Write-Allocate. `0b10`: Write-through Cacheable, Read-Allocate.
STE.S2SH0	`0b00`: Non-shareable. `0b01`: Outer Shareable. `0b10`: Inner Shareable.
STE.S2VMID	Contains the VMID associated with these translations.

B.2 Enabling the SMMU

Software can enable the SMMU by writing to SMMU_CR0 after the Stream table is populated.

To enable the SMMU:

Procedure

1. Ensure that all Stream table entries are populated in memory.
2. Set the SMMU_CR0.SMMUEN bit to 1.
3. Check that the enable operation is complete by polling SMMU_CR0ACK until SMMUEN reads as 1.

Appendix C
Revisions

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

It contains the following section:

- *C.1 Revisions* on page Appx-C-140.

C.1 Revisions

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

Table C-1 Issue 0000-00

Change	Location	Affects
First release	-	-

Table C-2 Differences between Issue 0000-00 and Issue 0000-01

Change	Location	Affects
Clarified feature list.	*1.3 Features* on page 1-14.	All revisions.
Added revised information about TCU, TBU, and DTI interconnect.	*2.1 About the functions* on page 2-22.	All revisions.
Added various clarifications.	*2.2 Interfaces* on page 2-28.	All revisions.
Added various clarifications.	*2.3.2 Performance Monitoring Unit* on page 2-37.	All revisions.
Added new section.	*SMMUv3 PMU register architectural options* on page 2-42.	All revisions.
Added information about DTI.	*2.3.1 DTI overview* on page 2-36.	All revisions.
Added various clarifications.	*2.3.6 Quality of Service* on page 2-48.	All revisions.
Added new section.	*2.3.11 Conversion between ACE-Lite and ARMv8 attributes* on page 2-50.	All revisions.
Added various clarifications.	*2.4 Constraints and limitations of use* on page 2-54.	All revisions.
Amended address ranges.	*3.3 MMU-600 memory map* on page 3-69.	All revisions.
New subsection TCU and TBU PMU identification register summary.	*3.4 Register summary* on page 3-71.	All revisions.
Added new section.	*3.6 TCU PMU Component and Peripheral ID Registers* on page 3-75.	All revisions.
Modified bits[2:0].	*3.7.1 TCU_CTRL* on page 3-76.	All revisions.
Amended section.	*3.7.7 TCU_NODE_STATUSn* on page 3-82.	All revisions.
Added new sections.	*3.8.4 TCU_ERRGEN* on page 3-87.	All revisions.
	3.10 TBU PMU Component and Peripheral ID Registers on page 3-90.	
	3.12.4 TBU_ERRGEN on page 3-95.	
Amended sections.	*A.9 TCU interrupt signals* on page Appx-A-108. *A.12 TBU TBS interface signals* on page Appx-A-111. *A.13 TBU TBM interface signals* on page Appx-A-115. *A.18 TBU interrupt signals* on page Appx-A-123. *A.19 TBU tie-off signals* on page Appx-A-124.	All revisions.

Table C-2 Differences between Issue 0000-00 and Issue 0000-01 (continued)

Change	Location	Affects
Added new sections.	*A.20 DTI interconnect switch signals* on page Appx-A-126.	All revisions.
	A.21 DTI interconnect sizer signals on page Appx-A-128.	
	A.22 DTI interconnect register slice signals on page Appx-A-130.	
	B.1.7 Creating a basic Context Descriptor on page Appx-B-135.	
	B.1.8 Creating a Stream Table Entry on page Appx-B-136.	

Table C-3 Differences between Issue 0000-01 and Issue 0001-00

Change	Location	Affects
Added new section.	*1.2.5 AMBA APB protocol* on page 1-13.	All revisions.
Modified description of Main TLB.	*2.1.2 Translation Buffer Unit* on page 2-25.	All revisions.
Added information about **sup_btm** signal.	*2.3.7 Distributed Virtual Memory (DVM) messages* on page 2-48.	All revisions.
Added a note about the configurability of the ID register values.	*2.4.1 SMMUv3 support* on page 2-54.	All revisions.
Modified description of SMMU_IIDR.Revision.	*2.4.1 SMMUv3 support* on page 2-54.	r0p1.
Clarified description of CleanShared, CleanInvalid, MakeInvalid, and CleanSharedPersist transaction handling.	*Transactions that can result in a translation fault* on page 2-57.	All revisions.
Added SMMU_PMCG_IRQ_STATUS to list of unimplemented PMCG registers.	*3.1 About the programmers model* on page 3-62.	All revisions.
Modified the value and description of SMMU_PIDR2[7:4] and SMMU_PIDR3[7:4].	*3.5 TCU Component and Peripheral ID Registers* on page 3-74. *3.6 TCU PMU Component and Peripheral ID Registers* on page 3-75.	r0p1.
Modified register description. Modified register bits [31:16] and [7:0].	*3.7.1 TCU_CTRL* on page 3-76.	r0p1.
Added information about calculating the offset of a specific register.	*3.7.6 TCU_NODE_CTRLn* on page 3-81. *3.7.7 TCU_NODE_STATUSn* on page 3-82.	All revisions.
Added a note to DCC and DWC bit descriptions about conditions that apply when setting the bits.	*3.8.4 TCU_ERRGEN* on page 3-87.	All revisions.
Modified the value and description of SMMU_PIDR2[7:4] and SMMU_PIDR3[7:4].	*3.9 TBU Component and Peripheral ID Registers* on page 3-89. *3.10 TBU PMU Component and Peripheral ID Registers* on page 3-90.	r0p1.

Table C-3 Differences between Issue 0000-01 and Issue 0001-00 (continued)

Change	Location	Affects
Modified register description. Modified register bits.	*3.11.1 TBU_CTRL* on page 3-91.	r0p1.
Added a note to DMTLB bit description about conditions that apply when setting the bit.	*3.12.4 TBU_ERRGEN* on page 3-95.	All revisions.

Table C-4 Differences between Issue 0001-00 and Issue 0001-01

Change	Location	Affects
Clarified description of translation manager.	*2.1.2 Translation Buffer Unit* on page 2-25.	All revisions.
Clarified note about DTI translation requests.	*2.3.2 Performance Monitoring Unit* on page 2-37.	All revisions.
Clarified note about configurable values. Added note to SMMU_IIDR table entry.	*2.4.1 SMMUv3 support* on page 2-54.	All revisions.
Added note to clarify reset values of architectural registers. Modified incorrect entries for SMMU_S_GBPA in SMMUv3 architectural registers table.	*3.2 SMMU architectural registers* on page 3-64.	All revisions.
Modified introductory description of TCU_CTRL.	*3.7.1 TCU_CTRL* on page 3-76.	r0p1.
Modified register name.	*3.8.2 TCU_ERRCTLR* on page 3-84. *3.12.2 TBU_ERRCTLR* on page 3-93.	r0p1.
Modified section title. Removed **dftclkenable** signal. Added **mbistresetn** and **mbistreq** signals.	*A.11 TCU and TBU test and debug signals* on page Appx-A-110.	All revisions.

Table C-5 Differences between Issue 0001-01 and Issue 0002-00

Change	Location	Affects
Modified description of configuration inputs.	*1.6.2 Design flow* on page 1-18.	All revisions.
Modified AXI5 extensions list. Removed two notes.	*TBU TBS interface* on page 2-30. *TBU TBM interface* on page 2-31.	All revisions.
Removed information about **sec_override**. Removed note.	*2.3.2 Performance Monitoring Unit* on page 2-37.	All revisions.
Clarified information about SMMU_PMCG_SMR0 event filtering.	*SMMUv3 architectural performance events* on page 2-37. *MMU-600 TCU events* on page 2-38. *MMU-600 TBU events* on page 2-40.	All revisions.

Table C-5 Differences between Issue 0001-01 and Issue 0002-00 (continued)

Change	Location	Affects
Changed Low_Power_Signals to Wakeup_Signals in the table.	*AXI5 support* on page 2-59.	All revisions.
Modified reset value of NS_INIT.	*3.7.5 TCU_SCR* on page 3-80.	All revisions.
Modified register names in table.	*3.10 TBU PMU Component and Peripheral ID Registers* on page 3-90.	All revisions.
Modified the value of SMMU_PMCG_CIDR1.	*3.10 TBU PMU Component and Peripheral ID Registers* on page 3-90.	All revisions.
Modified description of TCU tie-off signals.	*A.10 TCU tie-off signals* on page Appx-A-109	All revisions.
Modified description of TBU tie-off signals.	*A.19 TBU tie-off signals* on page Appx-A-124	All revisions.
Modified value and description of SMMU_PIDR3[7:4].	*3.5 TCU Component and Peripheral ID Registers* on page 3-74. *3.9 TBU Component and Peripheral ID Registers* on page 3-89.	r0p2.
Modified value and description of SMMU_PMCG_PIDR3[7:4].	*3.6 TCU PMU Component and Peripheral ID Registers* on page 3-75. *3.10 TBU PMU Component and Peripheral ID Registers* on page 3-90.	r0p2.

Table C-6 Differences between Issue 0002-00 and Issue 0100-00

Change	Location	Affects
Clarified description of AMBA ACE5 compliance.	*1.2.4 AMBA ACE5-Lite and AMBA® AXI5 protocol* on page 1-13.	r1p0.
Added new features.	*1.3 Features* on page 1-14.	r1p0.
Added TBU direct indexing and MTLB partitioning information to Main TLB description.	*2.1.2 Translation Buffer Unit* on page 2-25.	r1p0.
Added information about ACE configuration.	*TBU TBS interface* on page 2-30. *TBU TBM interface* on page 2-31.	r1p0.
Modified address width.	*TBU TBM interface* on page 2-31.	All revisions.
Added new event, CC miss.	*MMU-600 TCU events* on page 2-38.	r1p0.
Added new sections.	*2.3.3 ACE protection support* on page 2-43. *Stalling faults* on page 2-46. *2.3.4 TBU direct indexing and MTLB partitioning* on page 2-46. *2.3.9 TCU prefetch* on page 2-49.	r1p0.
Added new section.	*2.3.8 TCU transaction handling* on page 2-49.	All revisions.

Table C-6 Differences between Issue 0002-00 and Issue 0100-00 (continued)

Change	Location	Affects
Fixed incorrect references to **aruser_s** and **awuser_s**. Sentence now correctly refers to **aruser_m** and **awuser_m**. Other clarifications.	*2.3.12 AXI USER bits defined by the MMU-600 TBU* on page 2-52.	All revisions.
Added information about S1HWATTR[3:0] and S2HWATTR[3:0].	*2.3.12 AXI USER bits defined by the MMU-600 TBU* on page 2-52.	r1p0.
Added PRI field to SMMU_IDR0. Modified value of PRIQS field in SMMU_IDR1. Added PPS field to SMMU_IDR3. Removed statement that said PRIQ_ABT_ERR global error cannot occur.	*2.4.1 SMMUv3 support* on page 2-54.	r1p0.
Added new sections.	*Upstream ACE master restrictions* on page 2-60. *Avoiding deadlock when using fully coherent ACE masters* on page 2-60.	r1p0.
Modified value and description of SMMU_PIDR2[7:4] and SMMU_PIDR3[7:4].	*3.5 TCU Component and Peripheral ID Registers* on page 3-74. *3.9 TBU Component and Peripheral ID Registers* on page 3-89.	r1p0.
Removed SMMU_PRIQ_* from list of non-implemented registers.	*3.1 About the programmers model* on page 3-62.	r1p0.
Added new registers: • SMMU_PRIQ_BASE. • SMMU_PRIQ_PROD. • SMMU_PRIQ_CONS. • SMMU_PRIQ_IRQ_CFG0. • SMMU_PRIQ_IRQ_CFG1. • SMMU_PRIQ_IRQ_CFG2.	*3.2 SMMU architectural registers* on page 3-64.	r1p0.
Added new registers: • SMMU_PMCG_PMAUTHSTATUS. • SMMU_PMCG_PMDEVARCH. • SMMU_PMCG_PMDEVTYPE.	*3.2 SMMU architectural registers* on page 3-64.	All revisions.
Modified value and description of SMMU_PMCG_PIDR2[7:4] and SMMU_PMCG_PIDR3[7:4].	*3.6 TCU PMU Component and Peripheral ID Registers* on page 3-75. *3.10 TBU PMU Component and Peripheral ID Registers* on page 3-90.	r1p0.
Added new signal **pri_q_irpt_ns**.	*A.9 TCU interrupt signals* on page Appx-A-108.	r1p0.
Added new signals for ACE TBU configurations.	*A.12 TBU TBS interface signals* on page Appx-A-111. *A.13 TBU TBM interface signals* on page Appx-A-115.	r1p0.
Added new signal **cmo_disable**.	*A.19 TBU tie-off signals* on page Appx-A-124.	r1p0.