ARM® CoreLink™ MMU-500 System Memory Management Unit

Revision: r0p0

Technical Reference Manual

ARM®

ARM DDI 0517A (ID090313)

ARM CoreLink MMU-500 System Memory Management Unit
Technical Reference Manual

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
22 August 2013	A	Non-Confidential	First release for r0p0

Proprietary Notice

Confidentiality Status

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

Product Status

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

Web Address

`http://www.arm.com`

Contents
ARM CoreLink MMU-500 System Memory Management Unit Technical Reference Manual

Preface

This preface introduces the *ARM® Corelink™ MMU-500 System Memory Management Unit (MMU-500) Technical Reference Manual* in the following sections:

About this book

This book is for the MMU-500.

Product revision status

The r*n*p*n* identifier indicates the revision status of the product described in this book, where:

r*n* Identifies the major revision of the product.

p*n* Identifies the minor revision or modification status of the product.

Intended audience

This book is written for system designers, system integrators, and programmers who are designing or programming a device that uses the MMU-500.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this for an introduction to the MMU-500 and its features.

Chapter 2 *Functional Description*

Read this for an overview of the major functional blocks and the operation of the MMU-500.

Chapter 3 *Programmers Model*

Read this for a description of the MMU-500 memory map and registers.

Appendix A *Signal Descriptions*

Read this for a description of the MMU-500 signals.

Appendix B *Revisions*

Read this for a description of 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 *ARM® Glossary*, `http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html`.

Conventions

This book uses the conventions that are described in:

- *Typographical conventions* on page vii.
- *Signals* on page vii.

Typographical conventions

The following table describes the typographical conventions:

Typographical conventions

Style	Purpose
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.

Signals

The signal conventions are:

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

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

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

Additional reading

This section lists publications by ARM and by third parties.

See Infocenter, http://infocenter.arm.com, for access to ARM documentation.

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* (ARM IHI 0062).

The following confidential books are only available to licensees:

- *ARM® CoreLink™ MMU-500 System Memory Management Unit Supplement to AMBA® Designer (ADR-400) User Guide* (ARM DSU 0031).
- *ARM® CoreLink™ MMU-500 System Memory Management Unit Technical Reference Manual Supplement* (ARM DSU 0030).

- *ARM® CoreLink™ MMU-500 System Memory Management Unit Implementation Guide* (ARM DII 0289).
- *ARM® CoreLink™ MMU-500 System Memory Management Unit Integration Manual* (ARM DIT 0051).
- *ARM® CoreSight™ Architecture Specification* (ARM DSU 0029).
- *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R editions* (ARM DDI 0406).
- *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile* (ARM DDI 0487).
- *ARM® AMBA® AXI™ and ACE™ Protocol Specification AXI3™, AXI4™, and AXI4-Lite™ ACE™ and ACE-Lite™* (ARM IHI 0022).
- *ARM® Low Power Interface Specification* (ARM IHI 0068).

Feedback

ARM welcomes feedback on this product and its documentation.

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.
- The number, ARM DDI 0517A.
- The page numbers to which your comments apply.
- 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-500 in the following sections:

1.1 About the MMU-500

The MMU-500 is a system-level *Memory Management Unit* (MMU), that translates an input address to an output address, by performing one or more translation table walks.

It supports the translation table formats defined by the ARM architecture, ARMv7 and ARMv8, and 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 output IPA and then translate that IPA to a PA. The MMU-500 performs a translation table walk for each stage of the translation.

A single stage of address translation requires a single translation table walk. This walk often requires multiple translation table lookups, that are called the levels of lookup.

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 defined by the stage 1 translation.

A stage of address translation can be disabled, or bypassed, and the MMU-500 can define memory attributes for a bypassed stage of translation.

The MMU recognizes independent Secure and Non-secure translation contexts. A translation context provides information and resources required by the MMU-500 to process a transaction.

For the stage 1 translations that are typically associated with application and *Operating System* (OS) level operation, the VA range can be split into two subranges, each with associated translation tables and control registers.

These features mean the MMU-500 can perform all of the address translations defined by the ARMv7 and ARMv8 architectures, for memory accesses from either AArch32 state or from AArch64 state.

Stage 1 translations are supported for both Secure and Non-secure translation contexts. Usually, the appropriate OS:

- Defines the translation tables, in memory, for the stage 1 translations for its security state.
- Programs the MMU-500 to configure those stage 1 translations, and then enables the translations.

Stage 2 translations are supported only for Non-secure translation contexts. For Non-secure processor operation, the typical usage model for two stages of address translation is as follows:

- The Non-secure OS defines the stage 1 address translations for application level and OS level operation. Typically, it does this believing it is defining the mapping from VAs to PAs, but it is actually defining the mapping from VAs to IPAs.

  Note

  This means all the addresses the OS uses in the translation tables it defines are in the IPA address space, and require a stage 2 translation to map them to the PA address space.

- The hypervisor defines the stage 2 address translations, that map the IPAs to PAs. It does this as part of its virtualization of one or more Non-secure guest operating systems.

The MMU-500 can cache the result of a translation table lookup in a *Translation Lookaside Buffer* (TLB). This means the MMU-500 also supports TLB maintenance operations.

For more information about:

- The features of the MMU-500, see the *ARM® System Memory Management Architecture Specification*.
- Address translation, including the translation table formats and TLB maintenance operations, see either:
  - The *ARM® Architecture Reference Manual, ARMv7-A and ARMv-7 R edition*.
  - The *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.

The MMU-500 has the following key components:

***Translation Buffer Unit* (TBU)**
: The TBU contains a *Translation Look-aside Buffer* (TLB) that caches page tables. The MMU-500 implements a TBU for each connected master, and a TBU can be implemented so that it is local to the master rather than local to the MMU-500.

***Translation Control Unit* (TCU)**
: Controls and manages the address translations. The MMU-500 implements a single TCU.

Interconnect
: Connects the multiple TBUs to the TCU.

Figure 1-1 on page 1-4 shows the block diagram for MMU-500.

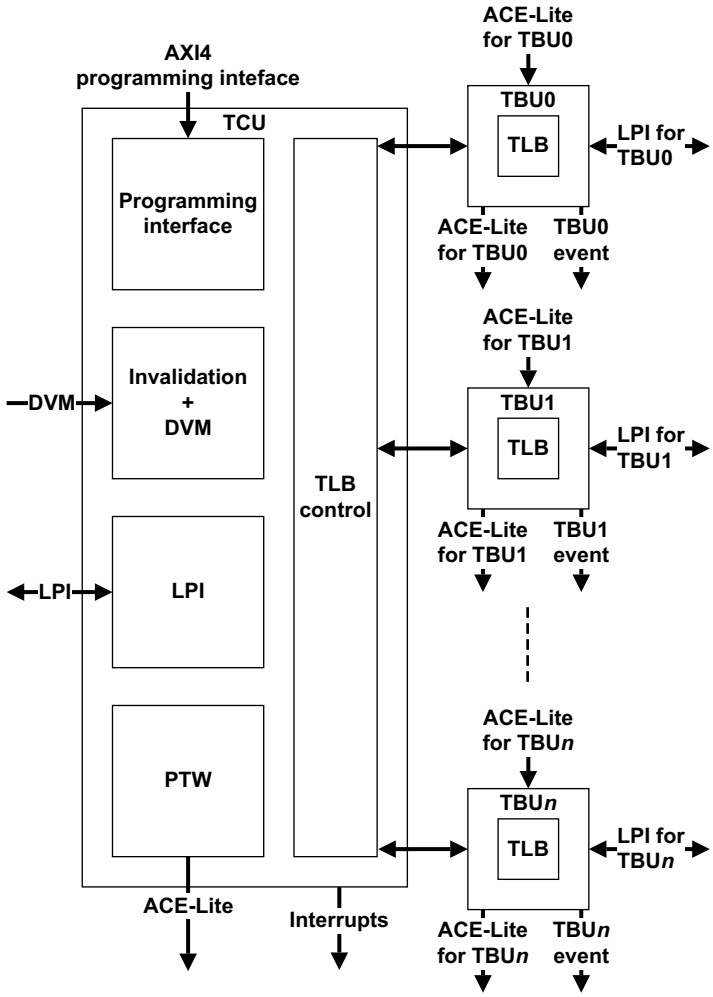


Figure 1-1 MMU-500 block diagram

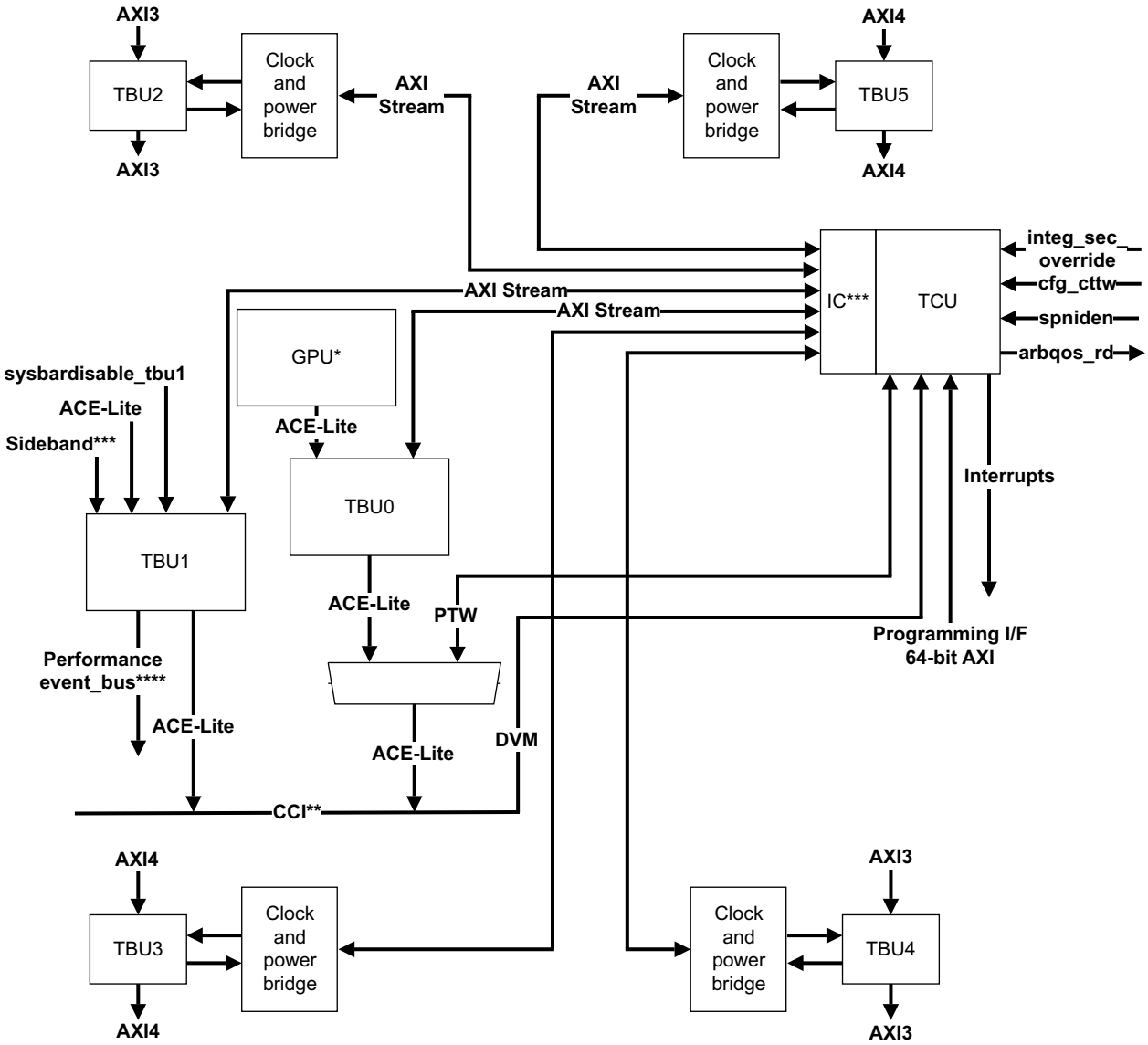
See Chapter 2 *Functional Description* for more information about logical processing steps, interfaces, and operational features.

The following are example masters for the MMU-500:

- GPUs.
- Video engines.
- *Direct Memory Access* (DMA) controllers.
- *Color LCD* (CLCD) controllers.
- Network controllers.

1.1.1 MMU-500 example system

Figure 1-2 on page 1-5 shows the MMU-500 in an example ARM processor and *CoreLink™ Cache Coherent Interconnect-400* (CCI-400) system, performing address translation functions for a *Graphics Processor Unit* (GPU).



* - *Graphics Processor Unit* (GPU) is an example master for the MMU-500.
* - *Cache Coherent Interconnect* (CCI) is not a part of the MMU-500.
** - *Interconnect* (IC).
*** - **sysbardisable**, performance event bus, and other sideband signals are present on all TBUs. These are shown on only one TBU for convenience.

Figure 1-2 MMU-500 in system context

1.2 Compliance

This TRM complements architecture reference manuals, architecture specifications, protocol specifications, and relevant external standards. It does not duplicate information from these sources.

ARM SMMU architecture

The MMU-500 implements the ARM SMMU architecture v2.

See the *ARM® System Memory Management Unit Architecture Specification*.

ARMv7 and v8 architecture

The MMU-500 supports the ARMv7 and ARMv8 address translation schemes. That is, it supports VMSAv7, VMSAv8-32, and VMSAv8-64. This includes support for the long-descriptor and short-descriptor translation table formats.

See the following documents:

- *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.
- *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R editions*.

***Low-Power Interface* (LPI) support**

The MMU-500 supports the ARM LPI.

See the *ARM® Low Power Interface Specification*.

1.3 Features

The MMU-500 provides the following features:

- Address virtualization to processors and other bus masters in the system.
- Supports stage 1 translations, stage 2 translations, and stage 1 followed by stage 2 translations.
- Programmable *Quality of Service* (QoS).
- Distributed translation support for up to 32 TBUs.
- Translation support for 32-bit to 49-bit virtual address ranges and 48-bit physical address ranges.

  See the following documents:
  - *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.
  - *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R editions*.
- Multiple transaction contexts can apply to address translations for specific streams of transactions.
  - Supports up to 64 configurable contexts and programmable page size. The MMU-500 maps each context by using an input stream ID from the master device that requires address translation.
- Translation support for the following:
  - Stage 1 ARMv7 VMSA.
  - ARMv8 AArch32.
  - AArch64 with 4KB and 64KB granules.
  - Stage 1 followed by stage 2 translations.

  See the following documents:
  - *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.
  - *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R editions*.
- Supports 1-bit error detection in the TBU, and 1-bit error detection and correction in the TCU.
- Supports 4KB, 64KB, 1MB, 2MB, 16MB, 512MB, and 1GB page sizes.

  See the following documents:
  - *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.
  - *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R editions*.
- Arbitrates transactions from different TBUs by using the programmed QoS value.
- Provides page table walk cache for storing intermediate page table walk data.
- Caches page table entries in the TLB.
- Supports TLB *Hit-Under-Miss* (HUM).
- Provides configurable 4-32 PTW depth.
- Provides TLB invalidation through the AMBA 4 DVM signalling or register programming.

See the *ARM® AMBA® AXI™ and ACE™ Protocol Specification AXI3™, AXI4™, and AXI4-Lite™ ACE™ and ACE-Lite™* for more information on the DVM.

- Supports translation and protection checks including support for TrustZone® extensions.
- Fault handling, logging, and signalling that includes demand paging and the support for the stall model.
- Debug and performance-monitoring events.
- One AMBA slave interface that supports ACE-Lite for connecting the bus master device that requires address translations.

  See the *ARM® CoreLink™ MMU-500 System Memory Management Unit Implementation Guide* for more information on connecting AXI3 or AXI4 devices.
- One AMBA master interface for master device transactions or PTWs that support ACE-Lite and DVM.

  See the *ARM® CoreLink™ MMU-500 System Memory Management Unit Implementation Guide* for more information on connecting AXI3 or AXI4 devices.
- An AXI4 interface for programming or configuration.

The MMU-500 is based on the *ARM® System Memory Management Unit Architecture Specification*.

1.4 Interfaces

The MMU-500 supports the following interfaces:

- TCU interfaces:
  - Programming interface.
  - Interrupts.
- TBU interfaces:
  - ACE-Lite interface.
  - Sideband interfaces. (Stream interface and security state determination interface.)
- Common interfaces:
  - LPI and clock gating.
  - Performance interface.
  - Tie-off signal interface.

See *Interfaces* on page 2-4 for more information.

1.5 Configurable options

Table 1-1 shows the options that the MMU-500 implementer can configure. The configurable options of the MMU-500 are classified as follows:

- TBU options.
- TCU options.

Table 1-1 Configurable options

Parameter	Range	Description
TCU options		
`Number of configurable TBUs`	1-16	Use this to select the number of unique TBU configurations. Each TBU configuration can be instantiated multiple times, using the TBU mapping.
`Number of TBUs`	1-32	Use this to select the total number of TBUs.
`TBU mapping`	1-16	Each TBU can be mapped to a TBU configuration, using the configuration number.
`StreamID - width of the sideband signal`	1-10	Use this to select the width of the stream ID sideband signal on each TBU. The stream ID is specified on the sideband signals, and dedicated sideband signals are used for read and write accesses. See *Stream ID* on page 2-11.
`AXI programming interface ID signal width`	1-23	Use this to select the programming interface AXI ID width.
`PTW has a dedicated AXI port`	Enable or disable	Use this to select a dedicated AXI interface for PTWs. This ensures that the other TBU0 AXI interface is reserved for device transactions.
`PTW AXI data bus width`	64 or 128	Use this to set the width, in bits, of the TCU AXI data bus that is used for PTWs. This option is applicable only when the TCU has a dedicated AXI port for PTWs.
`Only stage 2 translations`	Enable or disable	Use this to configure the TCU to support stage 2 only or stage 1 followed by stage 2 translation. You can set the value to one of the following: **Enable** Only the stage 2 translation is supported. **Disable** All translations are supported.
`Number of contexts`	1, 2, 4, 8, 16, 32, or 64	Use this to specify the number of contexts. Select the value as one, only when the device is configured for `Only stage 2 translations`.
`Number of SMRs`	2, 4, 8, 16, 24, 32, 40, 48, 56, 64, or 128	Use this to set the number of *Stream Mapping Registers* (SMR) groups.
`PTW depth`	4, 8, 16, 24, or 32	Use this to set the number of PTWs.
`Macro-TLB depth`	0, 8, 128, 256, 512, 1024, or 2048	Use this to set the macro-TLB depth.
`PTW cache depth`	4, 32, 64, or 128	Use this to set the depth of the PTW cache, the IPA to PA translation cache, and the prefetch buffer.

Table 1-1 Configurable options (continued)

Parameter	Range	Description
TBU options		
Name	NA	Use this to set the name of the TBU.
AXI ID signal width	1-23 bits	Use this to select the incoming AXI ID width.
AXI data bus width	64 or 128 bits	Use this to set the width, in bits, of the AXI data bus.
Depth of write buffer	0, 4, 8, or 16	Use this to select the depth of the write buffer. The write buffer can accommodate multiple bursts up to the depth of the buffer. The MMU-500 does not stall the write data path for transactions that the write buffer can hold. The MMU-500 stalls transactions that cannot fit in the write buffer.
TLB depth	2, 8, 16, 32, 40, 48, 56, 64, or 128	Use this to specify the TLB depth.
Implement the TLB using the memory	Enable or disable	When enabled, you can implement the TLB using RAM. Otherwise, the MMU-500 implements the TLB as flip-flops. Implementing the TLB as RAM optimizes area, but the setup and clock-to-Q delay is higher compared to using flip-flops.
Width of the AXI slave interface AWUSER signals	2-128 bits	Use this to set the width of the AXI slave interface **AWUSER** signals. **Note** You must set the input user width to two bits more than the required data width. See the **AWUSER** signal in the Table A-3 on page A-4.
Width of the AXI slave interface WUSER signals	1-128 bits	Use this to set the width of the AXI slave interface **WUSER** signals.
Width of the AXI slave interface BUSER signals	1-128 bits	Use this to set the width of the AXI slave interface **BUSER** signals.
Width of the AXI slave interface ARUSER signals	2-128 bits	Use this to set the width of the AXI slave interface **ARUSER** signals. **Note** You must set the input user width to two bits more than the required data width. See the **ARUSER** signal in the Table A-9 on page A-7.
Width of the AXI slave interface RUSER signals	1-128 bits	Use this to set the width of the AXI slave interface **RUSER** signals.
TBU in separate clock and power domains	Enable or disable	Enable this to configure a clock and power domain cross bridge between the TBU-TCU and TCU-TBU paths. When disabled, the TBU and TCU are in the same clock and power domain.

Table 1-1 Configurable options (continued)

Parameter	Range	Description
`Depth of the asynchronous first-in first-out buffer on the TCU to the TBU channel`	0, 2, 4, or 10	Use this to configure a buffer in the clock and power domain cross bridge. This option is applicable only when the TBU is in a separate clock and power domain.
`TBU-TCU channel width`	0, 1, or 2	Use this to configure the width of the data path between the TCU and the TBU, and the width of the serial data bus. You can set the width to one of the following: **0** Full width, 14 bytes. By selecting this option, you can send all TBU-TCU messages as one packet. The data width between TBU-TCU is HIGH, but the message delay can be LOW. **1** Half width, 7 bytes. By selecting this option, you can send all TBU-TCU messages as two packets. The data width is half of the full width option, and a trade-off between message delay and data width. **2** 1 byte. By selecting this option, TBU-TCU messages can use maximum 14 clocks. The data width is only one byte, and used to connect lower priority TBUs.
Security options		
`SSD index signal width`	0-10 bits	When you specify the `SSD index signal width` as 0, the Non-secure state is directly assigned to the incoming sideband signals along with the transaction. **Writes** Non-secure state = **wsb_ns**, where **wsb_ns** is the write sideband signal for security. The *Security State Determination* (SSD) index is zero for a Secure access and it is one for a Non-secure master. **Reads** Non-secure state = **rsb_ns**, where **rsb_ns** is the read sideband signal for security. The SSD index is zero for a Secure access and it is one for a Non-secure access. The value driven on the sideband signal SSD index signal is used as a pointer into the SSD index table. You must configure at least one programmable or fixed Non-secure entry in the SSD index table.

Table 1-1 Configurable options (continued)

Parameter	Range	Description
`Specify use of SSDIndex0-7` `Specify SSDIndex0-7`	Disable Secure Programmable-Secure Programmable-Non-secure	Use this to specify Secure entries in the SSD index table. These options are applicable only when the width of the SSD index signal is greater than zero. When the SSD index is determined, the SSD index table comprises bits from 0-$2^{\text{SSD index signal width}}$-1. You must determine the status of all the bits as follows: List of non-programmable indices: • For these indices, the security state of the master is defined and does not change. • You must specify the indices of the masters whose security states are always Secure. List of programmable indices: • You can program the security state of these indices. • You must determine the default state of each master whose security state is programmable. • An SSD index is programmable or non-programmable, and is in the Secure or Non-secure state. By default, an SSD index is in the non-programmable Non-secure state. For example, if the `SSD index signal width` is 6-bit, there are 64 indices in the range 0-63, whose security states must be determined.
TBU timing options		
`AWUSER slave interface registering options`	Forward Reverse Full Bypass	Each AXI channel has a configurable register slice in the MMU-500 slave interface. An I/O delay of 70 percent of the clock is assumed for interfaces that are driven to, or driven by, a register. An I/O delay of 40 percent of the clock is assumed for bypassed interfaces.
`WUSER slave interface registering options`		
`BUSER slave interface registering options`		
`ARUSER slave interface registering options`		
`RUSER slave interface registering options`		
`TBU-TCU channel prebridge register slice 1 options`	Forward Reverse Full Bypass	Each TBU-TCU channel has 0-4 configurable register slices: • 0-2 register slices between the TBU and the clock and power domain cross bridge. • 0-2 register slices between the clock and power domain cross bridge and the TCU.
`TBU-TCU channel prebridge register slice 2 options`		
`TBU-TCU channel postbridge register slice 1 options`		
`TBU-TCU channel postbridge register slice 2 options`		

Table 1-1 Configurable options (continued)

Parameter	Range	Description
TCU-TBU channel prebridge register slice 1 options	Forward Reverse Full Bypass	Each TCU-TBU channel has 0-4 configurable register slices: • 0-2 register slices between the TCU and the clock and power domain cross bridge. • 0-2 register slices between the clock and power domain cross bridge and the TBU.
TCU-TBU channel prebridge register slice 2 options		
TCU-TBU channel postbridge register slice 1 options		
TCU-TBU channel postbridge register slice 2 options		

1.5.1 Output ID width

The following equation defines the output ID width of all TBUs, other than TBU0 in multiplexed configurations:

- TBU output width = Incoming AXI ID width + 1.

  Note

  The extra bit is required for barrier transactions generated by TBU to identify the transactions that are generated by the TBU.

If the AXI interface between TBU0 and TCU is multiplexed, then the output ID width is based on:

- The number of parallel PTW supported in the TCU.
- The input AXI ID width in TBU0.

If the TCU has a separate AXI interfaces, then:

- TCU Output ID width = TCUIDW + 1.

  Where,

  — TCUIDW = log_base_$2^{\text{Nummax_of_parallel_PTW}}$.

  — Nummax_of_parallel_PTW is the number of parallel PTW queues adjusted to the smallest power of two that is greater than this number, if the number is not already a power of 2.

  Note

  The extra bit is required to identify the synchronous complete transaction.

The output AXI ID width follows the following rules:

- When the TBU ID width is in the range 0-TCUIDW, the output width is TCUIDW + 2.
- For TBUIDW > TCUIDW, the output width is TBUIDW + 1.

The value driven on the AXI ID signal is:

- Passing through the incoming transaction on the TBU, with the following address translation condition:

  If TBUIDW is 0

  All 0s.

If TBUIDW is not 0

Incoming ID, appending with all 0s in the MSB.

- All 1s when the TBU generates a synchronous transaction, all 1s.
- `0b10` followed by 0s until TCUIDW when the TCU generates a PTW transaction. The TCU drives the ID in the range (TCUIDW-1) to 0.
- `0b110` followed by 0s when the TCU generates a synchronous complete transaction.

1.6 Product documentation and design flow

This section describes the MMU-500 books and how they relate to the design flow.

See *Additional reading* on page vii for more information about the books described in this section. For information on the relevant architectural standards and protocols, see *Compliance* on page 1-6.

1.6.1 Documentation

The MMU-500 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-500. It is required at all stages of the design flow. The choices made in the design flow can mean that some behavior described in the TRM is not relevant. If you are programming the MMU-500 then contact:

- The implementer to determine:
  - The build configuration of the implementation.
  - What integration, if any, was performed before implementing the MMU-500.
- The integrator to determine the pin configuration of the device that you are using.

Implementation Guide

The *Implementation Guide* (IG) describes:

- The available build configuration options and related issues in selecting them.
- How to configure the *Register Transfer Level* (RTL) with the build configuration options.
- The processes to sign off the configured design.

The ARM product deliverables include reference scripts and information about using them to implement your design.

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

Integration Manual

The *Integration Manual* (IM) describes how to integrate the MMU-500 into an SoC. It describes the pins that the integrator must tie off to configure the macrocell for the required integration. Some of the integration is affected by the configuration options used when implementing the MMU-500.

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

User Guide Supplement

The User Guide Supplement describes how to use the *AMBA® Designer* (ADR-400) application to build and configure the MMU-500.

Technical Reference Manual Supplement

The TRM Supplement describes how to intialize the MMU-500, and how the MMU-500 generates final memory attributes.

1.7 Test features

The MMU-500 includes the clock gating circuitry that you can use to enable the clock during MMU-500 testing.

The *Design For Test* (DFT) port, **dftclkenable**, allows you to bypasses architectural clock gates during a DFT shift.

1.8 Product revisions

This section describes the differences in functionality between product revisions of the MMU-500:

r0p0 First release.

Chapter 2
Functional Description

This section describes the functional operation of the MMU-500 in the following sections:

2.1 About the functions

The TBU and TCU are the major functional blocks of the MMU-500. The TBU caches frequently used address ranges and the TCU performs the page table walk.

Figure 2-1 shows the block diagram for MMU-500.

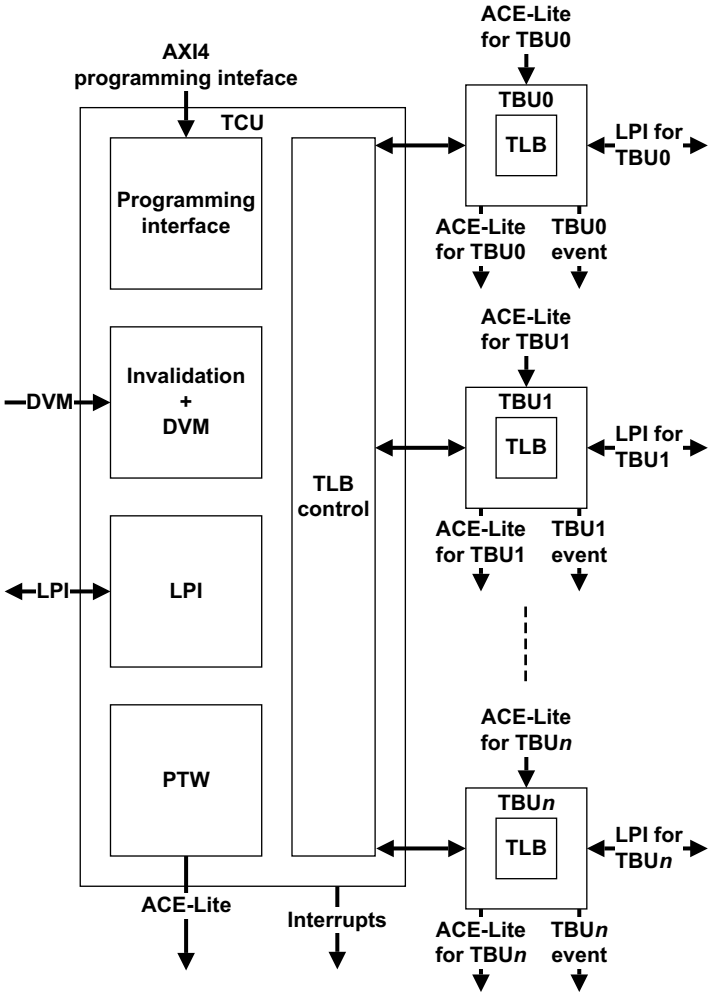


Figure 2-1 MMU-500 block diagram

The MMU-500 applies the following logical processing steps to every transaction that flows in:

1. Determines the security state of the device that originates the transaction. The security attribute presented on **AWPROT[1]** and **ARPROT[1]** is different from the security state of the device. Identifying the security state of the device is called security state determination.
2. Maps an incoming transaction to one of the contexts using an incoming stream ID.
3. Caches frequently used address ranges using the TLB. The best-case hit latency of this caching is two clocks when the TBU address slave register slices are not specified. The best-case latency is three clocks when the TBU address slave register slices are specified.
4. Performs the main memory PTW automatically on an address miss.

5. Shares with the processor the page table formats as specified in the *Large Physical Address Extension* (LPAE) for maximum efficiency.

   For more information on LPAE addresses, see the following documents:
   - *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition*.
   - *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.

6. Applies the required fault handling for every transaction.

7. Performs debug and performance monitoring through programmable performance counters, and reports statistics. For example, TLB refills or number of read or write accesses.

2.2 Interfaces

Figure 2-2 shows the MMU-500 interfaces.

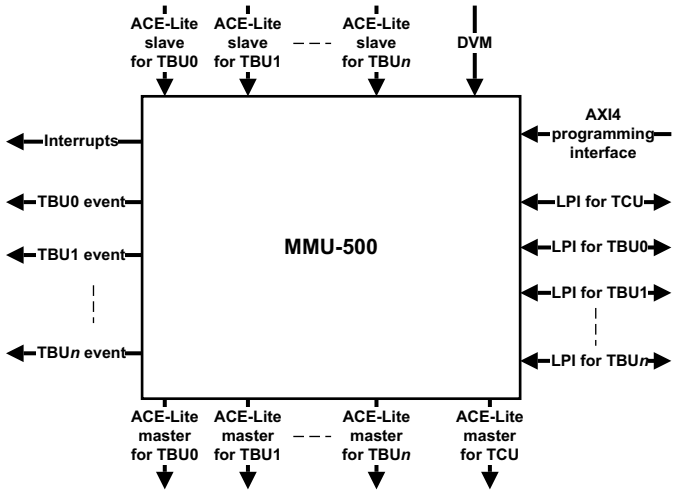


Figure 2-2 Interfaces of the MMU-500

Note

You can connect AXI3 interface, but you must tie-off the inactive ACE-Lite signals.

The MMU-500 contains the following interfaces:

- *TCU interfaces*.
- *TBU interfaces* on page 2-5.
- *Common interfaces* on page 2-7.

2.2.1 TCU interfaces

The MMU-500 supports the following TCU interfaces:

- *Programming interface*.
- *Interrupts* on page 2-5.

Programming interface

The MMU-500 requires a programming interface to permit the software to configure the controller and to initialize the memory devices. See *Modes of operation and execution* on page 3-3 for information about using the 64-bit AXI4 programming interface.

The MMU-500 provides 32-bit address buses, **awaddr_prog[31:0]** and **araddr_prog[31:0]**, but it only uses bits[23:2]. The MMU-500 ignores:

- Bits[31:24], but their presence facilitates the process of integrating the MMU-500 with adjacent RTL blocks, such as an interconnect.
- Bits[1:0], because the MMU-500 only permits word accesses to its internal registers.

This interface operates at the same frequency as the TCU clock.

Interrupts

This interface provides global, per-context, and performance interrupts. See *Interrupt signals* on page A-13 for more information.

2.2.2 TBU interfaces

The MMU-500 supports the following TCU interfaces:

- *ACE-Lite interfaces*.
  The ACE-Lite interface supports the following interfaces:
  - *AXI slave interface*.
  - *AXI master interface*.
  - *Snoop channel interface* on page 2-6.
  - *TBU barrier support* on page 2-6.
- *Sideband interface* on page 2-6.
  The sideband interface supports the following interfaces:
  - *Stream interface* on page 2-7.
  - *Security State Determination interface* on page 2-7.

ACE-Lite interfaces

The MMU-500 uses the ACE-Lite interfaces to receive transactions, translate transactions, and perform PTWs.

You can connect the AXI3 or AXI4 bus to this interface with certain limitations as described in *AXI3 and AXI4 support* on page 2-17.

In this mode, the MMU-500 generates barrier transactions and updates attributes of input barrier transactions. Barrier transactions guarantee the ordering and observation of transactions in a system.

See the *ARM® AMBA® AXI and ACE Protocol Specification AXI3™, AXI4™, and AXI4-Lite™ ACE and ACE-Lite™* for more information on barrier transactions.

The MMU-500 supports DVM signaling for TLB maintenance operations on its PTW port.

AXI slave interface

The MMU-500 supports only the ACE-Lite slave interface for every TBU. You must tie the extra ACE-Lite signals to their inactive values and the **sysbardisable_<tbuname>** signal to HIGH to use the AXI3 or AXI4 interfaces.

You must connect pin-to-pin the read address, write address, read data, write data, and buffered write response channels of the ACE-Lite slave interface, with _s suffix, to an ACE-Lite master interface. In a system, the master interface can be the AXI bus infrastructure output, or the output of a bridge that converts another bus protocol to AXI.

Note

A PTW read interface might be present depending on the specified configuration.

AXI master interface

The MMU-500 supports only the ACE-Lite master interface for every TBU and PTW read interface of the TCU. You must tie the extra ACE-Lite signals to their inactive values and the **sysbardisable_<tbuname>** signal to HIGH to use AXI3 or AXI4 interfaces.

The ACE-Lite master interface, with _m suffix, drives the translated address to the downstream slave. You must connect pin-to-pin the read address, write address, read data, write data, and buffered write response channels to the corresponding ACE-Lite slave interface.

If the MMU-500 is configured to support a dedicated interface for PTWs, you must connect the read address and read data channels of the slave interface associated with the PTWs to the MMU-500 PTW channel. In this configuration, the PTW channel contains the _ptw suffix. For example, **araddr_ptw** and **acaddr_ptw**.

Note

A PTW read interface might be present depending on the specified configuration.

Snoop channel interface

The AC channel of the ACE-Lite interface of the MMU-500 is connected to the CCI-driven AC channel or to the ACE-compatible slave interface that supports DVM messaging. ARM recommends that you use the DVM channel for TLB maintenance operations. If the system cannot access the DVM channel, the **acvalid** signal must be tied LOW, and the programming interface can be used for TLB maintenance operations.

When you configure the MMU-500 to provide a dedicated AXI channel to perform PTWs, the AC channel must be part of the PTW channel.

This interface supports the following:

Snoop data channel

The snoop data channel is not connected to the MMU-500.

Note

The snoop data channel is not supported in the MMU-500.

Snoop address channel

The 44-bit wide snoop address channel is connected to the TCU.

TBU barrier support

The TBU in the MMU-500 receives, passes on, and generates barriers of its own in response to the **SYNC** signal received from the TCU DVM channel.

The MMU-500 generates the DSBSYS barrier after ensuring that all invalidation-related transactions are initiated on receiving one of the following:

- The programmed **SYNC** message.
- The DVM **SYNC** message.

See the *ARM® AMBA® AXI and ACE Protocol Specification AXI3™, AXI4™, and AXI4-Lite™ ACE and ACE-Lite™* for more information on **SYNC** and DVM **SYNC** messages.

Sideband interface

This interface provides associated information along with the ACE-Lite interface. See *Sideband signals* on page A-13 for more information.

Note

The stream and security state determination are associated with the ACE-Lite slave interface to each TBU.

Stream interface

This interface is a sideband interface for the MMU-500 TBU slave interface. It provides information about the translation mechanism that the MMU-500 applies to an incoming transaction.

The MMU-500 samples signals in the interface along with each valid address transaction.

See *Stream ID* on page 2-11 for more information.

Security State Determination interface

This interface is a sideband interface for the MMU-500 TBU slave interface. It provides information about the security state of a transaction.

Similar to the *Stream interface*, the MMU-500 samples signals in this interface along with each valid address transaction.

See *Security determination* on page 2-11 for more information.

2.2.3 Common interfaces

The MMU-500 supports the following interfaces that are common to TBUs and the TCU:

- *Low-power interface for clock gating and power control*.
- *Performance interface* on page 2-9.
- *Tie-off signal interface* on page 2-9.

Low-power interface for clock gating and power control

The MMU-500 contains Low-power interfaces that enable:

- Power gating of the TBU module.
- Clock gating of the TBU module.
- Clock gating of the TCU module.

You can control the power-control interfaces at the system level by a system power-control module. Alternatively, if there is no system control block, you must tie the **qreqn_*** inputs HIGH, and can leave the outputs, **qacceptn_*** and **qactive_***, unconnected.

The MMU-500 never denies a powerdown request, and therefore you must tie LOW the **qdeny_*** input to the system power controller.

You must powerup the TCU module to powerup a TBU module.

Note

The LPI signals are not synchronized. The system must provide the synchronous signals to the MMU-500.

The MMU-500 provides low-power interface and clock gating support in the following manner:

- The TBU and TCU have dedicated Q-channel interfaces for clock gating:
  - **qreqn_tbu_<tbuname>_cg**, **qacceptn_tbu_<tbuname>_cg**, and **qactive_tbu_<tbuname>_cg**.
  - **qreqn_tcu**, **qacceptn_tcu**, and **qactive_tcu**.
- The TBU and the clock or power bridge each have a dedicated Q-channel interface for entering the power-down state:
  - **qreqn_tbu_<tbuname>_pd** and **qacceptn_tbu_<tbuname>_pd**.

— **qreqn_pd_slv_br_<tbuname>** and **qacceptn_pd_slv_br_<tbuname>**.

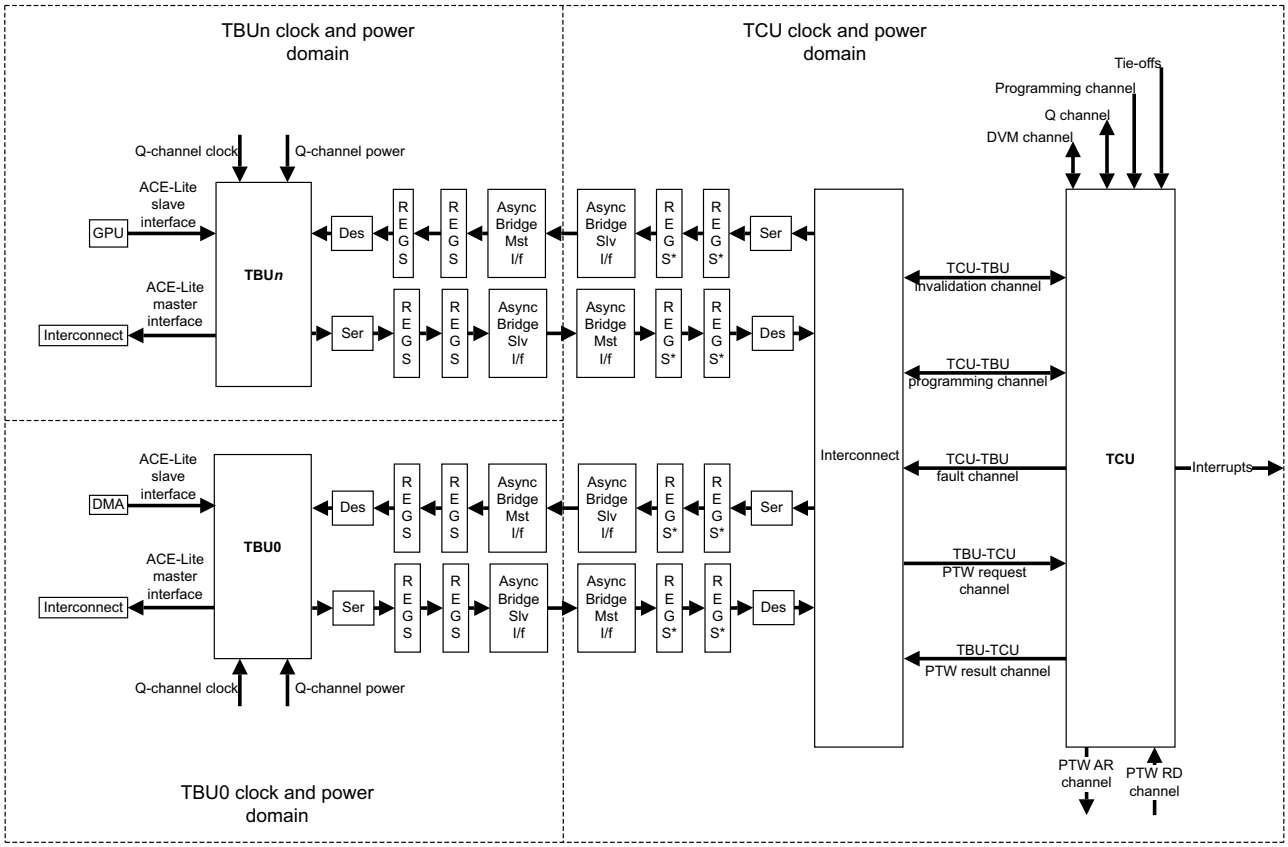
— **qreqn_pd_mst_br_<tbuname>** and **qacceptn_pd_mst_br_<tbuname>**.

Note

If the `TBU in separate clock and power domains` option is disabled, you must tie the **qreqn_tbu_<tbuname>_pd** and **qreqn_pd_br_<tbuname>** signals HIGH.

- The clock or power bridge contains the following **qactive** signals:
  — The **qactive_br_tbu_<tbuname>** signal for handling the cross-boundary clock wakeup to wakup the TBU clock.
  — The **qactive_br_tcu_<tbuname>** signal for handling the cross-boundary clock wakeup to wakeup the TCU clock.

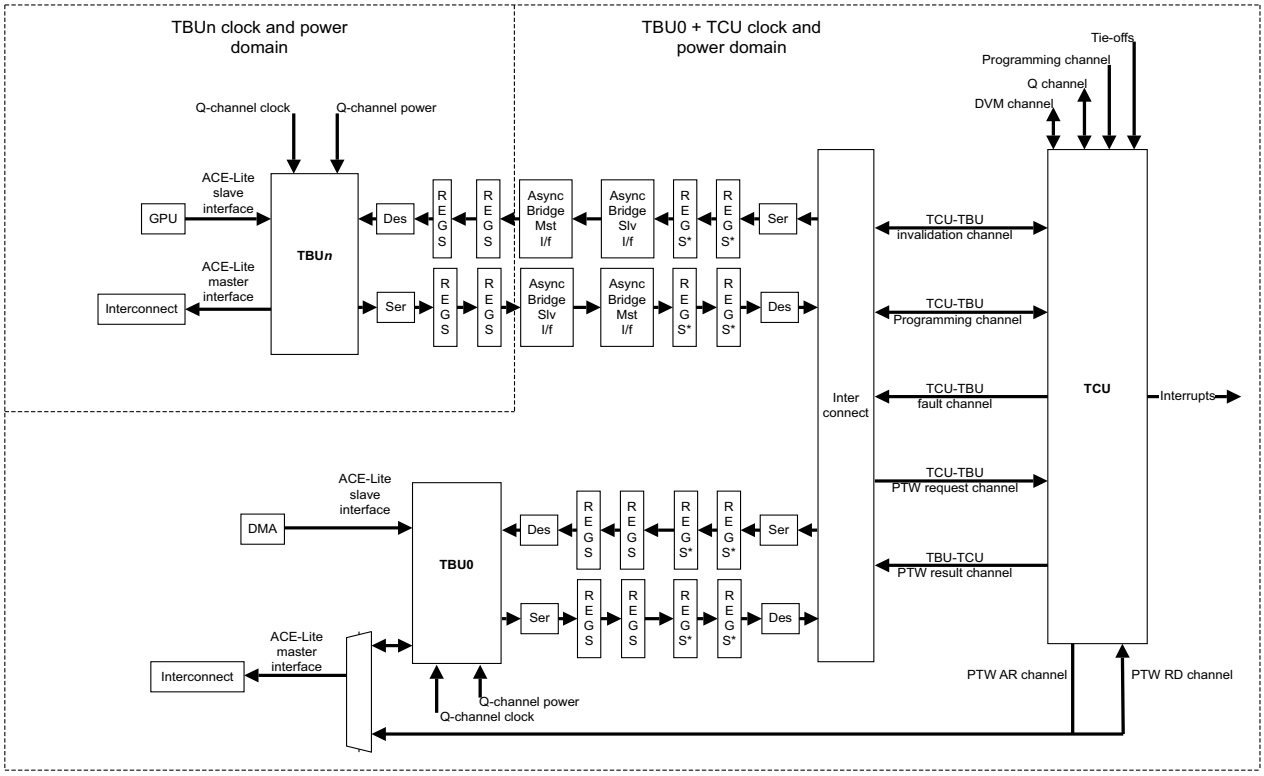
Figure 2-3 shows the possible clock and power domains of the MMU-500.



* AXI stream register slice

Figure 2-3 Clock and power domains of the MMU-500

Figure 2-4 on page 2-9 shows a scenario in which the TBU0 and the TCU share a common clock or power domain and a PTW read channel.



* AXI stream register slice

Figure 2-4 Sharing a common clock or power domain and PTW read channel

See the following documents for more information about low-power interface:

- *ARM® Low Power Interface Specification*.
- *ARM® CoreLink MMU-500 System Memory Management Unit Integration Manual.*

Performance interface

This interface contains the input signal **spniden**, which indicates whether security events need to be considered in the performance counters.

The performance interface also contains an event output interface that provides updates from each TBU to the performance counters.

See *Performance event signals* on page A-14 and *Authentication interface signal* on page A-14 for more information.

Tie-off signal interface

This interface provides configuration information about certain functionality. See *Tie-off signals* on page A-14 for more information.

2.3 Operation

The MMU-500 routes each translation through the following logical processing steps:

1. Security state determination.
2. Context determination.
3. Page table walk, if the translation is not cached in the TLB.
4. Protection checks.
5. Attribute generation or merging, depending on the programming.

You can configure the MMU-500 to bypass the transaction process for a transaction or to fault a transaction regardless of the translation state.

The primary function of the MMU-500 is to provide address translations from an input address to an output address, based on address mapping and memory attribute information stored in translation tables.

The MMU-500 uses the following steps to achieve this:

1. Receives an address transaction, along with security and stream information.
2. Uses the security information received along with a transaction to determine the further processing steps for the transaction. The received security information is the security state of the originator of a transaction. The MMU-500 uses a Secure or Non-secure set of registers, for additional processing of a transaction, depending on the security state of the originator is Secure or Non-secure, respectively. See *Security determination* on page 2-11 for more information.
3. Uses the stream information received along with the transaction to determine the translation mechanism to apply to the transaction. The translation mechanism can be a bypass, a stage 1 translation, a stage 2 translation, or a stage 1 followed by stage 2 translation. See the *ARM® System Memory Management Unit Architecture Specification* for more information.
4. Adds the fault information to the Global Fault Status Register if a fault is identified in the translation process before a context is mapped. The MMU-500 adds the fault information to the Context Banks Fault Status Register if a fault is identified after the context mapping.

   A fault results in an interrupt when interrupt reporting is enabled. You can clear interrupts by clearing the Fault Status Register.

   See the *ARM® System Memory Management Unit Architecture Specification* for more information.
5. The MMU-500 supports both little and big endian translation tables. You can program endianness in the SMMU_CB*n*_SCTLR register. See *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R editions* for more information.

See the *ARM® CoreLink™ MMU-500 System Memory Management Unit Technical Reference Manual Supplement* for information about initialization and configuration.

This section describes how the *ARM® CoreLink™ MMU-500 System Memory Management Unit* operates, and contain the following subsections:

- *Stream ID* on page 2-11.
- *Security determination* on page 2-11.
- *Hit-Under-Miss* on page 2-12.
- *Fault handling* on page 2-13.
- *Implementation defined operational features* on page 2-13.

2.3.1 Stream ID

A stream ID is used to map the incoming transaction to a context by using the stream mapping table. The characteristics of the stream ID are as follows:

- The width of the stream ID is selected during the MMU-500 configuration.
- You must specify the stream ID on a dedicated AXI sideband signal. Select the `Stream ID - width of the sideband signal parameter` value from the range 1-10 bits, and dedicated sideband signals are used for read and write transactions.

For more information about streamID signals, see *Sideband signals* on page A-13.

The stream ID width in the TCU is a constant 15-bits. The stream ID from each TBU is zero-extended to form a 10-bit field appended to a 5-bit TBU ID field, making it 15-bits wide. This arrangement ensures that the stream ID presented to each TBU must not be unique, and happens at the TCU. If the Stream ID presented to each TBU is already unique, and the TBU ID addition is not required, then you must ensure that the TBU ID field is masked in the SMR.

See the *ARM® System Memory Management Unit Architecture Specification* for more information on stream ID-to-context mapping.

2.3.2 Security determination

The MMU-500 determines the Secure ownership of a transaction in one of the following ways:

- Assigns the Non-secure state to an incoming sideband signal along with a transaction:
  - For write accesses, the Non-secure state is the write sideband signal for security.
  - For read accesses, the Non-secure state is the read sideband signal for security.
- Determines the security state of a master by using the input signals, **wsb_ssd_<tbuname>_s** and **rsb_ssd_<tbuname>_s**, that index an SSD index into the SSD index table. The entry in the SSD index table determines whether the master that initiated the transaction is Secure or Non-secure. For more information about SSD signals, see *Sideband signals* on page A-13.
  - You can configure the width of the SSD index in the range 0-10 bits. The MMU-500 uses a separate SSD index for each TBU.
  - You can configure the number of programmable entries in the SSD table in the range 1-32. The security state determination address space supports 15-bit wide SSD indices. This space is equally divided among 32 TBUs starting with TBU0 from the bottom of the address space. Each TBU contains 1024 entries.
  - You can program the security state of the SSD table entries at runtime, or specify the non-programmable and fixed SSD table entries at configuration time.

After the SSD index is determined, the SSD table contains bits from 0 to $2^{\text{SSD index signal width}}-1$. You must determine the status of the bits as follows:

List of non-programmable indices

For these indices, the security state of the master is defined, and does not change.

You must specify the indices of the masters whose security states are always Secure.

List of programmable indices

You can program the security state of the programmable indices.

You must determine the default state of each master whose security state is programmable.

An SSD index can be programmable or non-programmable, and can be in the Secure or Non-secure state. By default, an SSD index is in the non-programmable Non-secure state.

Note

An entry must not be duplicated in more than one list.

You must specify at least one programmable or fixed Non-secure entry for every configuration.

The number of indices is determined by the configured `SSD index signal width`. For example, if the `SSD index signal width` is six bits, there are 64 indices in the range 0-63. You must program the indices to be one of:

- Programmable Secure.
- Programmable Non-secure.
- Non-programmable Secure.

The unprogrammed indices default to non-programmable Non-secure.

The MMU-500 supports debug TLB accesses whose Secure accesses can access Secure and Non-secure TLBs.

The SSD table has a maximum of 32Kb bit space that is divided into 32 parts, with 1Kb assigned to each TBU. The TBU0 space is from 0-1Kb, TBU1 space is from 1-2Kb, and so on. The SSD index that is generated at each TBU, and is a maximum of 10 bits, is indexed into the 1Kb space allocated to the TBU. You must program the SSD table using this information.

Note

The security determination descriptions are valid when the tie-off **integ_sec_override** is set to zero.

When the tie-off **integ_sec_override** is set to one, the following conditions are true:

- All implementation and integration registers can be accessed with a non-secure access. This include the following global space 0 registers:
  - *Auxiliary Configuration Register* (ACR).
  - Debug registers.
- You cannot access any secure registers.
- All transactions are treated as originated from a Non-secure master.

See the *ARM® System Memory Management Unit Architecture Specification* for more information on security determination and extensions.

2.3.3 Hit-Under-Miss

Hit-Under-Miss (HUM) translates a TLB miss transaction and passes the transaction to a downstream slave if the translated TLB miss transaction results in a TLB hit. HUM characteristics for read and write transactions are as follows:

- If the transactions are read accesses, HUM is automatically enabled for read accesses.
- If the transactions are write operations, HUM is enabled or disabled based on the write buffer depth. You can specify the write buffer depth during configuration.

- If the depth of the write buffer is zero, HUM is automatically disabled for write transactions.
- If the depth of the write buffer is a non-zero value, a write hit transaction is translated only if the write data from a missed transaction can be accommodated in the write buffer.
- The number of outstanding missed transactions is determined by the depth of the write buffer. For example, if the depth of the buffer is four, then it can hold two transactions of length two. Each buffer entry holds only one beat of the transaction, even if it is of a narrow width.

Example 2-1 shows a hit under miss condition.

Example 2-1 Hit under miss

Consider that the write buffer depth is eight and there are two missed write transactions of lengths four and three. Both missed write transactions are stored in the write buffer during the PTWs for the transactions. If you perform another transaction before the missed write transactions are processed, the new transaction is passed through.

Note

If the write buffer is full with missed write transactions, HUM cannot happen.

2.3.4 Fault handling

The MMU-500 supports the terminate and stall fault handling modes. However, the MMU-500 does not support fault model overrides from the global space specified by using the *Global Stall Enable* (GSE) and *Stall Disable* (STALLD) bits of the Secure Configuration Register, SMMU_CR0 or SMMU_*s*CR0.

If the *Hit Under Previous Context Fault* (HUPCF) bit of the SMMU_CR0 or SMMU_*s*CR0 register is not enabled, the MMU-500 applies the fault model across TBUs that share the same context.

See the *ARM® System Memory Management Unit Architecture Specification* for more information on fault handling.

2.3.5 Implementation defined operational features

This section lists the operational features of the MMU-500 in the following sections:

- *Outstanding transactions per TBU*.
- *QoS arbitration* on page 2-14.
- *Address width* on page 2-14.
- *Programmable QoS support in the TCU* on page 2-14.

Outstanding transactions per TBU

Outstanding transactions are defined as transactions for which:

- The physical address access is generated and accepted by the slave.
- Write or read responses are stalled.

For every TBU, the MMU-500 supports 128 outstanding transactions each for write and read accesses.

The MMU-500 generates a PTW when accesses from the master result in a TLB miss. However, the MMU-500 supports only eight such parallel PTWs for a TBU. If eight PTWs are pending, a TLB miss on a channel indicates that the MMU-500 cannot accept additional transactions on the write or read channels.

QoS arbitration

The PTWs are initiated by multiple TBUs. Therefore, when there are multiple outstanding transactions in the PTW queue, priority is given to the TBU with the highest QoS. The MMU-500 also reuses the programmed QoS value for PTWs.

The **arqosarb** signal, a sideband signal from the MMU-500 to the CCI, has the highest QoS value compared to other read transactions in the MMU-500, including the transaction present on the PTW bus.

Note

You can leave the unused output ports unconnected.

For address translations, the MMU-500 uses the programmed QoS value.

For individual prefetch accesses, the MMU-500 uses the QoS value of the hit transaction.

For transactions within the same QoS, the MMU-500 uses a first-come, first-served model.

Address width

The incoming address width is fixed at 49-bits, where A[48] specifies VA sub-ranges. You must tie all unused bits to zero. The output address width is 48-bits and the width of the AC address bus is 48-bits.

Note

The MMU-500 does not support peripherals whose address width is greater than 49 bits.

Programmable QoS support in the TCU

You can program the QoS value to be used for each TBU PTW in the TCU. See *TBU QoS registers* on page 3-32.

2.4 Cache structures of the MMU-500

The cache structures of the MMU-500 are described in the following sections:

- *Micro TLB*.
- *Macro TLB*.
- *Prefetch buffer* on page 2-16.
- *Page table walk cache* on page 2-16.
- *IPA to PA cache* on page 2-16.

2.4.1 Micro TLB

The micro TLB in the TBU caches the PTW results returned by the TCU. The TBU compares the PTW results of incoming transactions with the entries in the micro TLB before performing a TCU PTW. The micro TLB is fully associative and you can configure the depth of a micro TLB based on your requirements.

Figure 2-5 shows the micro TLB cache structure.

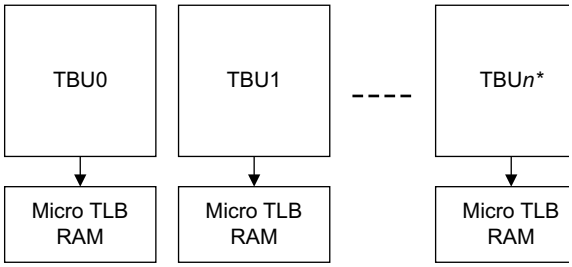


Figure 2-5 Micro TLB cache

See the *ARM® CoreLink™ MMU-500 System Memory Management Unit Supplement to AMBA® Designer (ADR-400) User Guide* for more information on the TBU configurability.

2.4.2 Macro TLB

The macro TLB caches PTW results in the TCU. You can configure the depth of the macro TLB based on your requirements.

Figure 2-6 on page 2-16 show the TCU cache structure, which consists of macro TLBs, Prefetch buffers, and PTW caches.

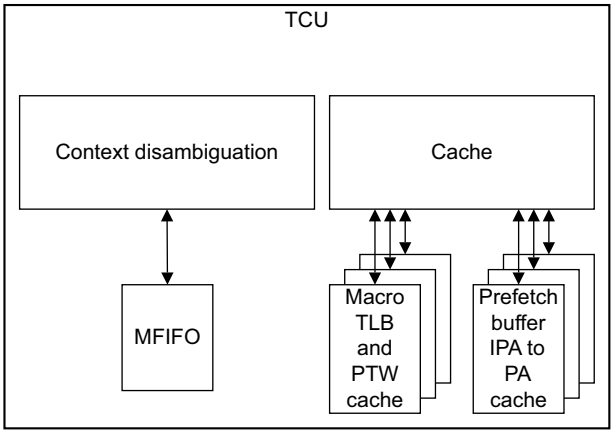


Figure 2-6 TCU cache

See the *ARM® CoreLink™ MMU-500 System Memory Management Unit Supplement to AMBA® Designer (ADR-400) User Guide* for more information on the TCU configurability.

2.4.3 Prefetch buffer

The MMU-500 fetches in advance 4KB and 64KB sized pages into the prefetch buffer. This reduces latency for future PTWs. You can configure the depth of the prefetch buffer.

The prefetch buffer is a single four-way set associative cache that you can enable or disable depending on the context. The prefetch buffer shares RAMs with the macro TLB cache. See *Macro TLB* on page 2-15.

2.4.4 Page table walk cache

The MMU-500 caches partial PTWs to reduce the number of PTWs on a TLB miss. The PTW cache exists in the TCU, and stage 1 and stage 2 level 3 PTWs are cached in the PTW cache.

2.4.5 IPA to PA cache

The MMU-500 implements an IPA to PA cache for stage 1 followed by stage 2 translations.

The IPA to PA cache is a single four-way set associative cache that you can enable or disable depending on the context. The IPA to PA cache shares RAMs with the PTW cache. See *Page table walk cache*.

2.5 Constraints and limitations of use

This sections describes the constraints of the MMU-500.

2.5.1 AXI3 and AXI4 support

The MMU-500 supports the AXI3 and AXI4 protocols when the **sysbardisable_<tbuname>** input signal is tied HIGH. In this mode, the following AXI3 features are not supported:

Write data interleaving

Write data and write address ordering must be the same, otherwise data corruption can happen.

Locked transfer

The input interface on a TBU contains only one bit of the **AWLOCK** and **ARLOCK** signals to ensure compliance with the AXI4 specification. Therefore, locked transfers are not supported even when the **sysbardisable_<tbuname>** signal is HIGH.

The WID signal generation

The MMU-500 does not generate the **WID** signals for the TBU write data channels because these signals are not required for AXI4 and ACE-Lite modes. You must add logic to generate the **WID** signal based on the **WID** signal values that are used for the address channel transfer, and use the values for each write data channel transfer for a transaction.

The MMU-500 does not support write data interleaving. Therefore, the MMU-500 generates write data transfers in the sequence that the write addresses are issued, so you can generate the **WID** signals using the **AWID** signal values.

Example 2-2 shows a scenario in which a FIFO is used to generate the **WID** signal.

Example 2-2 Using a FIFO to generate the WID signal

You can use a FIFO to generate the **WID** signal. The width of the FIFO is equal to the width of the **AWID** signal. You must set the depth to the outstanding write transaction depth supported by the system.

The MMU-500 supports an outstanding write transaction depth of 128. However, this depth can be limited by the outstanding write transaction depth of the master connected to the MMU-500.

You can generate the **WID** signal by using the following steps:

1. Write the **AWID** signal values to the FIFO when the **AWVALID** and **AWREADY** signals are HIGH.
2. Generate the **WID** signal by using the data from the FIFO that the data read pointer points to.
3. Increment the read pointer when the final write data transfer occurs for the specified address, that is when the **WREADY**, **WVALID**, and **WLAST** signals are HIGH at the same time.

2.5.2 Restrictions for configuration parameters

The MMU-500 has the following configuration restrictions:

- Start the TBU name only with a character.
- You must not duplicate the TBU names.
- If you have selected a common AXI channel between TBU and TCU, then the data widths must be the same, and must be limited to 64-bits or 128-bits.
  - If you select a TLB depth of 256, then there is no possibility for common AXI channel support between TBU0 and TCU (`separate_axi_ptw` is zero).
- When you select full translation support, the following restrictions apply:
  - You must select at least 2 for the `Number of contexts` option.
  - You must select at least 8 for the `Macro TLB depth` option.
- `Number of SMRs` must be greater than or equal to `Number of TBUs`.
- `Number of contexts` must be greater than or equal to Number of SMRs.

Chapter 3
Programmers Model

The MMU-500 requires a programming interface to enable the software to configure the controller. You can also use the programming interface as a debug interface for accessing the TLB details.

This chapter describes the MMU-500 registers and provides information about programming the MMU-500 in the following sections:

3.1 About this programmers model

The following information applies to the MMU-500 registers:

- Registers are implemented according to the *ARM® System Memory Management Unit Architecture Specification* with the security extensions implemented in the MMU-500 as follows:
  - *Global space 0 registers summary* on page 3-9.
  - *Translation context registers summary* on page 3-9.
  - *Integration registers summary* on page 3-10.
  - *Peripheral and component identification summary* on page 3-10.
- Unless otherwise stated in the accompanying text:
  - Do not modify undefined register bits.
  - Ignore undefined register bits on reads.
  - All register values are UNKNOWN on reset unless otherwise stated.
- The access types of the MMU-500 registers are as follows:

RAO	Read-As-One.
RAO/SBOP	Read-As-One, Should-Be-One-or-Preserved on writes.
RAO/WI	Read-As-One, Writes-ignored.
RAZ	Read-As-Zero.
RAZ/SBZP	Read-As-Zero, Should-Be-Zero-or-Preserved on writes.
RAZ/WI	Read-As-Zero, Write-ignored.
RO	Read-only.
RW	Read and write.
SBO	Should-Be-One.
SBOP	Should-Be-One-or-Preserved.
SBZ	Should-Be-Zero.
SBZP	Should-Be-Zero-or-Preserved.
UNK	Unknown.
WI	Write-ignored.
WNR	Write-not-read.
WO	Write-only.

3.1.1 Dynamic programming

The MMU-500 allows dynamic programming as specified by the architecture. See the *ARM® System Memory Management Unit Architecture Specification* for more information.

3.2 Modes of operation and execution

The MMU-500 provides a 64-bit AXI4 programming interface as per the *ARM® AMBA® AXI and ACE Protocol Specification AXI3™, AXI4™, and AXI4-Lite™ ACE and ACE-Lite™*. The interface restrictions are as follows:

- A single combined acceptance depth is used for write and read channels.
- All 32-bit registers write accesses must:
  - Be one beat long.
  - Be used for a word-aligned location.
  - Be 32 bits wide.
  - Have all relevant write strobes set.

  If any of these conditions is not met, the MMU-500 generates an **SLVERR**.
- All 64-bit registers write accesses must satisfy one of the following conditions:
  - Be one beat long, 64 bits wide, used for a double-word aligned location, and have all write stobes set.
  - Be two beats long, 32 bits wide, used for a double-word aligned location, and have all relevant write stobes set.
  - Be one beat long, 32 bits wide, used for a word aligned location, and have all relevant write stobes set.

  If any of these conditions is not met, the MMU-500 generates an SLVERR.
- The AXI4 programming interface ignores the **AxBURST, AxLOCK**, **AxCACHE**, **AxQOS**, **AxREGION**, and **AxUSER** signals.

3.3 Memory model

The address map of the programming interface is consistent with the *ARM® System Memory Management Unit Architecture Specification*.

In addition to the registers specified in the *ARM® System Memory Management Unit Architecture Specification*, the MMU-500 implements the following configuration, identification, debug, context, integration, performance, and control registers:

- Non-secure Auxiliary Configuration Register (SMMU_ACR).
- Secure Auxiliary Configuration Register (SMMU_*s*ACR).
- TBU-TLB Debug Read Pointer register (SMMU_DBGRPTRTBU).
  TBU-TLB Debug Read Data register (SMMU_DBGRDATATBU).
- TCU-TLB Debug Read Pointer register (SMMU_DBGRPTRTCU).
- TCU-TLB Debug Read Data register (SMMU_DBGRDATATCU).
- Auxiliary Control Register (SMMU_CB*n*_ACTLR).
- Integration Mode Control register (SMMU_ITCTRL).
- Integration Test Input register (SMMU_ITIP).
- Integration Test Output Global register (SMMU_ITOP_GLBL).
- TBU Performance Interrupt register (SMMU_ITOP_PERF_INDEX).
- Integration Test Output Context Interrupt registers (SMMU_ITOP_CXT*n*TO*m*_RAM*x)*.
  - Register for contexts 0-31 (SMMU_ITOP_CXT0TO31_RAM0).
  - Register for contexts 32-63 (SMMU_ITOP_CXT32TO63_RAM1).
  - Register for contexts 64-95 (SMMU_ITOP_CXT64TO95_RAM2).
  - Register for contexts 96-127 (SMMU_ITOP_CXT96TO127_RAM3).
- TBU QoS registers (SMMU_TBUQOS*x*).
  - TBU QoS register 0 (SMMU_TBUQOS0).
  - TBU QoS register 1 (SMMU_TBUQOS1).
  - TBU QoS register 2 (SMMU_TBUQOS2).
  - TBU QoS register 3 (SMMU_TBUQOS3).
- Parity Error Checker Register (SMMU_PER).
- Component Identification registers (CID*n*).
  - Component identification register 0 (CID0).
  - Component identification register 1 (CID1).
  - Component identification register 2 (CID2).
  - Component identification register 3 (CID3).
- Peripheral Identification registers (PeriphID*n*).
  - Peripheral Identification register 0 (PeriphID0).
  - Peripheral Identification register 1 (PeriphID1).
  - Peripheral Identification register 2 (PeriphID2).
  - Peripheral Identification register 3 (PeriphID3).
  - Peripheral Identification register 4 (PeriphID4).
  - Peripheral Identification registers 5-7 (PeriphID5-7).

The MMU-500 does not support the following Global Space Invalidation registers for stage 2 configurations:

- SMMU_STLBIALLM.
- SMMU_TLBIVMIDS1.
- SMMU_TLBIVALH.
- SMMU_STLBIVAM.
- SMMU_STLBIVALM.

- SMMU_TLBIVAH.
- SMMU_TLBIALLH.
- SMMU_STLBIALL.

Note

The SMMU_SCR1.NSNUMCBO bit field is RO for `Only stage 2 translations` configurations.

The MMU-500 is configured through a memory-mapped register frame. The total size of the MMU-500 address range depends on the number of implemented translation contexts.

The MMU-500 address map consists of the following equally-sized regions:

The global address space

The global address space is located at the bottom of the MMU-500 address space, at SMMU_BASE. See Figure 3-1.

The translation context bank address space

The translation context bank address space is located above the top of the global address space, at SMMU_TOP. See Figure 3-1.

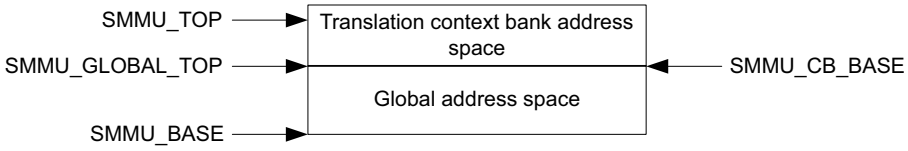


Figure 3-1 MMU-500 address map

You can determine the MMU-500 address range by reading the value of the following register fields:

- SMMU_IDR1.PAGESIZE.
- SMMU_IDR1.NUMPAGENDXB.

See the *ARM® System Memory Management Unit Architecture Specification* for more information.

You can program the context page size as 4KB or 64KB using the SMMU_SACR.PAGESIZE bit. This bit can only be programmed on reset, and can be re-programmed only when the MMU-500 is inactive.

Note

The MMU-500 ignores non-word aligned write accesses to any of the registers.

3.3.1 Reset values

- *MMU-500 Identification registers* on page 3-6.
- *MMU-500 Performance Monitor registers* on page 3-7.

MMU-500 Identification registers

Table 3-1 shows the register field reset values for the MMU-500 Identification registers.

Table 3-1 Reset values of SMMU_IDR registers, both Secure and Non-secure

Register field	Bits[a]	Reset values
SMMU_IDR0		
SMMU_IDR0.SES	[31]	0x1
SMMU_IDR0.S1TS	[30]	**0x0** Stage 1 followed by stage 2 translation. **0x1** Stage 2 translation.
SMMU_IDR0.S2TS	[29]	0x1
SMMU_IDR0.NTS	[28]	**0x0** Stage 1 followed by stage 2 translation. **0x1** Stage 2 translation.
SMMU_IDR0.SMS	[27]	0x1
SMMU_IDR0.ATOSNS	[26]	0x1
SMMU_IDR0.PTFS	[25:24]	**0x0** Stage 1 followed by stage 2 translation. **0x1** Stage 2 translation.
SMMU_IDR0.NUMIRPT[7:0]	[23:16]	`0x01`
SMMU_IDR0.CTTW	[14]	0x0
SMMU_IDR0.BTM	[13]	`0x1`
SMMU_IDR0.NUMSIDB[3:0]	[12:9]	`0xF`
SMMU_IDR0.NUMSMRG[7:0]	[7:0]	`SMR DEPTH`
SMMU_IDR1		
SMMU_IDR1.PAGESIZE	[31]	0x0
SMMU_IDR1.NUMPAGENDXB	[30:28]	The reset values for the following contexts are: **1-8 contexts** 0x2. **9-16 contexts** `0x3`. **17-32 contexts** `0x4`. **33-64 contexts** `0x5`. **64-128 contexts** `0x6`.
SMMU_IDR1.NUMS2CB[7:0]	[23:16]	0x0
SMMU_IDR1.SMCD	[15]	0x0
SMMU_IDR1.SSDTP	[12]	The possible reset values for SMMU_sIDR1.SSDTP (Secure) are: **0x0** Number of SSD entries is zero. **0x1** Otherwise. The reset value for SMMU_IDR1.SSDTP (Non-secure) is 0x0.

SMMU_IDR1.NUMSSDNDXB[3:0]	[11:8]	**For SMMU_sIDR1.NUMSSDNDBB[3:0]**
		`0xF`
		For SMMU_IDR1.NUMSSDNDBB[3:0]
		`0x0`

SMMU_IDR1.NUMCB	[7:0]	Number of contexts.

Table 3-1 Reset values of SMMU_IDR registers, both Secure and Non-secure (continued)

Register field	Bits[a]	Reset values
SMMU_IDR2		
RESERVED	[31:15]	0x0
SMMU_IDR2.PTFSv8_64kB	[14]	0x1
SMMU_IDR2.PTFSv8_16kB	[13]	0x0
SMMU_IDR2.TFSv8_4kB	[12]	0x1
SMMU_IDR2.UBS	[11:8]	0x5
SMMU_IDR2.OAS	[7:4]	0x5
SMMU_IDR2.IAS	[3:0]	0x5
SMMU_IDR7		
SMMU_IDR2.MAJOR	[7:4]	0x0
SMMU_IDR2.MINOR	[3:0]	0x0

a. The Reserved bits are not shown in Table 3-1 on page 3-6. See the *ARM® System Memory Management Unit Architecture Specification* for more information.

MMU-500 Performance Monitor registers

The reset value of the Performance Monitor registers is 0x0, except the registers that Table 3-2 shows. See the *ARM® System Memory Management Unit Architecture Specification* for more information.

Table 3-2 Reset values of Performance Monitor Extension registers

Register field	Bits[a]	Reset values
SMMU_PMCGCR.CGNC	[27:24]	0x4
SMMU_PMCEID0.EVENT	[1:0]	0x1
	[10:8]	0x1
	[18:16]	0x1
SMMU_PMAUTHSTATUS.SNI	[7]	0x1
SMMU_PMDEVTYPE.T	[7:4]	0x5
SMMU_PMDEVTYPE.C	[3:0]	0x6
SMMU_PMCFGR.NCG	[31:24]	Number of TBU - 1
SMMU_PMCFGR.UEN	[19]	0x0
SMMU_PMCFGR.SIDG	[18:17]	0x0
SMMU_PMCFGR.EX	[16]	0x1
SMMU_PMCFGR.CCD	[15]	0x0

Table 3-2 Reset values of Performance Monitor Extension registers (continued)

Register field	Bits[a]	Reset values
SMMU_PMCFGR.CC	[14]	`0x0`
SMMU_PMCFGR.SIZE	[13:8]	`0x1F`
SMMU_PMCFGR.N	[7:0]	(`Number of TBU` * 4) - 1

a. The Reserved bits are not shown in Table 3-2 on page 3-7. See the *ARM® System Memory Management Unit Architecture Specification* for more information.

3.4 Register summary

This section describes the implemented MMU-500 registers in base offset order. The register map contains the following blocks:

- *Global address space 0 registers summary*.
- *Translation context bank registers summary*.
- *Integration registers summary* on page 3-10.
- *Peripheral and component identification registers summary* on page 3-10.

3.4.1 Global address space 0 registers summary

Table 3-3 shows the global space 0 registers in base offset order.

Note

The addresses that are not shown in Table 3-3 are Reserved.

Table 3-3 Global space 0 registers summary

Name	Type	S or NS[a]	Offset	Description	Notes
SMMU_ACR	RW	NS	0x00010	*Auxiliary Configuration registers* on page 3-12	-
SMMU_*s*ACR		S			Banked with security.
SMMU_DBGRPTRTBU	RW	S	0x00080	*Debug registers* on page 3-16	-
SMMU_DBGRDATATBU	RO	S	0x00084		-
SMMU_DBGRPTRTCU	RW	S	0x00088		-
SMMU_DBGRDATATCU	RO	S	0x0008C		-

a. S stands for Secure and NS stands for Non-secure.

3.4.2 Translation context bank registers summary

Table 3-4 shows the context registers in base offset order.

Table 3-4 Translation context registers summary

Name	Type	S or NS[a]	Offset	Description	Notes
SMMU_CB*n*_ACTLR	RW	NS/S	0x004	*Auxiliary Control registers* on page 3-24	-

a. S stands for Secure and NS stands for Non-secure.

3.4.3 Integration registers summary

Table 3-5 shows the integration registers in base offset order.

Table 3-5 Integration registers summary

Name	Type	S or NS	Offset	Description
SMMU_ITCTRL	RW	NS/S	0x2000	*Integration Mode Control Register* on page 3-25
SMMU_ITIP	RO	NS/S	0x2004	*Integration Test Input register* on page 3-26
SMMU_ITOP_GLBL	WO	NS/S	0x2008	*Integration Test Output Global register* on page 3-27
SMMU_ITOP_PERF_INDEX	WO	NS/S	0x200C	*TBU Performance Interrupt register* on page 3-28
SMMU_ITOP_CXT*n*TO*m*_RAM*x* • SMMU_ITOP_CXT0TO31_RAM0. • SMMU_ITOP_CXT32TO63_RAM1. • SMMU_ITOP_CXT64TO95_RAM2. • SMMU_ITOP_CXT96TO127_RAM3.	WO	NS/S	 0x2010 0x2014 0x2018 0x201C	*Integration Test Ouptut Context Interrupt registers* on page 3-31
SMMU_TBUQOS*x* • SMMU_TBUQOS0. • SMMU_TBUQOS1. • SMMU_TBUQOS2. • SMMU_TBUQOS3.	RW	NS/S	 0x2100 0x2104 0x2108 0x210C	*TBU QoS registers* on page 3-32
SMMU_PER	RW	NS/S	0x2200	*Parity Error Checker Register* on page 3-33

3.4.4 Peripheral and component identification registers summary

Table 3-6 shows the peripheral and component identification registers in base offset order.

Table 3-6 Peripheral and component identification summary

Name	Type	S or NS	Offset	Description
PeriphID4	RO	NS/S	0x0FD0	*Peripheral Identification registers* on page 3-34
PeriphID5	RO	NS/S	0x0FD4	
PeriphID6	RO	NS/S	0x0FD8	
PeriphID7	RO	NS/S	0x0FDC	
PeriphID0	RO	NS/S	0x0FE0	
PeriphID1	RO	NS/S	0x0FE4	
PeriphID2	RO	NS/S	0x0FE8	
PeriphID3	RO	NS/S	0x0FEC	

Table 3-6 Peripheral and component identification summary (continued)

Name	Type	S or NS	Offset	Description
CID0	RO	NS/S	0x0FF0	*Component Identification registers* on page 3-34
CID1	RO	NS/S	0x0FF4	
CID2	RO	NS/S	0x0FF8	
CID3	RO	NS/S	0x0FFC	

3.5 Global address space 0

The MMU-500 global address space 0 provides high-level control of the MMU-500 resources, and maps device transactions to translation context banks. This section provides information about the following registers:

- *Auxiliary Configuration registers*.
- *Debug registers* on page 3-16.

3.5.1 Auxiliary Configuration registers

The SMMU_ACR and SMMU_*s*ACR characteristics are:

Purpose The Auxiliary Configuration registers, SMMU_ACR (Non-secure) and SMMU_*s*ACR (Secure), are defined as Table 3-7 on page 3-13 and Table 3-8 on page 3-15 show.

Configuration Available in all MMU-500 configurations.

Usage constraints The S1WC2EN bit is Non-secure only. Other bits are banked with security.

Attributes See *Global address space 0 registers summary* on page 3-9.

Figure 3-2 shows the bit assignments.

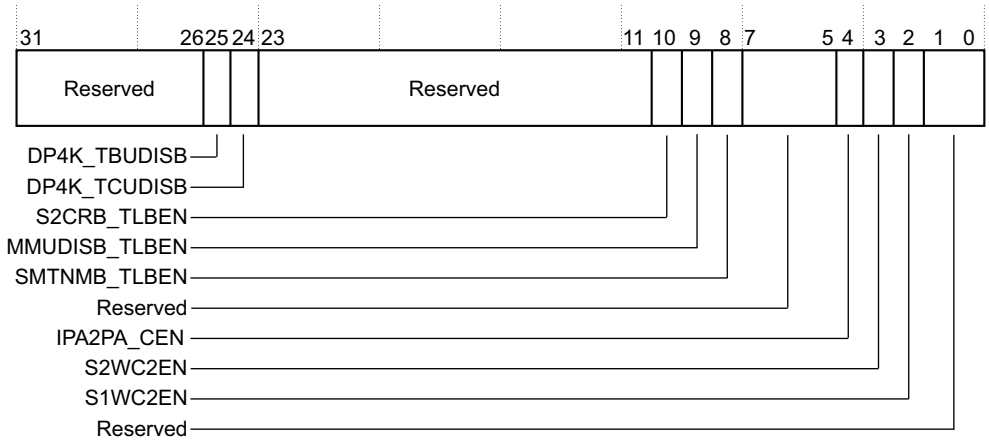


Figure 3-2 SMMU_ACR Register bit assignments

Table 3-7 shows the bit assignments.

Table 3-7 SMMU_ACR Register bit assignments

Bits	Name	Reset value	Description
[31:26]	Reserved	-	Reserved.
[25]	DP4K_TBUDISB	0b0	4KB page size dependency check. Enables or disables the 4KB page size dependency check in the TBU. Transactions form a dependency on a transaction that is already performing the PTW. The preconditions are that they are within the same 4KB address space and have the same stream ID or security state as the transaction performing the PTW. This is the default behavior, which you can change by programming the bits to not set the dependency. This results in extra latencies and power consumption, and is intended only for debug purposes. This bit resets to zero. When it is set to one, this disables the 4KB dependency check in the TBU.
[24]	DP4K_TCUDISB	0b0	4KB page size dependency check. Enables or disables the 4KB page size dependency check in the TCU. This bit resets to zero. When it is set to one, this disables the 4KB dependency check in the TCU.
[23:11]	Reserved	-	Reserved.
[10]	S2CRB_TLBEN	0b0	Stream to context register bypass TLB enable. This bit can have one of the following values: 0b0 Do not update the TLB with the stream to context register bypass transaction information. 0b1 Update the TLB with the stream to context register bypass transaction information. **Note** For the S2CRB_TLBEN, MMUDISB_TLBEN, and SMTNMB_TLBEN bits, the bypass TLB enable bits ensure that the latency is minimal for further transactions that must undergo the same bypass. However, this comes with the penalty of a TLB entry being occupied for the bypass entry, which reduces the effective depth of the TLB available for caching translation operations.
[9]	MMUDISB_TLBEN	0b0	MMU disable bypass TLB enable. The MMU-500 caches in the TLB the attribute information for transactions that have been allocated a context but the SMMU_CB*n*_SCTLR.M bit is set to 0b0 for the context. This caching reduces the transaction time by six clock cycles, but can save much more, depending on how busy the MMU-500 is. This bit can have one of the following values: 0b0 Do not update the TLB with the MMU disable transaction information. 0b1 Update the TLB with the MMU disable transaction information.
[8]	SMTNMB_TLBEN	0b0	Stream match table no match TLB enable. This bit can have one of the following values: 0b0 Do not update the TLB with the stream match table no match TLB enable bypass transaction information. 0b1 Update the TLB with the stream match table no match TLB enable bypass transaction information.
[7:5]	Reserved	-	Reserved.

Table 3-7 SMMU_ACR Register bit assignments (continued)

Bits	Name	Reset value	Description
[4]	IPA2PA_CEN	0b1	IPA to PA cache enable. This bit can have one of the following values: 0b0 Disable the IPA to PA cache. 0b1 Enable the IPA to PA cache. **Note** This bit is Reserved when the Only stage 2 translations option is enabled.
[3]	S2WC2EN	0b1	Stage 2 PTW cache 2 enable. The MMU-500 caches the level 2 PTWs in the stage 2 PTW cache 2. This bit can have one of the following values: 0b0 Disable the PTW cache 2. 0b1 Enable the PTW cache 2.
[2]	S1WC2EN	0b1	Stage 1 PTW cache 2 enable. The MMU-500 caches the level 2 PTW in the stage 1 PTW cache 2. This bit can have one of the following values: 0b0 Disable the PTW cache 2. 0b1 Enable the PTW cache 2.
[1:0]	Reserved	-	Reserved.

Figure 3-3 shows the bit assignments.

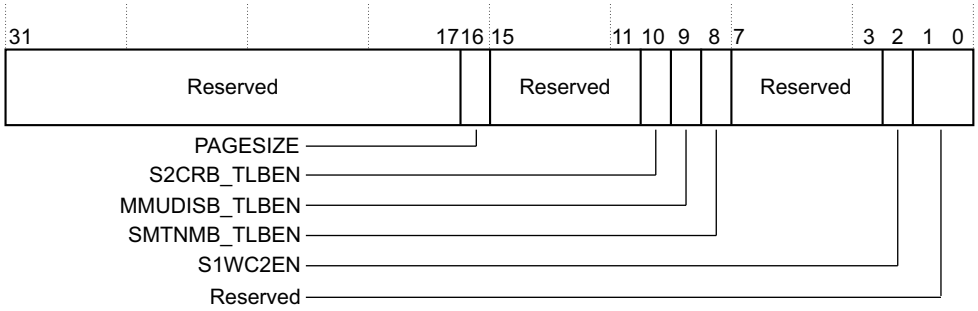


Figure 3-3 SMMU_sACR Register bit assignments

Table 3-8 shows the bit assignments.

Table 3-8 SMMU_sACR Register bit assignments

Bits	Name	Reset value	Description
[31:17]	Reserved	-	Reserved.
[16]	PAGESIZE	0b0	SMMU Page Size. An SMMU register map arranges state into a number of pages. Each page occupies, and is aligned to, a PAGESIZE space in the address map. Such organization permits a hypervisor to permit or deny access to system MMU state on a page-by-page basis. The SMMU architecture permits an implementation to support either 4KB or 64KB PAGESIZE options. This bit can have one of the following values: 0b0 4KB. 0b1 64KB.
[15:11]	Reserved	-	Reserved.
[10]	S2CRB_TLBEN	0b0	Stream to context register bypass TLB enable. This bit can have one of the following values: 0b0 Do not update the TLB with the stream to context register bypass transaction information. 0b1 Update the TLB with the stream to context register bypass transaction information.
[9]	MMUDISB_TLBEN	0b0	MMU disable bypass TLB enable. The MMU-500 caches in the TLB the attribute information for transactions that have been allocated a context but the SMMU_CB*n*_SCTLR.M bit is set to 0b0 for the context. This caching reduces the transaction time by six clock cycles, but could save much more, depending on how busy the MMU-500 is. This bit can have one of the following values: 0b0 Do not update the TLB with the MMU disable transaction information. 0b1 Update the TLB with the MMU disable transaction information.
[8]	SMTNMB_TLBEN	0b0	Stream match table no match TLB enable. This bit can have one of the following values: 0b0 Do not update the TLB with the stream match table no match TLB enable bypass transaction information. 0b1 Update the TLB with the stream match table no match TLB enable bypass transaction information.
[7:3]	Reserved	-	Reserved.
[2]	S1WC2EN	0b1	Stage 1 walk cache 2 enable. The MMU-500 caches the level 2 PTWs in the stage 1 PTW cache 2. You can enable or disable this behavior by using the SMMU_ACR.S1WC2EN bit. This bit can have one of the following values: 0b0 Disable the PTW cache 2. 0b1 Enable the PTW cache 2.
[1:0]	Reserved	-	Reserved.

3.5.2 Debug registers

Purpose

Use the debug AXI4 programming interface for the following information:

- The data stored in the TLB tag. This data is used for transaction matching.
- The physical address and its attributes. These are used for data transactions.

The MMU-500 supports TLB visibility by providing read pointer registers and read data registers to read the values.

On a read access to a TLB data register, the MMU-500 performs the following:

1. Initializes, sets to zero, the read pointer register.
2. Increments the read pointer register by one word, that is, four bytes.
3. Reads the TLB data from the read pointer registers.
4. Returns the TLB data on the AXI4 programming interface.

The read pointer register is writable but the read data register is read-only. The two lower bits of the read pointer registers are RAZ/WI. This ensures that the addresses for a debug TLB fetch are always word aligned.

If the read pointer registers are written with address values that are out-of-bounds of the TLB, the MMU-500 returns an SLVERR error on the AXI4 programming interface. The MMU-500 returns the error on read accesses to the corresponding read data registers.

See *Global space 0 registers summary* on page 3-9.

The following registers define the TLB access mechanism:

SMMU_GR0_BASE+0x80
: TBU-TLB read pointer register - only Secure domain access.

SMMU_GR0_BASE+0x84
: TBU-TLB read data register - only Secure domain access. This 32-bit register contains the part of the TLB pointed to by the read pointer register.

SMMU_GR0_BASE+0x88
: TCU-TLB read pointer register - only Secure domain access.

SMMU_GR0_BASE+0x8C
: TCU-TLB read data register - only Secure domain access. This is a 32-bit register that contains the part of the TLB pointed by the read pointer register.

You can read specific TLB entries by programming the read pointer registers. On an access to the read data register, the MMU-500 returns the TLB entry specified by the read pointer registers and increments the read pointer register. For the next access to the read data register, the MMU-500 reads the read data registers again and returns the next TLB entry.

Note

ARM recommends that you perform a TLB read access when there are no pending transactions. If the TLB read happens concurrently with transactions, the TLB read can return data before or after the data is updated.

The MMU-500 contains the following debug registers:

- *TBU-TLB Debug Read Pointer register*.
- *TCU-TLB Debug Read Pointer register*.
- *Debug Read Data registers* on page 3-18.

Configuration Available in all MMU-500 configurations.

Usage constraints Only Secure access is possible.

Attributes *Global address space 0 registers summary* on page 3-9.

TBU-TLB Debug Read Pointer register

Bits[31:16] are always 0 unless specified by an AXI4 access.

Figure 3-4 shows the bit assignments.

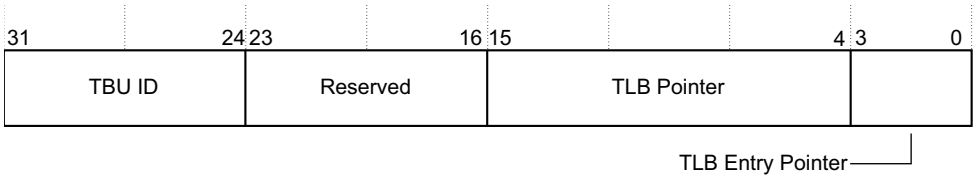


Figure 3-4 SMMU_DBGRPTRTBU register bit assignments

Table 3-9 shows the bit assignments.

Table 3-9 SMMU_DBGRPTRTBU register bit assignments

Bits	Name	Reset value	Description
[31:24]	TBU ID	0x00	TBU identifier. Specifies the TBU from which to read the data. The range of values depends on the number of TBUs configured.
[23:16]	Reserved	-	Reserved.
[15:4]	TLB Pointer	0x000	The pointer to the specified TLB Entry.
[3:0]	TLB Entry Pointer	0x0	Words within the TLB entry.

TCU-TLB Debug Read Pointer register

Bits[31:16] are always 0 unless specified by an AXI4 access.

Figure 3-5 shows the bit assignments.

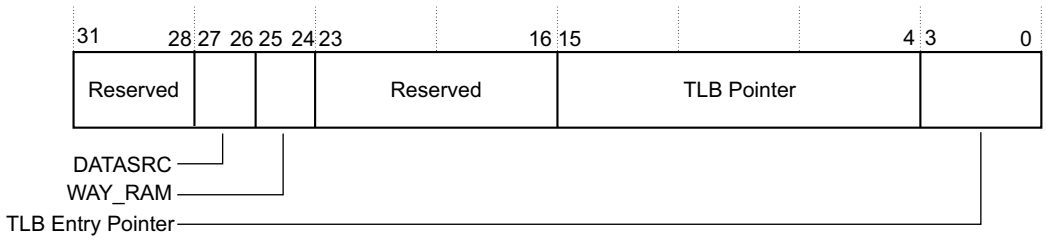


Figure 3-5 SMMU_DBGRPTRTCU register bit assignments

Table 3-10 shows the bit assignments.

Table 3-10 SMMU_DBGRPTRTCU register bit assignments

Bits	Name	Reset value	Description
[31:28]	Reserved	-	Reserved.
[27:26]	DATASRC	-	Specifies the source from which to read the data. This bit field can have one of the following values: 0b00 Macro-TLB. 0b01 Prefetch buffer. 0b10 IPA to PA translation cache. 0b11 PTW cache.
[25:24]	WAY_RAM	0b00	The way in which to connect the four-way set associative cache. This bit field can have one of the following values: 0b00 RAM 1. 0b01 RAM 2. 0b10 RAM 3. 0b11 RAM 4.
[23:16]	Reserved	-	Reserved.
[15:4]	TLB Pointer	0x000	The pointer to the specified TLB entry.
[3:0]	TLB Entry Pointer	0x0	Words within the TLB entry.

Debug Read Data registers

Table 3-11 shows the Debug Read Data register data format, word 0.

Note

If a parity error occurs due to a soft error in a TBU TLB RAM, and if the entry is invalid, the debug read data can be corrupted and it can contain an incorrect value.

Table 3-11 Debug read data register data format, word 0

Bits	Width	Description
[31:4]	28	The virtual address to use for address lookup.

Table 3-11 Debug read data register data format, word 0 (continued)

Bits	Width	Description
[3:2]	2	TLB_ENTRY_VALID. This bit field specifies whether the TLB entry is valid. This bit field can have one of the following values: 0b00 A word that is not the first or the last of the TLB entry. 0b01 First word of the TLB entry. 0b10 Last word of the TLB entry. 0b11 First word of the TLB.
[1]	1	TLB_POINTER_VALID. This bit specifies whether the TLB pointer is valid. This bit can have one of the following values: 0b0 Valid. 0b1 Invalid.
[0]	1	TLB_WORD_INFO. This bit specifies whether the TLB word information is valid. This bit can have one of the following values: 0b0 Valid. 0b1 Invalid.

Table 3-12 shows the Debug Read Data register data format, word 1. For more information, see the *ARM® System Memory Management Unit Architecture Specification*.

Table 3-12 Debug read data register data format, word 1

Bits	Width	Description
[31:16]	16	ASID, Address space identifier.
[15]	1	NSSTATE, Non-secure state.
[14:13]	2	The entry type. This bit field can have one of the following values: 0b00 The translation is enabled. 0b01 The translation is disabled. 0b10 The stage to context register bypass information as programmed in the SMMU_S2CR*n* register. 0b11 The SMMU_CR0.USFCFG bit is set.
[12:4]	9	The virtual address to use for address lookup.
[3:2]	2	TLB_ENTRY_VALID. See Table 3-11 on page 3-18.
[1]	1	TLB_POINTER_VALID. See Table 3-11 on page 3-18.
[0]	1	TLB_WORD_INFO. See Table 3-11 on page 3-18.

Table 3-13 shows the Debug Read Data register data format, word 2.

Table 3-13 Debug read data register data format, word 2

Bits	Width	Description
[31:4]	28	The physical address.
[3:2]	2	TLB_ENTRY_VALID. See Table 3-11 on page 3-18.
[1]	1	TLB_POINTER_VALID. See Table 3-11 on page 3-18.
[0]	1	TLB_WORD_INFO. See Table 3-11 on page 3-18.

Table 3-14 shows the Debug Read Data register data format, word 3.

Table 3-14 Debug read data register data format, word 3

Bits	Width	Description
[31]	1	UCI, User Cache Maintenance Operation Enable. See the *ARM® CoreLink™ MMU-500 System Memory Management Unit Technical Reference Manual Supplement* for more information on the usage of this bit.
[30]	1	MMU-500 Enable. This is the global enable bit for the translation context bank. This bit can have one of the following values: `0b0` The MMU-500 behavior for the translation context bank is disabled. `0b1` The MMU-500 behavior for the translation context bank is enabled.
[29]	1	S2 RW64. See the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.
[28]	1	S1 RW64. See the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.
[27]	1	S1 EAE.See the following documents for more information on LPAE addresses: • *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R editions*. • *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile.*
[26:20]	7	Translation Context Index.
[19]	1	Reserved.
[18:16]	3	Stage 2 Page Size. This bit field can have one of the following values: `0b000` 4KB. `0b001` 64KB. `0b010` Reserved. `0b011` 2MB. `0b100` Reserved. `0b101` 512MB. `0b110` 1GB. `0b111` Reserved.

Table 3-14 Debug read data register data format, word 3 (continued)

Bits	Width	Description
[15:13]	3	Stage 1 Page Size. This bit field can have one of the following values: 0b000 4KB. 0b001 64KB. 0b010 Reserved. 0b011 2MB. 0b100 Reserved. 0b101 512MB. 0b110 1GB. 0b111 Reserved.
[12]	1	Not global. Determines how the translation is marked in the TLB. See the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R editions*.
[11:4]	8	Physical Address.
[3:2]	2	TLB_ENTRY_VALID. See Table 3-11 on page 3-18.
[1]	1	TLB_POINTER_VALID. See Table 3-11 on page 3-18.
[0]	1	TLB_WORD_INFO. See Table 3-11 on page 3-18.

Table 3-15 shows the Debug Read Data register data format, word 4. For more information, see the *ARM® System Memory Management Unit Architecture Specification*.

Table 3-15 Debug read data register data format, word 4

Bits	Width	Description
[31:30]	2	NSCFG, Non-secure configuration.
[29:27]	3	SHCFG, shareability configuration.
[26:25]	2	Inner RACFG, read allocate configuration.
[24:23]	2	Outer RACFG, read allocate configuration.
[22:21]	2	Inner WACFG, write allocate configuration.
[20:19]	2	Outer WACFG, write allocate configuration.
[18]	1	PXN, privilege execute never.
[17]	1	Stage 2 XN, execute never.
[16]	1	Stage 1 XN, execute never.
[15:13]	3	Reserved.
[12:11]	2	HAP, stage 2 access permissions bits.
[10:8]	3	AP, access permissions bits.
[7:6]	2	PRIVCFG, privilege configuration.
[5:4]	2	INSTCFG, instruction configuration.

Table 3-15 Debug read data register data format, word 4 (continued)

Bits	Width	Description
[3:2]	2	TLB_ENTRY_VALID. See Table 3-11 on page 3-18.
[1]	1	TLB_POINTER_VALID. See Table 3-11 on page 3-18.
[0]	1	TLB_WORD_INFO. See Table 3-11 on page 3-18.

Table 3-16 shows the Debug Read Data register data format, word 5. For more information, see the *ARM® System Memory Management Unit Architecture Specification*.

Table 3-16 Debug read data register data format, word 5

Bits	Width	Description
[31:26]	6	Reserved.
[25]	1	Stage 2 Access Flag Enable bit. This bit is UNK/SBOP.
[24]	1	Stage 1 Access Flag Enable bit. This bit is UNK/SBOP.
[23:22]	2	Stage 2 Shared Configuration. Controls the shareable attributes for transactions in which the context bank is disabled. This bit field can have one of the following values: `0b00` Use shareable attribute as specified with transaction. `0b01` Outer shareable. `0b10` Inner shareable. `0b11` Non-shareable.
[21:20]	2	Stage 1 Shared Configuration. Controls the shareable attributes for transactions in which the context bank is disabled. This bit field can have one of the following values: `0b00` Use shareable attribute as specified with transaction. `0b01` Outer shareable. `0b10` Inner shareable. `0b11` Non-shareable.
[19]	1	Memory Endianness.
[18]	1	Carry Condition Flag.
[17:15]	3	*Type EXtension* (TEX), memory attributes. See the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R editions.*
[14]	1	Non-secure state, security status of the master that initiated the current transaction.
[13]	1	The parity bit.
[12:11]	2	Inner TRANSIENTCFG, transient configuration, controls the transient allocation hint.
[10:9]	2	Outer TRANSIENTCFG, transient configuration, controls the transient allocation hint.
[8:4]	5	Memory Attribute. The memory attributes can be overlaid if SMMU_CB*n*_SCTLR.M is set to `0b0`.
[3:2]	2	TLB_ENTRY_VALID. See Table 3-11 on page 3-18.
[1]	1	TLB_POINTER_VALID. See Table 3-11 on page 3-18.
[0]	1	TLB_WORD_INFO. See Table 3-11 on page 3-18.

Table 3-17 shows the Debug Read Data register data format, word 6.

Table 3-17 Debug read data register data format, word 6

Bits	Width	Description
[31:24]	8	Reserved
[23:14]	10	The stream ID mask.
[13:4]	10	The stream ID.
[3:2]	2	TLB_ENTRY_VALID. See Table 3-11 on page 3-18.
[1]	1	TLB_POINTER_VALID. See Table 3-11 on page 3-18.
[0]	1	TLB_WORD_INFO. See Table 3-11 on page 3-18.

Performance monitoring

The MMU-500 supports performance monitoring as explained in the *ARM® System Memory Management Unit Architecture Specification*. One counter group is provided for every TBU that can be used as a global group, as part of a context, or as a stream. The MMU-500 supports four event counters as the global group, and all event classes specified in the *ARM® System Memory Management Unit Architecture Specification*:

- All performance counters exist in the TBUs.
- When performance registers are programmed, the TCU sends the setup information of the counter messages to the TBUs.
- On counter overflows, the TBUs pass the information to the TCU, and the TCU raises an interrupt.
- The TCU can also send a request to the TBUs for the current state of the counters.
- If the counter value is preset, the TCU updates the TBUs.

3.6 Translation context address space

This section describes the translation context bank register present in the MMU-500.

3.6.1 Auxiliary Control registers

The SMMU_CB*n*_ACTLR characteristics are:

Purpose Enable context caching in the macro TLB or prefetch buffer.

Configuration Available in all MMU-500 configurations.

Usage constraints There are no usage constraints.

Attributes See *Translation context bank registers summary* on page 3-9.

Figure 3-6 shows the bit assignments.

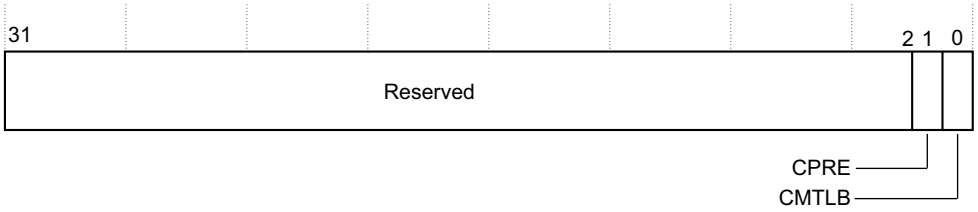


Figure 3-6 SMMU_CBn_ACTLR Registers bit assignments

Table 3-18 shows the bit assignments.

Table 3-18 SMMU_CBn_ACTLR Registers bit assignments

Bits	Name	Reset value	Description
[31:2]	Reserved	-	Reserved.
[1]	CPRE	0b1	Enable context caching in the prefetch buffer.
[0]	CMTLB	0b1	Enable context caching in the macro-TLB.

3.7 Integration registers

This section describes the MMU-500 integration registers in the following sections:

- *Integration Mode Control Register*.
- *Integration Test Input register* on page 3-26.
- *Integration Test Output Global register* on page 3-27.
- *TBU Performance Interrupt register* on page 3-28.
- *Integration Test Ouptut Context Interrupt registers* on page 3-31.
- *TBU QoS registers* on page 3-32.
- *Parity Error Checker Register* on page 3-33.

3.7.1 Integration Mode Control Register

The SMMU_ITCTRL register characteristics are:

Purpose This register enables the component to switch from functional mode to integration mode. You can directly control the inputs and outputs in integration mode.

Note

A device might not operate with the original behavior in integration mode.

After performing integration, you must reset the system to ensure the correct behavior of system components that are affected by the integration.

Writing to this register other than when in the disabled state results in UNPREDICTABLE behavior.

Configuration Available in all MMU-500 configurations.

Usage constraints There are no usage constraints.

Attributes See *Integration registers summary* on page 3-10.

Figure 3-7 shows the bit assignments.

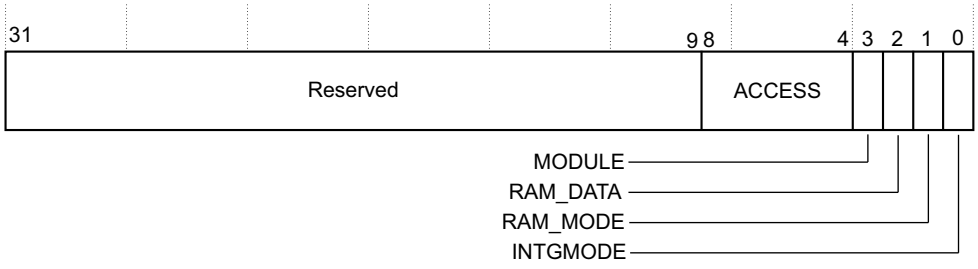


Figure 3-7 SMMU_ITCTRL register bit assignments

Table 3-19 shows the bit assignments.

Table 3-19 SMMU_ITCTRL register bit assignments

Bits	Name	Reset value	Description
[31:9]	Reserved	-	Reserved.
[8:4]	ACCESS	-	Specifies the access information. The functionality of this bit field is determined by the value of the SMMU_ITCTRL.MODULE bit. If the SMMU_ITCTRL.MODULE bit is set to `0b0`, the SMMU_ITCTRL bits[8:5] are ignored, and the SMMU_ITCTRL bit[4] field can have one of the following values: `0b0` The MMU-500 performs IPA to PA translation or accesses the prefetch RAM, sets the RAM mode and accesses it in the direction specified by the value of the RAM WNR bit of the SMMU_*s*GFSYNR0 or SMMU_CB*n*_FSYNR0 register. `0b1` MTLB_WC RAM access. The MMU-500 sets the RAM mode and accesses it in the direction specified by the RAM WNR bit. If the SMMU_ITCTRL.MODULE bit is set to `0b1` (that is, the TBU RAM is specified), this bit field provides the TBU number from which the RAM must read or write. Only log_base_2$^{\text{Number of TBUs}}$ bits are valid.
[3]	MODULE	-	The TBU or TCU RAM. This bit can have one of the following values: `0b0` TCU RAM. Bit[4] provides the TCU RAM information. `0b1` TBU RAM. Bit[4] provides the TBU RAM information.
[2]	RAM_DATA	-	RAM data WNR. This bit can have one of the following values: `0b0` The MMU-500 reads from the RAM with the index specified in the SMMU_ITOP_PERF_INDEX register. `0b1` The MMU-500 writes to the RAM with the index specified in the SMMU_ITOP_PERF_INDEX register.
[1]	RAM_MODE	`0b0`	RAM mode. This bit can have one of the following values: `0b0` The MMU-500 does not drive the RAM bus or read from the RAM. `0b1` The MMU-500 drives the RAM bus or reads from the RAM.
[0]	INTGMODE	`0b0`	Enables the component to switch between functional mode and integration mode. This bit can have one of the following values: `0b0` Disable integration mode. `0b1` Enable integration mode.

3.7.2 Integration Test Input register

The SMMU_ITIP register characteristics are:

Purpose Enables the MMU-500 to read the status of the **spniden** signal.

Configuration Available in all MMU-500 configurations.

Usage constraints There are no usage constraints.

Attributes See *Integration registers summary* on page 3-10.

Figure 3-8 on page 3-27 shows the bit assignments.

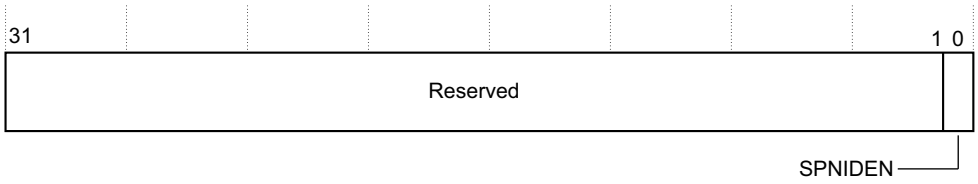


Figure 3-8 SMMU_ITIP register bit assignments

Table 3-20 shows the bit assignments.

Table 3-20 SMMU_ITIP register bit assignments

Bits	Name	Reset value	Description
[31:1]	Reserved	-	Reserved.
[0]	SPNIDEN	-	The Secure debug input, that is the value of the **spniden** signal.

3.7.3 Integration Test Output Global register

The SMMU_ITOP_GLBL register characteristics are:

Purpose Enables the MMU-500 to set the status of the signals that Table 3-21 on page 3-28 shows.

Configuration Available in all MMU-500 configurations.

Usage constraints There are no usage constraints.

Attributes See *Integration registers summary* on page 3-10.

Figure 3-9 shows the bit assignments.

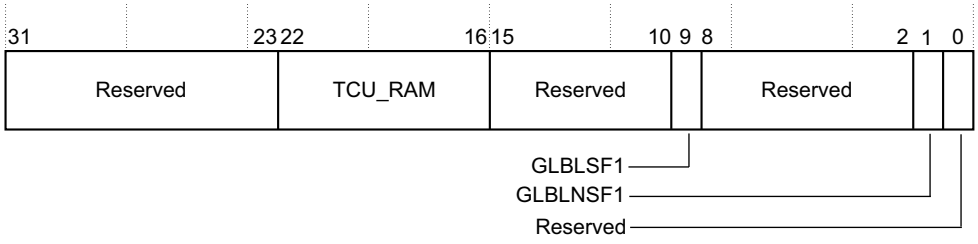


Figure 3-9 SMMU_ITOP_GLBL register bit assignments

Table 3-21 shows the bit assignments.

Table 3-21 SMMU_ITOP_GLBL register bit assignments

Bits	Name	Reset value	Description
[31:23]	Reserved	-	Reserved.
[22:16]	TCU_RAM_DATA	-	The TCU RAM information specified by the *Most-Significant Bits* (MSB) of the RAM RW data. The MSB bits can be one of the following: **[97:91]** For only stage 2 translations. **[129:123]** For stage 1, stage 2, and stage 1 followed by stage 2 (full stage) translations. This RW bit field has a variable width in the range of 1-log_base_2$^{\text{Number of contexts}}$. This bit field is enabled only for TCU RAMs, that is when the SMMU_ITCTRL.RAM_MODE bit is set to `0b1` and the SMMU_ITCTRL.MODULE bit is set to `0b0`. The SMMU_ITCTRL.RAM_DATA bit indicates read or write access information and the access direction.
[15:10]	Reserved	-	Reserved.
[9]	GLBLSFI	-	Global Secure fault interrupt. The value of this bit is equal to the value of the **gbl_flt_irpt_s** signal. This bit can have one of the following values: `0b0` Disable global Secure fault interrupt. `0b1` Enable global Secure fault interrupt.
[8:2]	Reserved	-	Reserved.
[1]	GLBLNSFI	-	Global Non-secure fault interrupt. The value of this bit is equal to the value of the **glbl_flt_irpt_ns** signal. This bit can have one of the following values: `0b0` Disable global Non-secure fault interrupt. `0b1` Enable global Non-secure fault interrupt.
[0]	Reserved	-	Reserved.

3.7.4 TBU Performance Interrupt register

The SMMU_ITOP_PERF_INDEX register characteristics are:

Purpose Enables TBU performance interrupts.

Configuration Available in all MMU-500 configurations.

Usage constraints The values of the RAM_MODE and MODULE bits of the SMMU_ITCTRL register define the behavior of this register, as follows:

- If the SMMU_ITCTRL.RAM_MODE bit is set to `0b0`, the register specifies the TBU interrupt information.
- If the SMMU_ITCTRL.RAM_MODE bit is set to `0b1` and the SMMU_ITCTRL.MODULE bit is set to `0b0`, the register specifies TCU RAM information.
- If the SMMU_ITCTRL.RAM_MODE bit is set to `0b1` and the SMMU_ITCTRL.MODULE bit is set to `0b1`, the register specifies TBU RAM information.

Attributes See *Integration registers summary* on page 3-10.

Figure 3-10 shows the bit assignments when the SMMU_ITCTRL.RAM_MODE bit is set to 0b0.

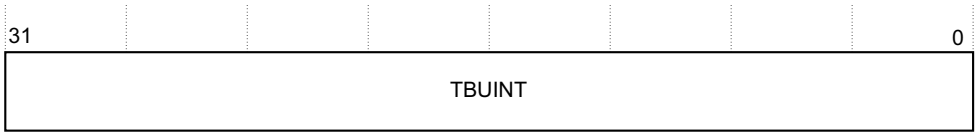


Figure 3-10 SMMU_ITOP_PERF_INDEX register bit assignments-SMMU_ITCTRL.RAM_MODE0

Table 3-22 shows the bit assignments when the SMMU_ITCTRL.RAM_MODE bit is set to 0b0.

Table 3-22 SMMU_ITOP_PERF_INDEX register bit assignments-SMMU_ITCTRL.RAM_MODE0

Bits	Name	Reset value	Description
[31:0]	TBUINT	-	TBU interrupt to enable. This bit field can have one of the following values: **0** Enable performance interrupt for TBU 0. **1** Enable performance interrupt for TBU 1. **...** ... **31** Enable performance interrupt for TBU 31. **Note** You must specify values only for existing TBUs.

Figure 3-11 shows the bit assignments when the SMMU_ITCTRL.RAM_MODE bit is set to 0b1 and the SMMU_ITCTRL.MODULE bit is set to 0b0 to specify TCU RAMs.

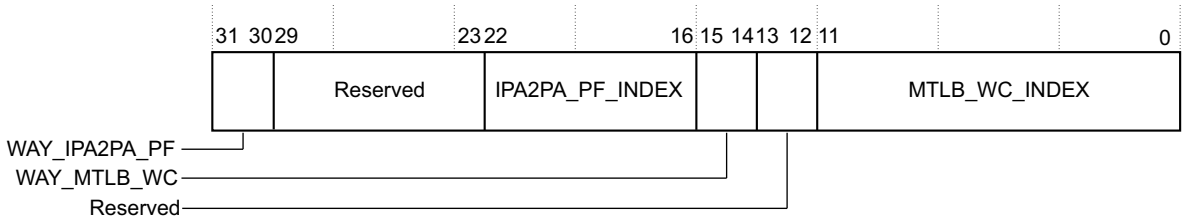


Figure 3-11 SMMU_ITOP_PERF_INDEX register bit assignments-SMMU_ITCTRL.RAM_MODE1.MODULE0

Table 3-23 shows the bit assignments when the SMMU_ITCTRL.RAM_MODE bit is set to 0b1 and the SMMU_ITCTRL.MODULE bit is set to 0b0 to specify TCU RAMs.

Table 3-23 SMMU_ITOP_PERF_INDEX register bit assignments-SMMU_ITCTRL.RAM_MODE1.MODULE0

Bits	Name	Reset value	Description
[31:30]	WAY_IPA2PA_PF	-	The way in which to read or write the IPA to PA translation prefetch RAM. This bit field can have one of the following values: 0b00 Way 0. 0b01 Way 1. 0b10 Way 2. 0b11 Way 3.
[29:23]	Reserved	-	Reserved.
[22:16]	IPA2PA_PF_INDEX	-	The index of the IPA to PA translation prefetch RAM. The number of valid bits depends on the size of the IPA to PA translation cache or the prefetch cache.
[14:15]	WAY_MTLB_WC	-	The way in which to read or write the MTLB_WC RAM. This bit field can have one of the following values: 0b00 Way 0. 0b01 Way 1. 0b10 Way 2. 0b11 Way 3.
[12:13]	Reserved	-	Reserved.
[11:0]	MTLB_WC_INDEX	-	The index of the MTLB_WC RAM. The number of valid bits depends on the size of the macro-TLB and the PTW cache.

Figure 3-11 on page 3-29 shows the bit assignments when the SMMU_ITCTRL.RAM_MODE bit is set to 0b1 and the SMMU_ITCTRL.MODULE bit is set to 0b1 to specify TBU RAMs.

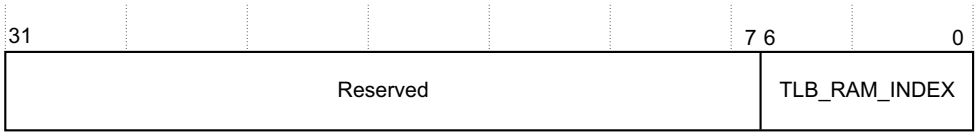


Figure 3-12 SMMU_ITOP_PERF_INDEX register bit assignments-SMMU_ITCTRL.RAM_MODE1.MODULE1

Table 3-23 shows the bit assignments when the SMMU_ITCTRL.RAM_MODE bit is set to 0b1 and the SMMU_ITCTRL.MODULE bit is set to 0b1 to specify TBU RAMs.

Table 3-24 SMMU_ITOP_PERF_INDEX register bit assignments-SMMU_ITCTRL.RAM_MODE1.MODULE1

Bits	Name	Reset value	Description
[31:7]	Reserved	-	Reserved.
[6:0]	TLB_RAM_INDEX	-	The TLB RAM index.

3.7.5 Integration Test Ouptut Context Interrupt registers

The SMMU_ITOP_CXT*n*TO*m*_RAM*x* registers characteristics are:

Purpose Enable the context performance interrupts. The MMU-500 provides the following context performance registers that you can use to select contexts 0-31, 32-63, 64-95, or 96-127:

SMMU_ITOP_CXT0TO31_RAM0
Register for contexts 0-31.

SMMU_ITOP_CXT32TO63_RAM1
Register for contexts 32-63.

SMMU_ITOP_CXT64TO95_RAM2
Register for contexts 64-95.

SMMU_ITOP_CXT96TO127_RAM3
Register for contexts 96-127.

Configuration Available in all MMU-500 configurations.

Usage constraints The value of the SMMU_ITCTRL.RAM_MODE bit defines the behavior of this register, as follows:

- If the bit is set to `0b0`, the register specifies the context interrupt to enable.
- If the bit is set to `0b1`, the register specifies the RAM information.

Attributes See *Integration registers summary* on page 3-10.

Figure 3-13 shows the bit assignments when the SMMU_ITCTRL.RAM_MODE bit is set to `0b0`.

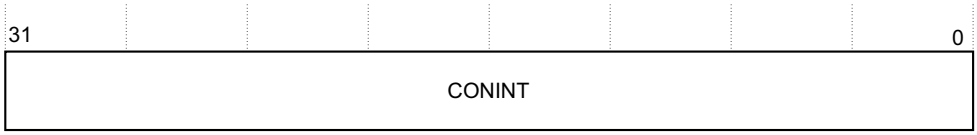


Figure 3-13 SMMU_ITOP_CXTnTOm_RAMx registers bit assignments-SMMU_ITCTRL.RAM_MODE0

Table 3-25 shows the bit assignments when the SMMU_ITCTRL.RAM_MODE bit is set to `0b0`.

Table 3-25 SMMU_ITOP_CXTnTOm_RAMx registers bit assignments-SMMU_ITCTRL.RAM_MODE0

Bits	Name	Reset value	Description
[31:0]	CONINT	-	The context interrupt to be enabled. This WO bit field can have one of the following values: **0** Enable performance interrupt for context *n*. **1** Enable performance interrupt for context n+1. **...** ... **31** Enable performance interrupt for context *m*.

Figure 3-13 shows the bit assignments when the SMMU_ITCTRL.RAM_MODE bit is set to `0b1`.

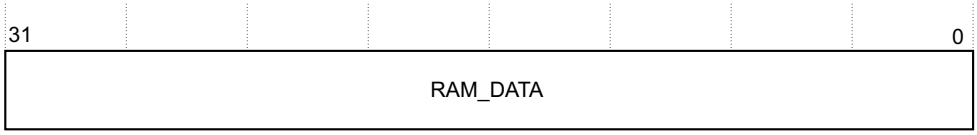


Figure 3-14 SMMU_ITOP_CXTnTOm_RAMx registers bit assignments-SMMU_ITCTRL.RAM_MODE1

Table 3-25 on page 3-31 shows the bit assignments when the SMMU_ITCTRL.RAM_MODE bit is set to `0b1`.

Table 3-26 SMMU_ITOP_CXTnTOm_RAMx registers bit assignments-SMMU_ITCTRL.RAM_MODE1

Bits	Name	Reset value	Description
[31:0]	RAM_DATA	-	The RAM data. The SMMU_ITCTRL.RAM_DATA bit indicates read or write access information and the access direction.

3.7.6 TBU QoS registers

The SMMU_TBUQOS*x* (where x = 0, 1, 2, or 3) registers characteristics are:

Purpose Specifies the QoS for TBUs.

Configuration Available in all MMU-500 configurations.

Usage constraints The types of registers are as follows:

TBU QoS register 0
Used when the TBU*n* is in the range 0-7.

TBU QoS register 1
Used when the TBU*n* is in the range 8-15.

TBU QoS register 2
Used when the TBU*n* is in the range 16-23.

TBU QoS register 3
Used when the TBU*n* is in the range 24-31.

Attributes See *Integration registers summary* on page 3-10.

Figure 3-15 shows the bit assignments.

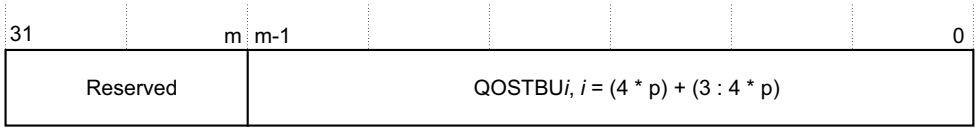


Figure 3-15 SMMU_TBUQOS*x* registers bit assignments

Table 3-27 shows the bit assignments.

Table 3-27 SMMU_TBUQOS*x* registers bit assignments

Bits	Name	Reset value	Description
[31:m]	Reserved	-	Reserved.
[m-1:0]	QOSTBU*i*	0	The QoS for value at index *i* for a TBU, TBU*n*, is calculated by the following equation: • `i = (4 * p+3) : (4*p)` Where, p is number of TBUs. If the value of *n* is between 0-7, then p=*n*. Otherwise, p= (*n*%8).

3.7.7 Parity Error Checker Register

The SMMU_PER characteristics are:

Purpose Checks for parity errors in TCU and TBU RAMs.

Configuration Available in all MMU-500 configurations.

Usage constraints There are no usage constraints.

Attributes See *Integration registers summary* on page 3-10.

Figure 3-16 shows the bit assignments.

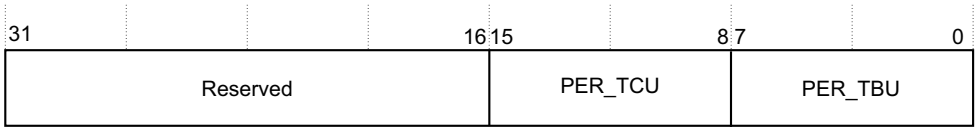


Figure 3-16 SMMU_PER Register bit assignments

Table 3-28 shows the bit assignments.

Table 3-28 SMMU_PER Register bit assignments

Bits	Name	Reset value	Description
[31:16]	Reserved	-	Reserved.
[15:8]	PER_TCU	`0x00`	Parity errors found in TCU RAMs. This bit field saturates after reaching the maximum value.
[7:0]	PER_TBU	`0x00`	Parity errors found in TBU RAMs. This bit field saturates after reaching the maximum value.

3.8 Peripheral and component identification registers

This section describes the following identification registers:

- *Component Identification registers*.
- *Peripheral Identification registers*.

3.8.1 Component Identification registers

The characteristics of the CID registers are:

Purpose Bits[7:0] of the CID 0-3 registers hold preamble information and bits[31:8] are Reserved.

Configuration Available in all MMU-500 configurations.

Usage constraints There are no usage constraints.

Attributes See *Peripheral and component identification registers summary* on page 3-10.

Figure 3-17 shows the bit assignments.

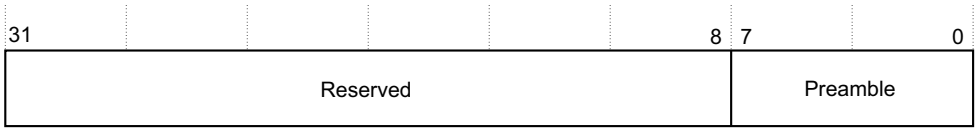


Figure 3-17 CID0-3 registers bit assignments

Table 3-29 shows the bit assignments.

Table 3-29 CID0-3 registers bit assignments

CID	Bits	Name	Reset value	Description
0	[7:0]	Preamble	0x0D	Preamble
1	[7:0]	Preamble	0xF0	Preamble
2	[7:0]	Preamble	0x05	Preamble
3	[7:0]	Preamble	0xB1	Preamble

3.8.2 Peripheral Identification registers

The characteristics of the PeriphID registers are:

Purpose Bits[7:0] of the PeriphID 0-4 registers are used and bits[31:8] are Reserved. The PeriphID 7-5 registers are Reserved.

Configuration Available in all MMU-500 configurations.

Usage constraints There are no usage constraints.

Attributes See *Peripheral and component identification registers summary* on page 3-10.

The peripheral identification registers are as follows:

- *Peripheral Identification register 0*.
- *Peripheral Identification register 1*.
- *Peripheral Identification register 2* on page 3-36.
- *Peripheral Identification register 3* on page 3-36.
- *Peripheral Identification register 4* on page 3-36.
- *Peripheral Identification registers 5-7* on page 3-37.

Peripheral Identification register 0

Figure 3-18 shows the bit assignments.

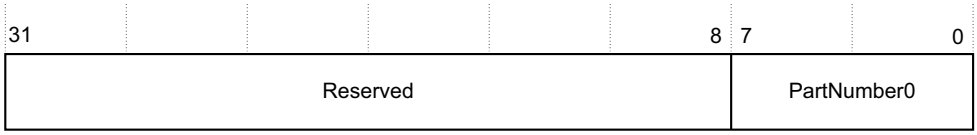


Figure 3-18 PeriphID0 register bit assignments

Table 3-30 shows the bit assignments.

Table 3-30 PeriphID0 register bit assignments

Bits	Name	Reset value	Description
[31:8]	Reserved	-	Reserved.
[7:0]	PartNumber0	0x81	Middle and lower-packed BCD value of the device number [7:0].

Peripheral Identification register 1

Figure 3-19 shows the bit assignments.

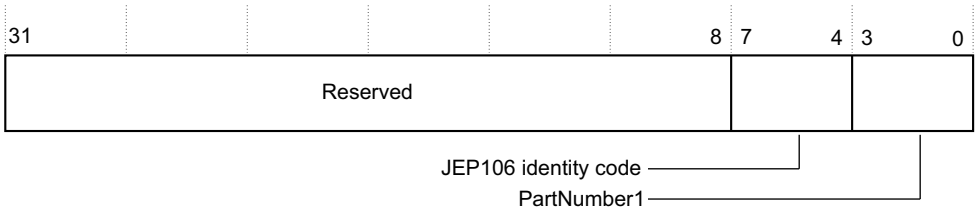


Figure 3-19 PeriphID1 register bit assignments

Table 3-31 shows the bit assignments.

Table 3-31 PeriphID1 register bit assignments

Bits	Name	Reset value	Description
[31:8]	Reserved	-	Reserved.
[7:4]	JEP106 identity code	0xB	JEP106 identity code.
[3:0]	PartNumber1	0x4	Upper packed-BCD value of the device number [11:8].

Peripheral Identification register 2

Figure 3-20 shows the bit assignments.

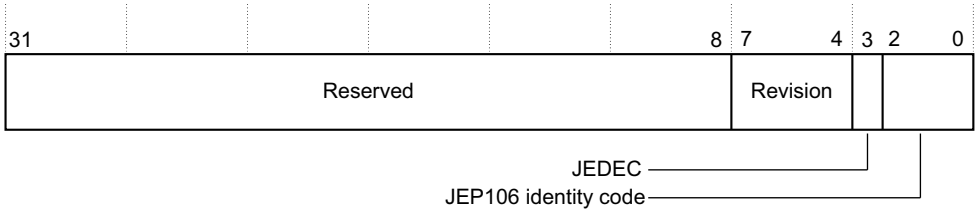


Figure 3-20 PeriphID2 register bit assignments

Table 3-32 shows the bit assignments.

Table 3-32 PeriphID2 register bit assignments

Bits	Name	Reset value	Description
[31:8]	Reserved	-	Reserved.
[7:4]	Revision	0x0	Revision number of the peripheral, which starts from 0x0.
[3]	JEDEC	0x1	Always set, indicates that a JEDEC-assigned value is used.
[2:0]	JEP106 identity code	0x3	JEP106 continuation code, which identifies the designer. The value of 0x3 indicates ARM.

Peripheral Identification register 3

Figure 3-21 shows the bit assignments.

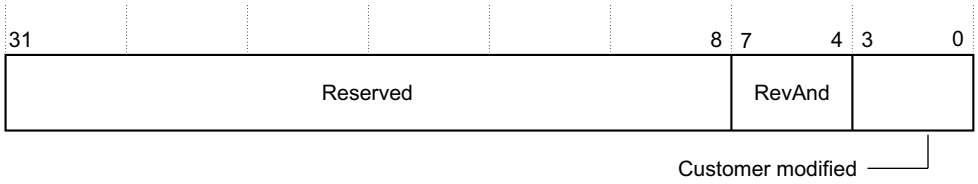


Figure 3-21 PeriphID3 register bit assignments

Table 3-33 shows the bit assignments.

Table 3-33 PeriphID3 register bit assignments

Bits	Name	Reset value	Description
[31:8]	Reserved	-	Reserved.
[7:4]	RevAnd	0x0	Manufacturer revision number. By default, this value is set to 0x0 (specified by ARM).
[3:0]	Customer modified	0x0	Customer modified number.This value is set to 0x0 (specified by ARM).

Peripheral Identification register 4

Figure 3-22 on page 3-37 shows the bit assignments.

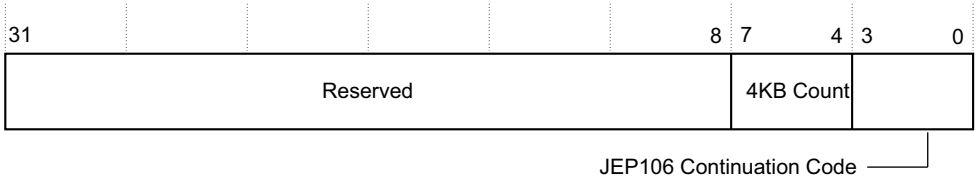


Figure 3-22 PeriphID4 register bit assignments

Table 3-34 shows the bit assignments.

Table 3-34 PeriphID4 register bit assignments

Bits	Name	Reset value	Description
[31:8]	Reserved	-	Reserved.
[7:4]	4KB Count	1-8 contexts: `0x8` 9-16 contexts: `0x9` 17-32 contexts: `0xA` 33-64 contexts: `0xB`	Indicates the log_base_2Number of 4KB blocks occupied by the interface value. The reset value varies with the number of configured contexts.
[3:0]	JEP106 continuation code	`0x4`	JEP106 continuation code, which identifies the designer. The value of `0x4` indicates ARM.

Peripheral Identification registers 5-7

Figure 3-23 shows the bit assignments.

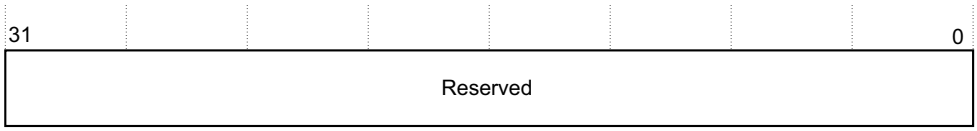


Figure 3-23 PeriphID5-7 register bit assignments

Table 3-35 shows the bit assignments.

Table 3-35 PeriphID5-7 register bit assignments

Bits	Name	Reset value	Description
[31:0]	Reserved	-	Reserved

Appendix A
Signal Descriptions

This appendix describes the MMU-500 signals in the following sections:

A.1 Clock and resets

This section describes the clock and reset signals of the MMU-500.

Table A-1 shows the clock and reset signals of the TCU.

Table A-1 TCU clock and reset signals

Signal	Width	I/O	Description
cclk	1	I	Clock for the TCU.
cresetn	1	I	Reset for the TCU.

Table A-2 shows the clock and reset signals of the TBU.

Table A-2 TBU clock and reset signals

Signal	Width	I/O	Description
<tbuname>_bclkn	1	I	TBU*n* clock, where *n* is a value in the range 0-31. If configured, the clock supplied to TBU0 also clocks the multiplexer between TBU0 and the TCU.
<tbuname>_bresetn	1	I	TBU*n* reset, where *n* is a value in the range 0-31.

A.2 ACE-Lite signals

The *ARM® AMBA® AXI™ and ACE™ Protocol Specification AXI3™, AXI4™, and AXI4-Lite™ ACE™ and ACE-Lite™* describes the AMBA ACE-Lite signals. The following sections describe the ACE-Lite signals:

- *Write address channel signals*.
- *Write data channel signals* on page A-5.
- *Write response channel signals* on page A-6.
- *Read address channel signals* on page A-7.
- *Read data channel signals* on page A-8.
- *Snoop channel signals* on page A-9.

For more information about the output ID width, see *Output ID width* on page 1-14.

The **awuser_<tbuname>_s** and **aruser_<tbuname>_s** input user signals consist of the following:

- Input user-defined bits that are passed as is. This information is stored at bits[(INPUT_AUSER_WIDTH-3):0]
- Input Transient attribute for outer and inner cacheable domains. This information is stored at bits[(INPUT_AUSER_WIDTH-1):(INPUT_AUSER_WIDTH-2)].

  Note

  If the system does not generate this information, you must tie-off bits[(INPUT_AUSER_WIDTH-1):(INPUT_AUSER_WIDTH-2)] to zero.

There is a 4-bit signal addition to the **awuser_<tbuname>_m** and **aruser_<tbuname>_m** input user signals to form the output user signal. Therefore, the output user signals consist of the following parts:

- Output user defined bits, which are same as the input user defined bits. This information is stored at bits[(INPUT_AUSER_WIDTH-3):0].

  Output Transient attribute for outer and inner cacheable domains. This information is stored at bits[(INPUT_AUSER_WIDTH-1):(INPUT_AUSER_WIDTH-2)]. These bits are not the same as the input Transient attribute, but are translated just like other attributes, based on register programming and page tables.

  Note

  If the system does not use the Transient attribute, you can ignore the corresponding output signal.

- Output cache attributes form the inner cacheable domain. The MMU-500 outputs this information at bits[(INPUT_AUSER_WIDTH+3):(INPUT_AUSER_WIDTH)].
  - The page tables provide the cacheability attributes for the outer and inner cacheability domains.
  - The **arcache** and **awcache** signals contain the outer cacheability domain attributes.
  - The MMU-500 appends the inner cacheability domain attributes to the user signal.

A.2.1 Write address channel signals

Table A-3 on page A-4 shows the ACE-Lite write address channel signals for the TBU.

Note

The ***_prog** signals follow the AXI4 protocol, and the TBU signals follow the ACE-Lite protocol.

Table A-3 TBU write address channel signals

ACE-Lite	TBU slave port	Width	I/O	TBU master port	Width	I/O
AWID	**awid_<tbuname>_s**[a]	SIW	I	**awid_<tbuname>_m**[b]	MIW	O
AWADDR	**awaddr_<tbuname>_s**	49	I	**awaddr_<tbuname>_m**	48	O
AWLEN	**awlen_<tbuname>_s**	8	I	**awlen_<tbuname>_m**	8	O
AWSIZE	**awsize_<tbuname>_s**	3	I	**awsize_<tbuname>_m**	3	O
AWBURST	**awburst_<tbuname>_s**	2	I	**awburst_<tbuname>_m**	2	O
AWLOCK	**awlock_<tbuname>_s**	1	I	**awlock_<tbuname>_m**	1	O
AWCACHE	**awcache_<tbuname>_s**	4	I	**awcache_<tbuname>_m**	4	O
AWPROT	**awprot_<tbuname>_s**	3	I	**awprot_<tbuname>_m**	3	O
AWVALID	**awvalid_<tbuname>_s**	1	I	**awvalid_<tbuname>_m**	1	O
AWREGION	**awregion_<tbuname>_s**	4	I	**awregion_<tbuname>_m**	4	O
AWQOS	**awqos_<tbuname>_s**	4	I	**awqos_<tbuname>_m**	4	O
AWSNOOP	**awsnoop_<tbuname>_s**	3	I	**awsnoop_<tbuname>_m**	3	O
AWBAR	**awbar_<tbuname>_s**	2	I	**awbar_<tbuname>_m**	2	O
AWDOMAIN	**awdomain_<tbuname>_s**	2	I	**awdomain_<tbuname>_m**	2	O
AWUSER	**awuser_<tbuname>_s**	(IAUW-2)[c]	I	**awuser_<tbuname>_m**	(IAUW+2)[c]	O
AWREADY	**awready_<tbuname>_s**	1	O	**awready_<tbuname>_m**	1	I

a. The slave ID width, SIW, which is same as the configured `AXI ID signal width`. See Table 1-1 on page 1-10 for more information.

b. The master ID width, MIW, is the calculated output ID width. See *Output ID width* on page 1-14 for more information.

c. The INPUT_AUSER_WIDTH, IAUW. See on page A-3*ACE-Lite signals* on page A-3.

Table A-4 shows the ACE-Lite write address channel signals for the TCU.

Note

AW, **W**, and **B** channels of the PTW interface are not used.

Table A-4 TCU write address channel signals

ACE-Lite	TCU slave port	Width	I/O	TCU master port[a]	Width	I/O
AWID	**awid_prog**	(AXIPID+1)[b]	I	**awid_ptw**	MIW[c]	O
AWADDR	**awaddr_prog**	32	I	**awaddr_ptw**	48	O
AWLEN	**awlen_prog**	8	I	**awlen_ptw**	8	O

Table A-4 TCU write address channel signals (continued)

ACE-Lite	TCU slave port	Width	I/O	TCU master port[a]	Width	I/O
AWSIZE	**awsize_prog**	3	I	**awsize_ptw**	3	O
AWBURST	**awburst_prog**	2	I	**awburst_ptw**	2	O
AWLOCK	**awlock_prog**	1	I	**awlock_ptw**	1	O
AWCACHE	**awcache_prog**	4	I	**awcache_ptw**	4	O
AWPROT	**awprot_prog**	3	I	**awprot_ptw**	3	O
AWVALID	**awvalid_prog**	1	I	**awvalid_ptw**	1	O
AWREGION	**awregion_prog**	4	I	**awregion_ptw**	4	O
AWQOS	**awqos_prog**	4	I	**awqos_ptw**	4	O
AWSNOOP	-	-	-	**awsnoop_ptw**	3	O
AWBAR	-	-	-	**awbar_ptw**	2	O
AWDOMAIN	-	-	-	**awdomain_ptw**	2	O
AWUSER	-	-	-	**awuser_ptw**	6	O
AWREADY	**awready_prog**	1	O	**awready_ptw**	1	I

a. For PTW, the write address channel signals are unused.

b. The AXI programming interface ID signal width, AXIPID, is the `AXI programming interface ID signal width`. See Table 1-1 on page 1-10 for more information.

c. The master ID width, MIW, is the calculated output ID width. See *Output ID width* on page 1-14 for more information.

A.2.2 Write data channel signals

Table A-5 shows the ACE-Lite write data channel signals for the TBU.

Table A-5 TBU write data channel signals

ACE-Lite	TBU slave port	Width	I/O	TBU master port	Width	I/O
WDATA	**wdata_<tbuname>_s**	(WDW+1)[a]	I	**wdata_<tbuname>_m**	(WDW+1)	O
WSTRB	**wstrb_<tbuname>_s**	(WSW+1)[b]	I	**wstrb_<tbuname>_m**	(WSW+1)	O
WLAST	**wlast_<tbuname>_s**	1	I	**wlast_<tbuname>_m**	1	O
WVALID	**wvalid_<tbuname>_s**	1	I	**wvalid_<tbuname>_m**	1	O
WUSER	**wuser_<tbuname>_s**	WUSER[c]	I	**wuser_<tbuname>_m**	WUSER	O
WREADY	**wready_<tbuname>_s**	1	O	**wready_<tbuname>_m**	1	I

a. The write data width, WDW, which is same as the configured `AXI data bus width` parameter. See Table 1-1 on page 1-10 for more information.

b. The write strobe width, WDW, which is 1/8 times the configured `AXI data bus width` parameter. See Table 1-1 on page 1-10 for more information.

c. The width of the **wuser** signal, specified by the `Width of the AXI slave interface WUSER signals` parameter. See Table 1-1 on page 1-10 for more information.

Table A-6 shows the ACE-Lite write data channel signals for the TCU.

Table A-6 TCU write data channel signals

ACE-Lite	TCU slave port	Width	I/O	TCU master port[a]	Width	I/O
WDATA	**wdata_prog**	64	I	**wdata_ptw**	(WDW+1)[b]	O
WSTRB	**wstrb_prog**	8	I	**wstrb_ptw**	(WSW+1)[c]	O
WLAST	**wlast_prog**	1	I	**wlast_ptw**	1	O
WVALID	**wvalid_prog**	1	I	**wvalid_ptw**	1	O
WUSER	-	-	-	**wuser_ptw**	WUSER	O
WREADY	**wready_prog**	1	O	**wready_ptw**	1	I

a. For PTW, the write data channel signals are unused.

b. The write data width, WDW, which is same as the configured `write data width parameter`. See Table 1-1 on page 1-10 for more information.

c. The write strobe width, WDW, which is 1/8 times the configured `write data width` parameter. See Table 1-1 on page 1-10 for more information.

A.2.3 Write response channel signals

Table A-7 shows the ACE-Lite write response channel signals for the TBU.

Table A-7 TBU write response channel signals

ACE-Lite	TBU slave port	Width	I/O	TBU master port	Width	I/O
BID	**bid_<tbuname>_s**	SIW[a]	O	**bid_<tbuname>_m**	MIW[b]	I
BRESP	**bresp_<tbuname>_s**	2	O	**bresp_<tbuname>_m**	2	I
BVALID	**bvalid_<tbuname>_s**	1	O	**bvalid_<tbuname>_m**	1	I
BUSER	**buser_<tbuname>_s**	BUSER[c]	O	**buser_<tbuname>_m**	BUSER	I
BREADY	**bready_<tbuname>_s**	1	I	**bready_<tbuname>_m**	1	O

a. The slave ID width, SIW, which is same as the configured `AXI ID signal width`. See Table 1-1 on page 1-10 for more information.

b. The master ID width, MIW, is the calculated output ID width. See *Output ID width* on page 1-14 for more information.

c. The width of the **buser** signal, specified by the `Width of the AXI slave interface BUSER signals` parameter. See Table 1-1 on page 1-10 for more information.

Table A-7 shows the ACE-Lite write response channel signals for the TCU.

Table A-8 TCU write response channel signals

ACE-Lite	TCU slave port	Width	I/O	TCU master port[a]	Width	I/O
BID	**bid_prog**	(AXIPID+1)[b]	O	**bid_ptw**	MIW[c]	I
BRESP	**bresp_prog**	2	O	**bresp_ptw**	2	I

Table A-8 TCU write response channel signals (continued)

ACE-Lite	TCU slave port	Width	I/O	TCU master port[a]	Width	I/O
BVALID	**bvalid_prog**	1	O	**bvalid_ptw**	1	I
BUSER	-	-	-	**buser_ptw**	BUSER	I
BREADY	**bready_prog**	1	I	**bready_ptw**	1	O

a. For PTW, the write response channel signals are unused.

b. The AXI programming interface ID signal width, AXIPID, is the `AXI programming interface ID signal width parameter`. See Table 1-1 on page 1-10 for more information.

c. The master ID width, MIW, is the calculated output ID width. See *Output ID width* on page 1-14 for more information.

A.2.4 Read address channel signals

Table A-9 shows the ACE-Lite read address channel signals for the TBU.

Table A-9 TBU read address channel signals

ACE-Lite	TBU slave port	Width	I/O	TBU master port	Width	I/O
ARID	**arid_<tbuname>_s**	SIW[a]	I	**arid_<tbuname>_m**	MIW[b]	O
ARADDR	**araddr_<tbuname>_s**	49	I	**araddr_<tbuname>_m**	48	O
ARLEN	**arlen_<tbuname>_s**	8	I	**arlen_<tbuname>_m**	8	O
ARSIZE	**arsize_<tbuname>_s**	3	I	**arsize_<tbuname>_m**	3	O
ARBURST	**arburst_<tbuname>_s**	2	I	**arburst_<tbuname>_m**	2	O
ARLOCK	**arlock_<tbuname>_s**	1	I	**arlock_<tbuname>_m**	1	O
ARCACHE	**arcache_<tbuname>_s**	4	I	**arcache_<tbuname>_m**	4	O
ARPROT	**arprot_<tbuname>_s**	3	I	**arprot_<tbuname>_m**	3	O
ARVALID	**arvalid_<tbuname>_s**	1	I	**arvalid_<tbuname>_m**	1	O
ARREGION	**arregion_<tbuname>_s**	4	I	**arregion_<tbuname>_m**	4	O
ARQOS	**arqos_<tbuname>_s**	4	I	**arqos_<tbuname>_m**	4	O
ARSNOOP	**arsnoop_<tbuname>_s**	4	I	**arsnoop_<tbuname>_m**	4	O
ARBAR	**arbar_<tbuname>_s**	2	I	**arbar_<tbuname>_m**	2	O
ARDOMAIN	**ardomain_<tbuname>_s**	2	I	**ardomain_<tbuname>_m**	2	O
ARUSER	**aruser_<tbuname>_s**	(IAUW-2)[c]	I	**aruser_<tbuname>_m**	(IAUW+2)[c]	O
ARREADY	**arready_<tbuname>_s**	1	O	**arready_<tbuname>_m**	1	I

a. The slave ID width, SIW, which is same as the configured `AXI ID signal width` parameter. See Table 1-1 on page 1-10 for more information.

b. The master ID width, MIW, is the calculated output ID width. See *Output ID width* on page 1-14 for more information.

c. The INPUT_AUSER_WIDTH, IAUW. See *ACE-Lite signals* on page A-3 for more information.

Table A-9 on page A-7 shows the ACE-Lite read address channel signals for the TCU.

Table A-10 TCU read address channel signals

ACE-Lite	TCU slave port	Width	I/O	TCU master port	Width	I/O
ARID	**arid_prog**	(AXIPID+1)[a]	I	**arid_ptw**	MIW[b]	O
ARADDR	**araddr_prog**	32	I	**araddr_ptw**	48	O
ARLEN	**arlen_prog**	8	I	**arlen_ptw**	8	O
ARSIZE	**arsize_prog**	3	I	**arsize_ptw**	3	O
ARBURST	**arburst_prog**	2	I	**arburst_ptw**	2	O
ARLOCK	**arlock_prog**	1	I	**arlock_ptw**	1	O
ARCACHE	**arcache_prog**	4	I	**arcache_ptw**	4	O
ARPROT	**arprot_prog**	3	I	**arprot_ptw**	3	O
ARVALID	**arvalid_prog**	1	I	**arvalid_ptw**	1	O
ARREGION	**arregion_prog**	4	I	**arregion_ptw**	4	O
ARQOS	**arqos_prog**	4	I	**arqos_ptw**	4	O
ARSNOOP	-	-	-	**arsnoop_ptw**	4	O
ARBAR	-	-	-	**arbar_ptw**	2	O
ARDOMAIN	-	-	-	**ardomain_ptw**	2	O
ARUSER	-	-	-	**aruser_ptw**	6[c]	O
ARREADY	**arready_prog**	1	O	**arready_ptw**	1	I

a. The AXI programming interface ID signal width, AXIPID, is the `AXI programming interface ID signal width` parameter. See Table 1-1 on page 1-10 for more information.

b. The master ID width, MIW, is the calculated output ID width. See *Output ID width* on page 1-14 for more information.

c. The bit assignments are as follows:

[5:2] Inner cache attributes for the PTW.

[1] Outer Transient attribute for the PTW.

[0] Inner Transient attribute for the PTW.

A.2.5 Read data channel signals

Table A-11 shows the ACE-Lite read data channel signals for the TBU.

Table A-11 TBU read data channel signals

ACE-Lite	TBU slave port	Width	I/O	TBU master port	Width	I/O
RID	**rid_<tbuname>_s**[a]	SIW	O	**arid_<tbuname>_m**[b]	MIW	I
RDATA	**rdata_<tbuname>_s**[c]	WDW	I	**rdata_<tbuname>_m**	WDW	I
RRESP[d]	**rresp_<tbuname>_s**	2	O	**rresp_<tbuname>_m**	4	I
RLAST	**rlast_<tbuname>_s**	1	O	**rlast_<tbuname>_m**	1	I

Table A-11 TBU read data channel signals (continued)

ACE-Lite	TBU slave port	Width	I/O	TBU master port	Width	I/O
RVALID	**rvalid_<tbuname>_s**	1	O	**rvalid_<tbuname>_m**	1	I
RUSER	**ruser_<tbuname>_s**	RUSER[e]	O	**ruser_<tbuname>_m**	RUSER	I
RREADY	**rready_<tbuname>_s**	1	I	**rready_<tbuname>_m**	1	O

a. The slave ID width, SIW, which is same as the configured `AXI ID signal width`. See Table 1-1 on page 1-10 for more information.

b. The master ID width, MIW, is the calculated output ID width. See *Output ID width* on page 1-14 for more information.

c. The read data width, RDW, which is same as the configured `write data width` parameter. See Table 1-1 on page 1-10 for more information.

d. In the ACE-Lite specification, the **RRESP** signal is two bits wide. However, when a shared interface is used in the MMU-500 to enable DVM operation, the ACE protocol definition is used to include AC and CR signals. As a result, the **RRESP** signal is increased in size by two bits, that is [3:2]. Bit[3] and bit[2] are not on ACE-Lite interfaces, so you can tie the **RRESP[3:2]** signal to `0x0`.

e. The width of the **RUSER** signal, specified by the `Width of the AXI slave interface RUSER signals` parameter. See Table 1-1 on page 1-10 for more information.

Table A-12 shows the ACE-Lite read data channel signals for the TCU.

Table A-12 TCU read data channel signals

ACE-Lite	TCU slave port	Width	I/O	TCU master port	Width	I/O
RID	**arid_prog**	(AXIPID+1)[a]	I	**arid_ptw**	MIW[b]	I
RDATA	**rdata_prog**	64	O	**rdata_ptw**	128	I
RRESP[c]	**rresp_prog**	2	O	**rresp_ptw**	4	I
RLAST	**rlast_prog**	1	O	**rlast_ptw**	1	I
RVALID	**rvalid_prog**	1	O	**rvalid_ptw**	1	I
RUSER	-	-	-	**ruser_ptw**	RUSER	I
RREADY	**rready_prog**	1	I	**rready_ptw**	1	O

a. The AXI programming interface ID signal width, AXIPID, is the `AXI programming interface ID signal width` parameter. See Table 1-1 on page 1-10 for more information.

b. The master ID width, MIW, is the calculated output ID width. See *Output ID width* on page 1-14 for more information.

c. In the ACE-Lite specification, the **RRESP** signal is two bits wide. However, when a shared interface is used in the MMU-500 to enable DVM operation, the ACE protocol definition is used to include AC and CR signals. As a result, the **RRESP** signal is increased in size by two bits, that is [3:2]. Bit[3] and bit[2] are not on ACE-Lite interfaces, so you can tie the **RRESP[3:2]** signal to `0x0`.

A.2.6 Snoop channel signals

Table A-13 on page A-10 shows the ACE-Lite snoop channel signals.

Table A-13 Snoop channel signals

ACE-Lite	Signal	Width	I/O	Description
Snoop address channel signals				
ACADDR	**acaddr_<tbuname>_m**	44	I	Snoop address.
ACPROT	**acprot_<tbuname>_m**	3	I	Snoop protection information.
ACVALID	**acvalid_<tbuname>_m**	1	I	Valid signal for the snoop address channel.
ACSNOOP	**acsnoop_<tbuname>_m**	4	I	Snoop transaction type.
ACREADY	**acready_<tbuname>_m**	1	O	Ready signal for the snoop address channel.
Snoop response channel signals				
CRRESP	**crresp_<tbuname>_m**	5	O	Snoop response.
CRVALID	**crvalid_<tbuname>_m**	1	O	Valid signal for the snoop response channel.
CRREADY	**crready_<tbuname>_m**	1	I	Ready signal for the snoop response channel.

A.3 Low-power interface signals

Table A-14 shows the standard TCU LPI signals.

Table A-14 TCU LPI signals

LPI signal	TCU block signal	Width	Direction
QREQn	**qreqn_tcu**	1	I
QACTIVE	**qactive_tcu**	1	O
QACCEPTn	**qacceptn_tcu**	1	O

Table A-15 shows the standard TBU LPI signals.

Table A-15 TBU LPI signals

LPI signal	TBU block signal	Width	Direction
QREQn	**qreqn_tbu_<tbuname>_pd**	1	I
	qreqn_tbu_<tbuname>_cg	1	
	qreqn_pd_slv_br_<tbuname>	1	
	qreqn_pd_mst_br_<tbuname>	1	
QACCEPTn	**qacceptn_tbu_<tbuname>_pd**	1	O
	qacceptn_tbu_<tbuname>_cg	1	
	qacceptn_pd_slv_br_<tbuname>	1	
	qacceptn_pd_mst_br_<tbuname>	1	
QACTIVE	**qactive_tbu_<tbuname>_cg**	1	O
	qactive_br_tbu_<tbuname>	1	
	qactive_br_tcu_<tbuname>	1	

Table A-16 shows the **awakeup** signals. See the *ARM® CoreLink™ MMU-500 System Memory Management Unit Integration Manual* for more information.

The MMU-500 ensures that the **awakeup** signal outputs are driven from a flip-flop and along with a transaction **axvalid**, or for multiplexed configurations the awakeup signal can be delayed by one cycle with respect to an asserted **axvalid** to ensure it is a registered output.

Table A-16 awakeup signals

Signal	Width	I/O	Description	Block
awakeup_<tbuname>_s	1	I	Indicates that one of address read, address write, or write data channels have active transactions pending at the TBU slave interface.	TBU
awakeup_<tbuname>_m	1	O	Indicates that one of address read, address write, or write data channels have active transactions pending at the TBU master interface	TBU
awakeup_ptw	1	O	Indicates that a PTW transaction is present at the TCU master interface. This signal is present only when you specify a dedicated PTW channel.	TCU

Table A-16 awakeup signals (continued)

Signal	Width	I/O	Description	Block
awakeup_dvm_ptw	1	I	Indicates that a DVM transaction is present on the TCU AC channel. This signal is present only when you specify a dedicated PTW channel.	TCU
awakeup_dvm_<tbuname>	1	I	Indicates that a DVM transaction is present on the TBU AC channel. This signal is present only when you do not specify a dedicated PTW channel.	TBU
awakeup_prog	1	I	Indicates that one of address read, address write, or write data channels have active transactions pending at the TCU programming interface.	TCU

A.4 Miscellaneous signals

This section describes the non-AMBA signals as follows:

- *Sideband signals*.
- *Interrupt signals*.
- *Authentication interface signal* on page A-14.
- *Tie-off signals* on page A-14.
- *Performance event signals* on page A-14.

A.4.1 Sideband signals

Table A-17 shows the sideband signals.

Table A-17 Sideband signals

Signal	I/O	Width	Description
rsb_ns_<tbuname>_s	I	1	Determines the Non-secure state of an incoming read transaction. The value of this signal depends on the **arvalid** signal.
wsb_ns_<tbuname>_s	I	1	Determines the Non-secure state of an incoming write transaction. The value of this signal depends on the **awvalid** signal.
wsb_ssd_<tbuname>_s	I	10	Sideband signal to indicate the SSD index. If the **rsb_ns** or **wsb_ns** signal exists, then this signal does not exist. The value of this signal depends on the **awvalid** signal.
rsb_ssd_<tbuname>_s	I	10	Sideband signal to indicate the SSD index. If the **rsb_ns** or **wsb_ns** signal exists, then this signal does not exist. The value of this signal depends on the **arvalid** signal.
wsb_sid_<tbuname>_s	I	1-15	Sideband signal to indicate the write stream ID. The value of this signal depends on the **awvalid** signal.
rsb_sid_<tbuname>_s	I	1-15	Sideband signal to indicate the read stream ID. The value of this signal depends on the **arvalid** signal.

A.4.2 Interrupt signals

Table A-18 shows the interrupt signals generated by the MMU-500. See the *ARM® System Memory Management Unit Architecture Specification* for more information.

Table A-18 Interrupt signals

Signal	I/O	Width	Description
gbl_flt_irpt_s	O	1	Global Secure fault interrupt.
gbl_flt_irpt_ns	O	1	Global Non-secure fault interrupt.
perf_irpt_<tbuname>	O	1	Performance counter interrupt, one for every TBU.
cxt_irpt_<SMMU_IDR1.NUMCB-1:0>	O	1	Non-secure context interrupts for 0-(`NUM_CONTEXT` - 1), where `NUM_CONTEXT` is the number of contexts.
comb_irpt_ns	O	1	Non-secure combined interrupt. This combined interrupt is the logical OR of **gbl_flt_irpt_ns**, **perf_irpt_<tbuname>**, and **cxt_irpt_[(SCR1.NSNUMCBO-1):0]**.
comb_irpt_s	O	1	Secure combined interrupt. This combined interrupt is the logical OR of **gbl_flt_irpt_s** and **cxt_irpt_[(SMMU_IDR1.NUMCB-1):(SMMU_SCR1.NSNUMCBO]**.

A.4.3 Authentication interface signal

The authentication interface disables AXI accesses. Table A-19 shows the authentication interface signal. See the *ARM® CoreSight™ Architecture Specification* for more information.

Table A-19 Authentication Interface signal

Signal	I/O	Width	Description
spniden	I	1	Secure privileged non-invasive debug enable. When the **spniden** signal is high, it enables counting of the Secure events. You can specify one of the following values: `0b0` Do not count Secure events in the performance counters. `0b1` Count Secure events in the performance counters.
arqosarb	I	4	Highest QoS value of all read transactions in the MMU-500, including the ones on the bus.

A.4.4 Tie-off signals

Table A-20 shows the tie-off signals.

Table A-20 Tie-off signals

Signal	I/O	Width	Description
cfg_cttw	I	1	Static configuration to indicate whether the MMU-500 performs coherent PTWs. This signal cannot change after reset.
dftclkenable	I	1	When this signal is HIGH, the MMU-500 bypasses architectural clock gates. This signal is used in the DFT test mode. You can specify one of the following values: `0b0` Functional mode. `0b1` Bypass architectural clock gates in the DFT test mode.
integ_sec_override	I	1	When this signal is set, Non-secure accesses can access the integration registers. See *Integration registers* on page 3-25.
sysbardisable_<tbuname>	I	0	Indicates that the master or slave connected to the MMU-500 is the AXI3 interface. It is assumed that no barriers are generated at input or output.

A.4.5 Performance event signals

Table A-21 shows the performance event signals.

Table A-21 Performance event signals

Signal	I/O	Width	Description
event_clk_<tbuname>	O	1	Event of every TBU clock.
event_clk64_<tbuname>	O	1	Event for every 64th TBU clock.
event_wr_access_<tbuname>	O	1	Event of every write access that passes through the TBU.
event_rd_access_<tbuname>	O	1	Event of every read access that passes through the TBU.
event_wr_refill_<tbuname>	O	1	Event of the allocation in the TLB due to a write access in the corresponding TBU.
event_rd_refill_<tbuname>	O	1	Event of the allocation in the TLB due to a read access in the corresponding TBU.

Appendix B
Revisions

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

Table B-1 Issue A

Change	Location	Affects
No changes, first release	-	-