Arm® CoreLink™ MMU-600AE System Memory Management Unit

Revision: r1p0

Technical Reference Manual

arm

101412_0100_00_en

Arm® CoreLink™ MMU-600AE System Memory Management Unit

Technical Reference Manual

Release Information

Document History

Issue	Date	Confidentiality	Change
0000-00	14 December 2018	Confidential	First release for r0p0 BET.
0000-01	07 March 2019	Non-Confidential	First release for r0p0 EAC.
0100-00	31 July 2020	Non-Confidential	First release for r1p0 REL.

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Additional Notices

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

Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

Product Status

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

Web Address

developer.arm.com

Contents

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

Preface

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

It contains the following:

About this book

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

Product revision status

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

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

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

Intended audience

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

Using this book

This book is organized into the following chapters:

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

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

Chapter 3 Programmer's model
This chapter describes the MMU-600AE programmer's model.

Chapter 4 Functional Safety
This chapter describes the *Functional Safety* (FuSa) detection features unique to MMU-600AE.

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

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

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

Glossary

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

See the *Arm® Glossary* for more information.

Typographic conventions

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

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

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

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

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

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

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

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

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

Timing diagrams

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

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

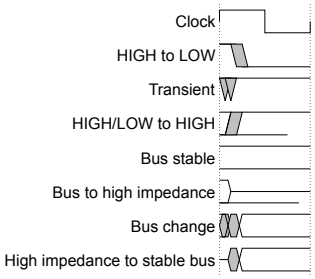

Figure 1 Key to timing diagram conventions

Signals

The signal conventions are:

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

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

Additional reading

Arm publications

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

- *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3.0 and version 3.1* (IHI 0070).
- *Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile* (DDI 0487).
- *Arm® AMBA® Distributed Translation Interface (DTI) Protocol Specification* (100225).
- *Arm® AMBA® APB Protocol Specification* (IHI 0024).
- *Arm® AMBA® AXI and ACE Protocol Specification, AXI3, AXI4, AXI5, ACE and ACE5* (IHI 0022).
- *Arm® AMBA® 4 AXI4-Stream Protocol Specification* (IHI 0051).
- *Arm® AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces* (IHI 0068).
- *Arm® CoreLink™ LPD-500 Low Power Distributor Technical Reference Manual* (100361).
- *Arm® Server Base System Architecture* (DEN 0029).
- *Arm® Reliability, Availability, and Serviceability (RAS) Architecture Extension* (PRD03-PRDC-010953).
- *AMBA® Interface Parity Specification AHB, APB and AXI4-Stream* (AES 0010).

The following confidential books are only available to licensees:

- *Arm® CoreLink™ MMU-600AE System Memory Management Unit Configuration and Integration Manual* (101413).
- *Arm® CoreLink™ MMU-600AE System Memory Management Unit Safety Manual* (101414).
- *Arm® CoreLink™ MMU-600AE System Memory Management Unit Development Interface Report* (101415).
- *Arm® CoreLink™ MMU-600AE System Memory Management Unit FMEDA Report* (PJDOC-1779577084-12336).
- *Arm® CoreLink™ MMU-600AE System Memory Management Unit Dependent Failure Analysis Report* (PJDOC-1779577084-12315).
- *Arm® CoreLink™ ADB-400 AMBA® Domain Bridge User Guide* (DUI 0615).

Feedback

Feedback on this product

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

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

Feedback on content

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

- The title *Arm CoreLink MMU-600AE System Memory Management Unit Technical Reference Manual*.
- The number 101412_0100_00_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

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

Chapter 1
Introduction

This chapter provides an overview of the MMU-600AE.

It contains the following sections:

1.1 About the MMU-600AE

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

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

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

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

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

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

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

The MMU-600AE contains the following key components:

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

1.2 Compliance

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

1.2.1 Arm architecture

The MMU-600AE implements parts of the Armv8 *Virtual Memory System Architecture* (VMSA), as the *Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile* defines. The SMMUv3 architecture describes the parts of VMSA that apply to the MMU-600AE.

1.2.2 SMMU architecture

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

Related information

2.4.1 SMMUv3 support on page 2-54

1.2.3 AMBA Distributed Translation Interface protocol

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

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

Related information

2.3.1 DTI overview on page 2-38

1.2.4 AMBA ACE5-Lite and AMBA® AXI5 protocol

The MMU-600AE complies with the AMBA ACE5-Lite protocol.

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

Related information

2.4.2 AMBA support on page 2-57

1.2.5 AMBA APB protocol

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

1.2.6 AMBA Interface Parity

The MMU-600AE complies with the AMBA Interface Parity Specification, as the *AMBA® APB Protocol Specification* defines.

1.3 Features

The MMU-600AE provides the following features:

- Compliance with the SMMUv3.1 architecture:
  - Support for stage 1 translation, stage 2 translation, and stage 1 followed by stage 2 translation
  - Support for Armv8 AArch32 and AArch64 translation table formats
  - Support for 4KB, 16KB, and 64KB granule sizes in AArch64 format
  - Support for *PCI Express* (PCIe) integration, including *Address Translation Services* (ATS) and *Process Address Space IDs* (PASIDs)
  - Support for *Page Request Interface* (PRI), as SMMUv3 defines. PRI is an optional PCIe ATS extension that enables support for unpinned memory in PCIe.
  - Support for ACE5-Lite atomic transactions in the TBU
  - Masters can be stalled while a processor handles translation faults, enabling software support for demand paging
  - Configuration tables in memory can support millions of active translation contexts
  - Queues in memory perform MMU-600AE management. There is no requirement to stall a processor when it accesses the MMU-600AE.
  - Support for *Generic Interrupt Controller* (GIC) integration, with *Message Signaled Interrupts* (MSIs) supported for common interrupt types
  - A *Performance Monitoring Unit* (PMU) in each TBU and TCU that enables MMU-600AE performance to be investigated
  - *Reliability, Serviceability, and Availability* (RAS) features for cache corruption detection and correction
- Support for AMBA interfaces, including:
  - ACE5-Lite TBU transaction interfaces that support cache stash transactions, deallocating transactions, and cache maintenance
  - Option to disable cache maintenance operations on a TBU, a sideband channel protection feature
  - An architected AXI5 extension that communicates per-transaction translation stream information
  - An ACE5-Lite+*Distributed Virtual Memory* (DVM) TCU table walk interface that enables Armv8.2 processors to perform shared TLB invalidate operations without accessing the MMU-600AE directly
  - An ACE5 Low-Power extension that enables the TCU to subscribe to DVM TLB invalidate requests on powerup and powerdown without reprogramming the DTI interconnect
  - AMBA DTI communication between the TCU and TBUs, enabling masters to request translations and implement TBU functionality internally
  - Support for the AMBA *Low-Power Interface* (LPI) Q-Channel so that standard controllers can control power and clock gating
  - AXI5 **WAKEUP** signaling on all interfaces, including DTI and APB interfaces
- Support for flexible integration:
  - You can place a configurable number of TBUs close to the masters being translated
  - Communication between TBU and TCU over AXI4-Stream is supported using the supplied DTI interconnect components, or any other AXI4-Stream interconnect
  - DTI interconnect components support hierarchical topologies and control the tradeoff between the number of wires and the DTI bandwidth
- Support for high-performance translation:
  - Scalable configurable micro TLB and *Main TLB* (MTLB) in the TBU can reduce the number of translation requests to the TCU
  - TBU direct indexing and MTLB partitioning enable the use of MTLB entries to be managed outside the TBU, improving real-time translation performance
  - Optimization enables storage of all architecturally-defined page and block sizes, including contiguous page and block entries, as a single entry in the TBU and TCU TLBs
  - Per-TBU prioritization in the TCU enables high-priority transaction streams to be translated before low-priority streams

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

1.4 Interfaces

Both the TCU and TBU support the following common interfaces:

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

The TCU also supports the following interfaces:

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

The TBU also supports the following interfaces:

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

Related information

2.2 Interfaces on page 2-30

1.5 Configurable options

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

For the TCU, you can configure the following:

- Size of each cache
- Data width of the QTW/DVM interface
- Number of translations that can be performed at the same time
- Number of translation requests that can be accepted from all DTI masters

For the TBU, you can configure the following:

- Write data buffer depth
- Size of each cache
- Number of transactions that can be translated at the same time
- Number of outstanding read and write transactions that the TBM interface supports
- Width of data, ID, user, StreamID, and SubstreamID signals on the TBS and TBM interfaces

Note

Depths are specified as a discrete number of entries.

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

1.6 Product documentation and design flow

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

1.6.1 Documentation

The MMU-600AE 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-600AE. It is required at all stages of the design flow. The choices that are made in the design flow can mean that some behaviors that are described in the TRM are not relevant. If you are programming the MMU-600AE, then contact:

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

Configuration and Integration Manual

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

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

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

1.6.2 Design flow

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

Implementation

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

Integration

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

Programming

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

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

The operation of the final device depends on:

Build configuration

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

- Area
- Maximum frequency
- Features of the resulting macrocell

Configuration inputs

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

Software configuration

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

Related information

1.2 Compliance on page 1-14
1.5 Configurable options on page 1-18

1.7 Product revisions

This section describes the differences in functionality between product revisions:

r0p0 First release.

1.8 Functional Safety (FuSa)

Functionality includes:

1. Description of the *Fault Management Unit* (FMU) PV and APB interface used to program *Safety Mechanisms* (SMs), inject errors, read fault status information, and clear fault status *Error Recovery Interrupt* (ERI) and *Fault Handling Interrupt* (FHI) interrupts.
2. Additional FuSa configurations options.
3. Safety Mechanism descriptions and configuration.
4. FuSa I/Os, reset, and clocking.

Chapter 4 Functional Safety on page 4-98 describes the functionality in detail.

Chapter 2
Functional description

This chapter describes the functionality of the MMU-600AE.

It contains the following sections:

2.1 About the functions

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

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

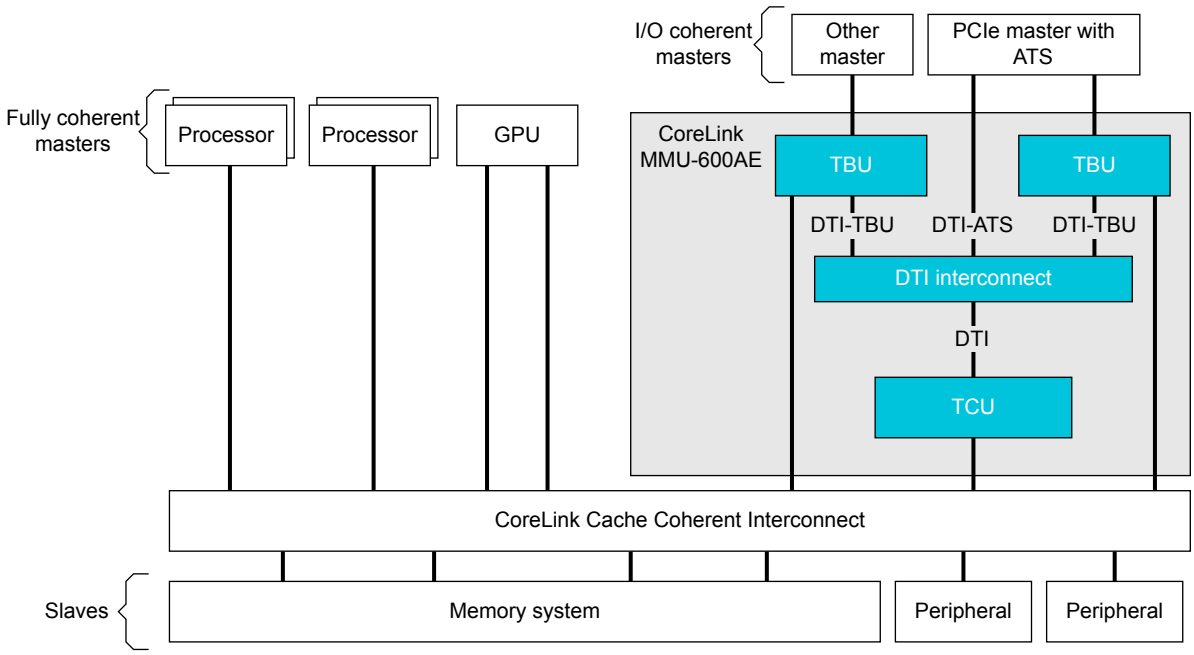

Figure 2-1 Example system with the MMU-600AE

The MMU-600AE contains the following key components:

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

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

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

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

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

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

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

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

A processor controls the TCU by:

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

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

- Translation.
- How software communicates with the TCU.

This section contains the following subsections:

- *2.1.1 Translation Buffer Unit* on page 2-25.
- *2.1.2 Translation Control Unit* on page 2-26.
- *2.1.3 DTI interconnect* on page 2-28.

2.1.1 Translation Buffer Unit

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

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

The following figure shows the TBU.

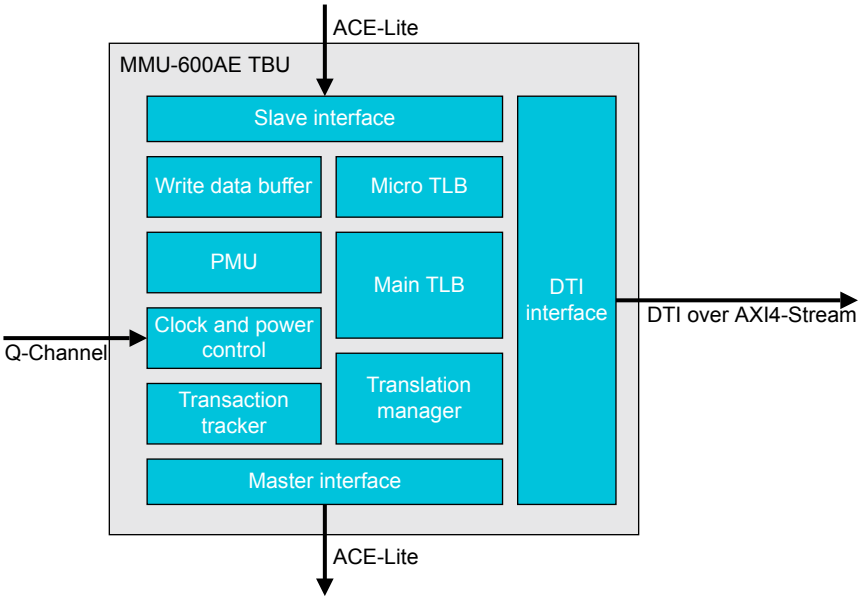

Figure 2-2 MMU-600AE TBU

The TBU consists of:

Master and slave interfaces
These interfaces manage the TBS and TBM interfaces.

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

Main TLB

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

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

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

Translation manager

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

Write data buffer

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

PMU

The PMU counts TBU performance-related events.

Clock and power control

The TBU has its own clock and power control, that the Q-Channel provides.

DTI interface

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

Transaction tracker

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

Related information

2.3.3 TBU direct indexing and MTLB partitioning on page 2-45
3.2 SMMU architectural registers on page 3-64

2.1.2 Translation Control Unit

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

The TCU:

- Manages the memory queues
- Performs translation table walks

- Performs configuration table walks
- Implements backup caching structures
- Implements the SMMU programmers model

The following figure shows the TCU.

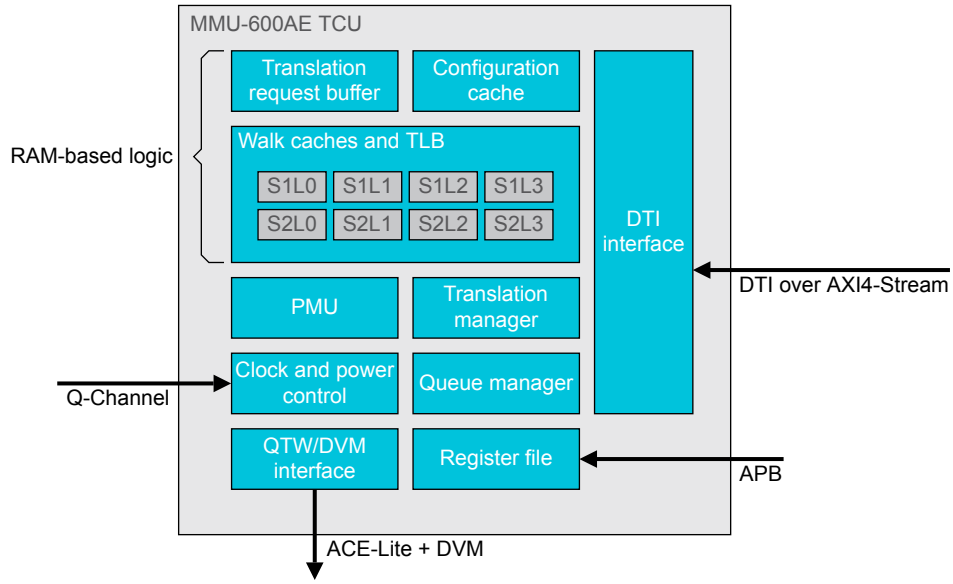

Figure 2-3 MMU-600AE TCU

The TCU consists of:

Walk caches

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

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

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

Configuration cache

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

Note

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

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

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

PMU
The PMU counts TCU performance-related events.

Clock and power control
The TCU has its own clock and power control, that the Q-Channel provides.

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

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

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

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

Related information

2.2 Interfaces on page 2-30
2.3.7 TCU transaction handling on page 2-47
2.3.8 TCU prefetch on page 2-48
3.2 SMMU architectural registers on page 3-64

2.1.3 DTI interconnect

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

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

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

The DTI interconnect includes switch, sizer, and register slice components.

Switch

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

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

Note

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

Sizer

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

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

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

Register slice

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

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

Related information

2.3 Operation on page 2-38

2.2 Interfaces

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

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

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

This section contains the following subsections:

2.2.1 TCU interfaces

The MMU-600AE TCU includes several master and slave interfaces.

The following figure shows the TCU interfaces.

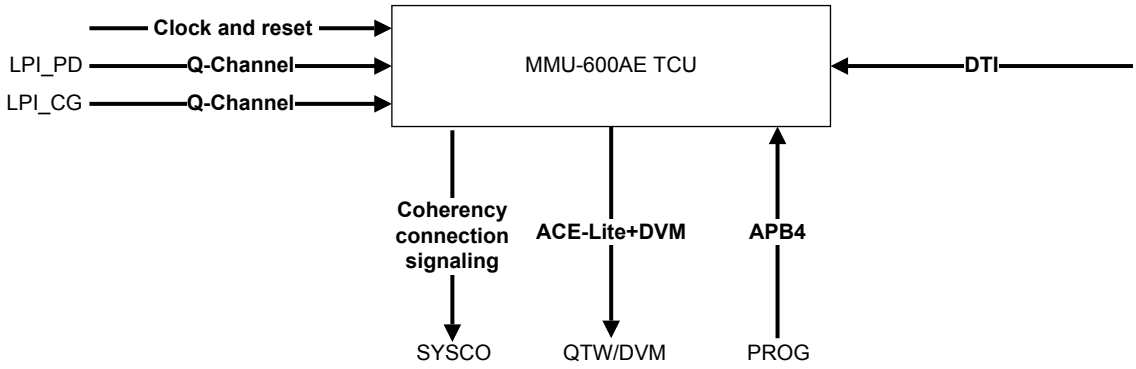

Figure 2-4 TCU interfaces

TCU Queue and Table Walk/Distributed Virtual Memory interface

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

The QTW/DVM interface issues the following transaction types:

- ReadNoSnoop
- WriteNoSnoop
- ReadOnce
- WriteUnique
- DVM Complete

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

Table 2-1 Possible arid_qtw values

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

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

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

Related information

2.3.6 Distributed Virtual Memory (DVM) messages on page 2-46

2.3.9 Error responses on page 2-49

AXI5 support on page 2-59

A.2 TCU QTW/DVM interface signals on page Appx-A-152

TCU PROG interface

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

This interface runs synchronously with the other TCU interfaces.

The applicable address width for this interface is 21 bits, corresponding to `TCUCFG_NUM_TBU` = 14.

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

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

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

Related information

A.3 TCU programming interface signals on page Appx-A-155

TCU LPI_PD interface

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

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

Related information

A.6 TCU LPI_PD interface signals on page Appx-A-158

TCU LPI_CG interface

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

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

Related information

A.7 TCU LPI_CG interface signals on page Appx-A-159

TCU DTI interface

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

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

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

Related information

2.3.1 DTI overview on page 2-38
A.8 TCU DTI interface signals on page Appx-A-160

TCU interrupt interfaces

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

Related information

A.9 TCU interrupt signals on page Appx-A-161

TCU SYSCO interface

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

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

If the **sup_btm** signal is tied LOW, the **syscoreq** signal is always driven LOW and **syscoack** is ignored.

Related information

A.4 TCU SYSCO interface signals on page Appx-A-156

TCU tie-off signals

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

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

Related information

A.11 TCU tie-off signals on page Appx-A-164

2.2.2 TBU interfaces

Each MMU-600AE TBU includes several master and slave interfaces.

The following figure shows the TBU interfaces:

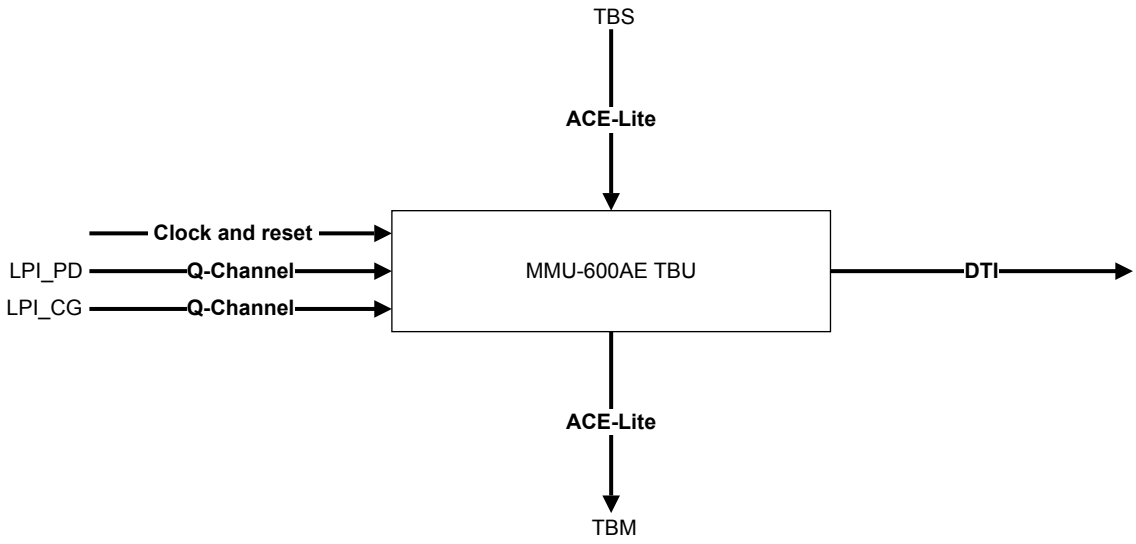

Figure 2-5 TBU interfaces

TBU TBS interface

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

This interface supports a 64-bit address width.

The interface implements optional signals to support the following AXI5 extensions:

- Wakeup_Signals
- Untranslated_Transactions
- Cache_Stash_Transactions
- DeAllocation_Transactions

The TBS interface supports ACE Exclusive accesses.

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

Related information

2.3.9 Error responses on page 2-49
A.13 TBU TBS interface signals on page Appx-A-166

TBU TBM interface

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

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

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

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

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

The interface implements optional signals to support the following AXI5 extensions:

- Wakeup_Signals
- Untranslated_Transactions
- Cache_Stash_Transactions
- DeAllocation_Transactions

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

The TBM interface supports ACE Exclusive accesses.

Related information

2.3.9 Error responses on page 2-49

2.4.2 AMBA support on page 2-57

A.14 TBU TBM interface signals on page Appx-A-169

TBU LPI_PD interface

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

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

Related information

A.16 TBU LPI_PD interface signals on page Appx-A-173

TBU LPI_CG interface

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

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

Related information

A.17 TBU LPI_CG interface signals on page Appx-A-174

TBU DTI interface

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

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

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

Related information

2.3.1 DTI overview on page 2-38

A.18 TBU DTI interface signals on page Appx-A-175

TBU interrupt interfaces

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

Related information

A.19 TBU interrupt signals on page Appx-A-176

TBU tie-off signals

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

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

Related information

A.20 TBU tie-off signals on page Appx-A-177

2.2.3 DTI interconnect interfaces

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

DTI interconnect switch interfaces

The DTI interconnect switch component includes dedicated interfaces.

The following figure shows the DTI interconnect switch interfaces.

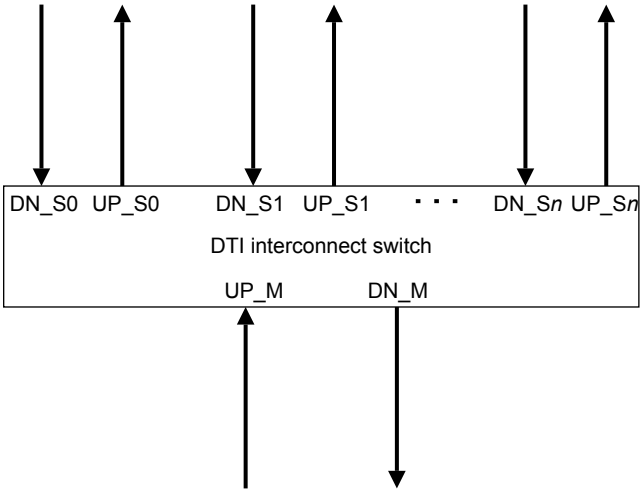

Figure 2-6 DTI interconnect switch interfaces

The following table provides more information about the switch interfaces.

Table 2-2 DTI interconnect switch interfaces

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

Note

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

DTI interconnect sizer interfaces

The DTI interconnect sizer component includes dedicated interfaces.

The following figure shows the DTI interconnect sizer interfaces.

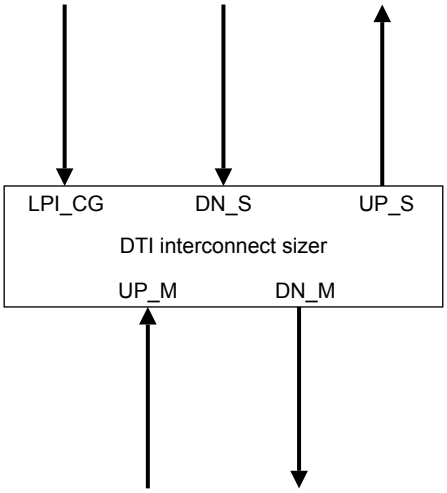

Figure 2-7 DTI interconnect sizer interfaces

The following table provides more information about the sizer interfaces.

Table 2-3 DTI interconnect sizer interfaces

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

DTI interconnect register slice interfaces

The DTI interconnect register slice component includes dedicated interfaces.

The following figure shows the DTI interconnect register slice interfaces.

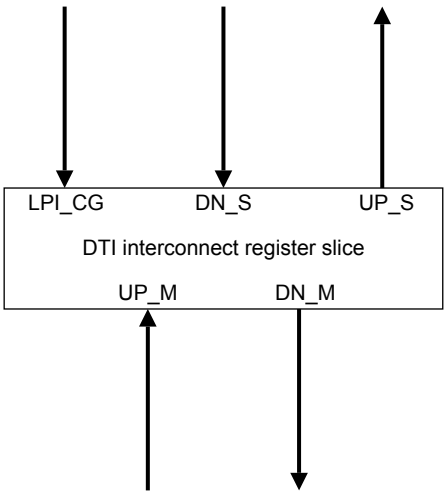

Figure 2-8 DTI interconnect register slice interfaces

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

Table 2-4 DTI interconnect register slice interfaces

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

2.3 Operation

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

This section contains the following subsections:

2.3.1 DTI overview

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

The AMBA DTI protocol includes both:

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

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

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

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

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

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

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

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

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

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

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

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

Note

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

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

2.3.2 Performance Monitoring Unit

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

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

SMMUv3 architectural performance events

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

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

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

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

Table 2-5 SMMUv3 performance events for the TCU

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

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

Table 2-6 SMMUv3 performance events for the TBU

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

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

MMU-600AE TCU events

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

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

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

The following table shows the TCU performance events.

Table 2-7 MMU-600AE TCU performance events

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

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

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

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

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

Note

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

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

MMU-600AE TBU events

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

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

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

The following table shows the TBU performance events.

Table 2-8 MMU-600AE TBU performance events

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

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

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

SMMUv3 PMU register architectural options

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

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

Table 2-9 MMU-600AE SMMU_PMCG_CFGR register architectural options

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

Related information

3.3 MMU-600AE memory map on page 3-69

PMU snapshot interface

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

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

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

Related information

2.3.4 Reliability, Availability, and Serviceability on page 2-46
A.5 TCU PMU snapshot interface signals on page Appx-A-157
A.15 TBU PMU snapshot interface signals on page Appx-A-172

2.3.3 TBU direct indexing and MTLB partitioning

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

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

- It can be guaranteed that different streams do not overwrite prefetched entries
- The MTLB can be partitioned into different sets of entries that different streams use

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

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

When `TBUCFG_DIRECT_IDX` = 1, or when an MTLB is partitioned, the width of the **AxUSER** signals on the TBS interface is extended to convey the indexing information that is required for TBU direct indexing or MTLB partitioning.

Note

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

Table 2-10 Extended aruser_s and awuser_s bits for MTLB partitioning

Field name	Width	Description
mtlbidx	When direct indexing is enabled, the width of this field is $\log^2$(`TBUCFG_MTLB_DEPTH`) - `2`. When direct indexing is not enabled, the width of this field is `0`.	MTLB index
mtlbway	When direct indexing is enabled, the width of this field is `2`. When direct indexing is not enabled, the width of this field is `0`.	MTLB way
mtlbpart	$\log^2$(`TBUCFG_MTLB_PARTS`)	MTLB partition
-	`TBUCFG_AWUSER_WIDTH` for **awuser_s**. `TBUCFG_ARUSER_WIDTH` for **aruser_s**.	Regular **AxUSER** signals

If an MTLB is partitioned:

- The MTLB size is multiplied by `TBUCFG_MTLB_PARTS`
- The mtlbpart field defines the $\log^2$(`TBUCFG_MTLB_PARTS`) most significant index bits

When direct indexing is enabled for a TBU:

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

To maintain system performance, Arm recommends that DVM invalidation is disabled on TBUs on which direct indexing is enabled. Disable DVM invalidation by setting the appropriate TCU_NODE_CTRLn.DIS_DVM bit. See *3.7.6 TCU_NODE_CTRLn* on page 3-82.

2.3.4 Reliability, Availability, and Serviceability

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

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

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

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

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

Related information

3.8.1 TCU_ERRFR on page 3-85
3.8.2 TCU_ERRCTLR on page 3-85
3.8.3 TCU_ERRSTATUS on page 3-86
3.12.1 TBU_ERRFR on page 3-94
3.12.2 TBU_ERRCTLR on page 3-94
3.12.3 TBU_ERRSTATUS on page 3-95

2.3.5 Quality of Service

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

The TCU uses this priority level to:

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

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

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

Related information

3.7.2 TCU_QOS on page 3-78
3.7.6 TCU_NODE_CTRLn on page 3-82

2.3.6 *Distributed Virtual Memory* (DVM) messages

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

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

Tie the **sup_btm** input signal HIGH when the system supports Broadcast TLB Maintenance.

When Broadcast TLB Maintenance is supported, you can use SMMU_CR2 and SMMU_S_CR2 to control how the SMMU handles TLB Invalidate operations as follows:

SMMU_CR2.PTM = 0 Non-secure TLB Invalidate operations are applied to the TLBs.

SMMU_CR2.PTM = 1 Non-secure TLB Invalidate operations have no effect.

SMMU_S_CR2.PTM = 0 Secure TLB Invalidate operations are applied to the TLBs.

SMMU_S_CR2.PTM = 1 Secure TLB Invalidate operations have no effect.

——— **Note** ———

When **sup_btm** is tied HIGH, the reset value of SMMU_CR2.PTM and SMMU_S_CR2.PTM is 1.

———————————

——— **Note** ———

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

———————————

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

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

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

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

2.3.7 TCU transaction handling

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

The following table shows the TCU support for read transactions.

Table 2-11 TCU support for read transactions

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

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

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

The following table shows the TCU support for write transactions.

Table 2-12 TCU support for write transactions

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

Only one write transaction can be outstanding at a time.

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

2.3.8 TCU prefetch

The TCU can prefetch translations on a per-context basis to improve translation performance for real-time masters that access memory linearly. If TCU prefetch is enabled, a second translation request occurs after the original request. This second translation request is regarded as the *prefetch* because it is an advance request of the next translation that is expected to be accessed. This second request is Speculative and is used to allocate into the caches of the TCU.

Software can request a TCU prefetch for a particular translation context by programming the STE.PF field using bits [121:120].

PF, bits [121:120]
This field determines whether prefetch is enabled or disabled for the translation context that this *Stream Table Entry* (STE) defines and has the following settings:

`0b00` Prefetching disabled.

`0b01` Reserved.

`0b10` Forward prefetching. The address to be prefetched is the first address following the end of the translation range, as DTI_TBU_TRANS_RESP.TRANS_RNG/ DTI_ATS_TRANS_RESP.TRANS_RNG indicates.

`0b11` Backward prefetching. The address to be prefetched is the last address before the beginning of the translation range, as DTI_TBU_TRANS_RESP.TRANS_RNG/ DTI_ATS_TRANS_RESP.TRANS_RNG indicates.

Whenever a miss occurs in the micro TLB and Main TLB of the TBU, the TBU sends a translation request to the TCU. If the STE that performs the translation is programmed to enable prefetch, each translation request to the TCU can also potentially result in a prefetch that occurs after the original request is complete. When each incoming translation request completes its translation in the TCU, the STE.PF indicates whether TCU prefetch is enabled. If TCU prefetch is enabled, a second translation request, the prefetch request, is then issued into the same TCU translation slot. This prefetch request is Speculative, and only allocates into the TCU walk caches. A translation response for the prefetch is not returned to the TBU.

When the TCU handles each incoming translation request from the TBU, translation table walks might or might not occur depending on whether there is a hit in each level of walk cache that is looked up. Translation table walks also might or might not occur for the subsequent prefetch request. The number of memory accesses that are performed for this prefetch are unrelated to the number of memory accesses of the original translation request.

Consider the following examples:

1. An incoming translation request might hit in the lowest level of walk cache, but the subsequent prefetch request might still require at least one translation table walk to memory.

2. The original translation request might require multiple translation table walks, but the subsequent prefetch request might hit in the lowest level of walk cache and not require any memory accesses. If the prefetch request hits in the lowest level of walk cache, then the walk caches are not updated and no memory accesses are performed.

   Note

   The walk caches use a random replacement policy, so lookups are not recorded and do not affect the order in which the entries are evicted.

The prefetch can only occur when the original request is complete irrespective of whether translation table walks were required because the prefetch uses the same translation slot as the original request. Waiting for completion of the original request means that by the time it becomes possible for the prefetch to be initiated, the TCU might have already received a non-speculative request for the next translation and begun to handle this request using a separate translation slot. Therefore, TCU prefetch only results in a performance advantage if the number of cycles between each sequential translation request from the TBU is greater than the number of cycles that is taken for the TCU to complete the original translation request and to start the subsequent prefetch.

Even if TCU prefetch is enabled, a second translation request, or prefetch, does not occur if one of the following caused the original request:

- A Speculative translation request, that is, **DTI_TBU_TRANS_REQ.PERM[1:0]** = `2'b11`, if a TBU receives a StashOnceShared, StashOnceUnique, or StashTranslation transaction
- A translation request for an atomic transaction that provides a data response, that is, **DTI_TBU_TRANS_REQ.PERM[1:0]** = `2'b10`, if a TBU receives an AtomicLoad, AtomicSwap, or AtomicCompare transaction

If the original translation request returns one the following, prefetch also does not occur:

- Fault response
- Global bypass response
- Stream bypass response

Note

Prefetch applies to both ATS and non-ATS translation requests.

2.3.9 Error responses

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

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

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

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

2.3.10 Conversion between ACE-Lite and Arm®v8 attributes

The SMMUv3 architecture defines attributes in terms of the Armv8 architecture. The MMU-600AE components are therefore required to perform conversion between ACE-Lite and Armv8 attributes.

The TBU must convert:

- ACE-Lite attributes to Armv8 attributes when it receives transactions on the *Transaction Slave* (TBS) interface
- Armv8 attributes to ACE-Lite attributes when it outputs transactions on the *Transaction Master* (TBM) interface

The TCU must convert Armv8 attributes to ACE-Lite attributes when it outputs transactions on the QTW/DVM interface.

Attribute handling

This section describes attribute handling in the MMU-600AE.

When translation is enabled and a PCIe Root Complex issues transactions to a TBU, the following apply, depending on the type of transaction:

Untranslated (non-ATS) transaction

The SMMU applies attributes that a combination of the input attributes, STE overrides, and translation table descriptors determine.

Fully-translated (full ATS) transaction

The MMU-600AE generates fixed attributes.

See the *Arm® System Memory Management Unit Architecture Specification, SMMU architecture versions 3.0 and version 3.2* for information about the preceding transactions and their attributes.

——— **Note** ———

TBUs that are connected to a PCIe Root Complex must have the **pcie_mode** input signal tied HIGH, as the table in *A.20 TBU tie-off signals* on page Appx-A-177 describes.

———————————

Slave interface memory type attribute handling

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

The following table shows the ACE-Lite to Armv8 attribute conversions that the TBU TBS interface performs.

Table 2-13 MMU-600AE ACE-Lite to Armv8 memory attribute conversions

AxCACHE attribute	AxDOMAIN attribute	Armv8 memory attribute	Armv8 Shareability
Device Non-bufferable	System	Device-nGnRnE	Outer Shareable
Device Bufferable	System	Device-nGnRE	Outer Shareable
Normal Non-cacheable Bufferable Normal Non-cacheable Non-bufferable Write-Through No Allocate Write-Through Read-Allocate Write-Through Write-Allocate Write-Through Read and Write-Allocate	Any	Normal Inner Non-cacheable Outer Non-cacheable	Outer Shareable
Write-Back No Allocate Write-Back Read-Allocate Write-Back Write-Allocate Write-Back Read-Allocate Write-Allocate	Non-shareable Inner Shareable Outer Shareable	Normal Inner Write-Back Outer Write-Back	Non-shareable Non-shareable Outer Shareable

Note

- Write-Back transactions are always treated as non-transient.
- The Armv8-A Read-Allocate and Write-Allocate hints are the same as the hints that the **AxCACHE** Write-Back type provides.
- The TBU TBS interface converts instruction writes into data writes. That is, it treats **awprot_s[2]** as 0.

Master interface memory type attribute handling

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

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

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

The following table shows the Armv8 to ACE-Lite attribute conversions that the master interfaces perform.

Table 2-14 MMU-600AE Armv8 to ACE-Lite memory attribute conversions

Armv8 memory attribute	AxCACHE attribute	AxDOMAIN attribute	AxLOCK attribute	AxUSER Outer Cacheable
Device-nGnRnE	Device Non-bufferable	System	As *Transaction Slave* (TBS) **AxLOCK** value	0
Device-GRE Device-nGRE Device-nGRE	Device Bufferable	System	As TBS **AxLOCK** value	0
Normal Inner Non-cacheable Outer Non-cacheable Normal Inner Write-Through Outer Non-cacheable Normal Inner Write-Back Outer Non-cacheable	Normal Non-cacheable Bufferable	System	As TBS **AxLOCK** value	0

Table 2-14 MMU-600AE Armv8 to ACE-Lite memory attribute conversions (continued)

Armv8 memory attribute	AxCACHE attribute	AxDOMAIN attribute	AxLOCK attribute	AxUSER Outer Cacheable
Normal Inner Non-cacheable Outer Write-Through Normal Inner Write-Through Outer Write-Through Normal Inner Write-Back Outer Write-Through Normal Inner Non-cacheable Outer Write-Back Normal Inner Write-Through Outer Write-Back	Normal Non-cacheable Bufferable	System	As TBS **AxLOCK** value	1
Normal Inner Write-Back Outer Write-Back	Write-Back No Allocate Write-Back Read-Allocate Write-Back Write-Allocate Write-Back Read and Write-Allocate	If **AxBURST** == FIXED, Non-shareable. If **AxBURST** != FIXED, the attribute reflects the Armv8 Shareability: • Non-shareable • Inner Shareable • Outer Shareable	0	1

2.3.11 AXI USER bits defined by the MMU-600AE TBU

The TBU TBM interface **AxUSER** signals, **aruser_m** and **awuser_m**, have 13 bits more than `TBUCFG_AxUSER_WIDTH` defines. These extra bits are output in higher-order bits of **aruser_m** and **awuser_m**.

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

Table 2-15 MMU-600AE defined aruser_m and awuser_m bits

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

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

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

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

The S1HWATTR and S2HWATTR fields are calculated as follows:

S1HWATTR

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

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

Otherwise, S1HWATTR[*n*] = 0.

S2HWATTR

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

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

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

Related information

Master interface memory type attribute handling on page 2-51

2.4 Constraints and limitations of use

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

Unless otherwise specified:

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

This section contains the following subsections:

2.4.1 SMMUv3 support

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

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

——— **Note** ———

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

———————————

Table 2-16 MMU-600AE SMMUv3 ID register architectural options

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

Table 2-16 MMU-600AE SMMUv3 ID register architectural options (continued)

Register	Field	Value	Description for value
SMMU_IDR2	BA_VATOS	0	No VATOS page is present.
SMMU_IDR3	HAD	1	Hierarchical Attribute Disable is supported.
	PBHA	1	Page-based hardware attributes are supported.
	XNX	1	EL0/EL1 execute control distinction at stage 2 is supported for both AArch64 and AArch32 stage 2 translation tables.
	PPS	1	If the request has a *Process Address Space ID* (PASID), the PASID is included in PRI queue overflow auto-generated responses. The STE.PPAR field is not checked and is treated as 1.
SMMU_IDR4	IMPDEF	0	No IMPLEMENTATION DEFINED features apply.
SMMU_IDR5	OAS	**sup_oas**	The size of the physical address that is output from the SMMU.
	GRAN4K	1	4KB translation granule is supported.
	GRAN16K	1	16KB translation granule is supported.
	GRAN64K	1	64KB translation granule is supported.
	VAX	0b00	Virtual addresses of 48 bits per CD.TTBx are supported.
	STALL_MAX	TCUCFG_XLATE_SLOTS	Maximum number of outstanding stalled transactions that the SMMU supports.
SMMU_IIDR	Implementer	0x43B	Arm implementation.
	Revision	MAX[0x0, **ecorevnum**]	Minor revision is p0. **Note** **ecorevnum** is not configurable.
	Variant	1	Product variant, or major revision is r1.
	ProductID	0x483	Arm ID.
SMMU_AIDR	ArchMinorRev	0b0001	Architecture minor revision is SMMUv3.1.
	ArchMajorRev	0b0000	Architecture major revision is SMMUv3.
SMMU_S_IDR0	MSI	1	Secure MSIs are supported.
	STALL_MODEL	0b00	Stall model and Terminate model are both supported.
SMMU_S_IDR1	S_SIDSIZE	0b11000	24-bit Secure stream IDs are supported.
	SECURE_IMPL	1	Security implemented.
SMMU_S_IDR3	SAMS	1	Secure *Address Translation Services* (ATS) maintenance is not implemented.

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

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

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

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

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

F_UUT
Unsupported Upstream Transaction.

F_TLB_CONFLICT
TLB conflict.

F_CFG_CONFLICT
Configuration cache conflict.

E_PAGE_REQUEST
Speculative page request hint.

IMPDEF_EVENTn
IMPLEMENTATION DEFINED event allocation.

Note

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

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

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

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

Related information

3.2 SMMU architectural registers on page 3-64

2.4.2 AMBA support

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

TBU support for ACE-Lite transactions

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

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

Transactions that can result in a translation fault

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

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

- CleanShared.
- CleanInvalid.
- MakeInvalid.

- CleanSharedPersist.
- ReadOnceMakeInvalid.
- ReadOnceCleanInvalid.

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

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

- WriteUniquePtlStash.
- WriteUniqueFullStash.

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

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

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

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

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

WriteUniquePtlStash and WriteUniqueFullStash transactions
If they pass the translation fault check, the MMU-600AE converts WriteUniquePtlStash and WriteUniqueFullStash transactions to ordinary write transactions if either:

- The translation did not grant *Directed Cache Prefetch* (DCP) permission.
- The final transaction attributes on the TBU TBM interface are not Inner Shareable or Outer Shareable Write-Back.

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

Transactions that cannot result in a translation fault

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

The following transactions never result in a translation fault:

- StashOnceShared
- StashOnceUnique
- StashTranslation

If any of these transactions require a translation request to the TCU, the MMU-600AE issues a Speculative translation request on the DTI interconnect. StashOnceShared and StashOnceUnique transactions are terminated in the TBU, with a **BRESP** value of OKAY, when any of the following cases apply:

- The translation did not grant *Directed Cache Prefetch* (DCP) permission
- The final transaction attributes on the TBM interface are not Inner Shareable or Outer Shareable Write-Back
- The translation did not grant any of read, write, or execute permission at the appropriate privilege level

Note

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

Note

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

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

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

Note

A StashTranslation transaction can be used to prefetch translations into the Main TLB of the MMU-600AE. However, for this prefetching to be useful, any subsequent transactions that intend to take advantage of the translations that have been prefetched into the Main TLB must use the same StreamID as the original prefetch. The StreamID identifies a translation context. Using a different StreamID for a subsequent transaction means that this subsequent transaction uses a different translation context to the translation that has been prefetched into the Main TLB and might lead to a TLB miss.

AXI5 support

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

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

Table 2-17 TCU and TBU interface support for AXI5 extensions

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

Table 2-17 TCU and TBU interface support for AXI5 extensions (continued)

AXI5 extension	QTW/DVM	TBU TBS	TBU TBM
Trace_Signals	-	-	-
Loopback_Signals	-	-	-
NSAccess_Identifiers	-	-	-
Persist_CMO	-	Yes	Yes

Chapter 3
Programmer's model

This chapter describes the MMU-600AE programmer's model.

It contains the following sections:

3.1 About the programmer's model

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

The following information applies to the MMU-600AE registers:

- The base address is not fixed, and can be different for any particular system implementation. The offset of each register from the base address is fixed.
- Access type is described as follows:

  RW Read and write.

  RO Read-only.

  WO Write-only.

  RAZ Read-As-Zero.

  WI Writes ignored.
- Do not attempt to access reserved or unused address locations. Reading these locations results in RAZ and writing to these locations results in WI.
- Unless otherwise stated in the accompanying text:
  - Do not modify UNDEFINED register bits.
  - Ignore UNDEFINED register bits on reads.
  - All register bits are reset to 0 by a system or Cold reset.
- Bit positions that are described as reserved are:
  - In an RW register, RAZ/WI.
  - In an RO register, RAZ.
  - In a WO register, WI.

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

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

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

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

- SMMU_PMCG_IRQ_CFG0
- SMMU_PMCG_IRQ_CFG1
- SMMU_PMCG_IRQ_CFG2
- SMMU_PMCG_IRQ_STATUS

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

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

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

3.2 SMMU architectural registers

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

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

Note

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

Table 3-1 SMMUv3 architectural registers

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

Table 3-1 SMMUv3 architectural registers (continued)

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

Table 3-1 SMMUv3 architectural registers (continued)

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

Table 3-1 SMMUv3 architectural registers (continued)

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

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

Table 3-2 SMMUv3 PMCG registers

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

Table 3-2 SMMUv3 PMCG registers (continued)

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

3.3 MMU-600AE memory map

The MMU-600AE memory map contains all registers.

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

Table 3-3 MMU-600AE memory map

Address range	Description
0x000000 - 0x03FFFC	TCU registers.
0x040000 - 0x05FFFC	TBU0 registers.
0x060000 - 0x07FFFC	TBU1 registers
0x080000 - 0x09FFFC	TBU2 registers.
.	.
.	.
.	.
0x1C00000 - 0x1DFFFC	TBU12 registers.
0x1E00000 - 0x1FFFFC	TBU13 registers.

Note

All TBU and TCU register addresses in this manual are described relative to the base address for that component.

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

Table 3-4 MMU-600AE TCU memory map

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

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

Table 3-5 MMU-600AE TBU memory map

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

3.4 Register summary

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

TBU identification register summary

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

Table 3-6 TBU identification register summary

Offset	Name	Type	Description
0x00FD0	SMMU_PIDR4	RO	*3.9 TBU component and peripheral ID registers* on page 3-90
0x00FD4	SMMU_PIDR5	RO	
0x00FD8	SMMU_PIDR6	RO	
0x00FDC	SMMU_PIDR7	RO	
0x00FE0	SMMU_PIDR0	RO	
0x00FE4	SMMU_PIDR1	RO	
0x00FE8	SMMU_PIDR2	RO	
0x00FEC	SMMU_PIDR3	RO	
0x00FF0	SMMU_CIDR0	RO	
0x00FF4	SMMU_CIDR1	RO	
0x00FF8	SMMU_CIDR2	RO	
0x00FFC	SMMU_CIDR3	RO	

TBU RAS register summary

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

Table 3-7 TBU RAS register summary

Offset	Name	Type	Description
0x08E80	TBU_ERRFR	RO	*3.12.1 TBU_ERRFR* on page 3-94
0x08E88	TBU_ERRCTLR	RW	*3.12.2 TBU_ERRCTLR* on page 3-94
0x08E90	TBU_ERRSTATUS	RW	*3.12.3 TBU_ERRSTATUS* on page 3-95
0x08EC0	TBU_ERRGEN	RW	*3.12.4 TBU_ERRGEN* on page 3-96

TBU microarchitectural register summary

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

Table 3-8 TBU microarchitectural register summary

Offset	Name	Type	Description
0x08E00	TBU_CTRL	RW	*3.11.1 TBU_CTRL* on page 3-92
0x08E18	TBU_SCR	RW	*3.11.2 TBU_SCR* on page 3-92

TCU identification register summary

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

Table 3-9 TCU identification register summary

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

TCU and TBU PMU identification register summary

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

Table 3-10 TCU and TBU PMU identification register summary

Offset	Name	Type	Description
0x02FB8	SMMU_PMCG_PMAUTHSTATUS	RO	*3.6 TCU PMU component and peripheral ID registers* on page 3-75 *3.10 TBU PMU component and peripheral ID registers* on page 3-91
0x02FD0	SMMU_PMCG_PIDR4	RO	
0x02FD4	SMMU_PMCG_PIDR5	RO	
0x02FD8	SMMU_PMCG_PIDR6	RO	
0x02FDC	SMMU_PMCG_PIDR7	RO	
0x02FE0	SMMU_PMCG_PIDR0	RO	
0x02FE4	SMMU_PMCG_PIDR1	RO	
0x02FE8	SMMU_PMCG_PIDR2	RO	
0x02FEC	SMMU_PMCG_PIDR3	RO	
0x02FF0	SMMU_PMCG_CIDR0	RO	
0x02FF4	SMMU_PMCG_CIDR1	RO	
0x02FF8	SMMU_PMCG_CIDR2	RO	
0x02FFC	SMMU_PMCG_CIDR3	RO	

TCU RAS register summary

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

Table 3-11 TCU RAS register summary

Offset	Name	Type	Description
0x08E80	TCU_ERRFR	RO	*3.8.1 TCU_ERRFR* on page 3-85
0x08E88	TCU_ERRCTLR	RW	*3.8.2 TCU_ERRCTLR* on page 3-85
0x08E90	TCU_ERRSTATUS	RW	*3.8.3 TCU_ERRSTATUS* on page 3-86
0x08EC0	TCU_ERRGEN	RW	*3.8.4 TCU_ERRGEN* on page 3-88

TCU microarchitectural register summary

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

Table 3-12 TCU microarchitectural register summary

Offset	Name	Type	Description
0x08E00	TCU_CTRL	RW	*3.7.1 TCU_CTRL* on page 3-76
0x08E04	TCU_QOS	RW	*3.7.2 TCU_QOS* on page 3-78
0x08E08	TCU_CFG	RO	*3.7.3 TCU_CFG* on page 3-79
0x08E10	TCU_STATUS	RO	*3.7.4 TCU_STATUS* on page 3-80
0x08E18	TCU_SCR	RW	*3.7.5 TCU_SCR* on page 3-81
0x09000 - 0x093FC	TCU_NODE_CTRL*n*	RW	*3.7.6 TCU_NODE_CTRLn* on page 3-82
0x09400 - 0x097FC	TCU_NODE_STATUS*n*	RO	*3.7.7 TCU_NODE_STATUSn* on page 3-83

3.5 TCU component and peripheral ID registers

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

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

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

Register	Offset	Bits	Value	Description
SMMU_PIDR4	0x00FD0	[7:4]	0x0	4KB region count.
		[3:0]	0x4	JEP106 continuation code for Arm.
SMMU_PIDR5	0x00FD4	[7:0]	0x00	Reserved.
SMMU_PIDR6	0x00FD8	[7:0]	0x00	Reserved.
SMMU_PIDR7	0x00FDC	[7:0]	0x00	Reserved.
SMMU_PIDR0	0x00FE0	[7:0]	0x87	Part number[7:0].
SMMU_PIDR1	0x00FE4	[7:4]	0xB	JEP106 ID code[3:0] for Arm.
		[3:0]	0x4	Part number[11:8].
SMMU_PIDR2	0x00FE8	[7:4]	0x1	MMU-600AE major revision. The value 0x1 indicates major product revision r1.
		[3]	0b1	The component uses a manufacturer identity code that JEDEC allocates, according to the JEP106 specification.
		[2:0]	0b011	JEP106 ID code[6:4] for Arm.
SMMU_PIDR3	0x00FEC	[7:4]	MAX[0x0,ecorevnum]	MMU-600AE minor revision. The value 0x0 indicates minor product revision p0.
		[3:0]	0x0	CMOD. This field is not used.
SMMU_CIDR0	0x00FF0	[7:0]	0x0D	Preamble.
SMMU_CIDR1	0x00FF4	[7:0]	0xF0	
SMMU_CIDR2	0x00FF8	[7:0]	0x05	
SMMU_CIDR3	0x00FFC	[7:0]	0xB1	

3.6 TCU PMU component and peripheral ID registers

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

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

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

Register	Offset	Bits	Value	Description
SMMU_PMCG_PMAUTHSTATUS	0x02FB8	[7:0]	0x00	No authentication interface is implemented.
SMMU_PMCG_PIDR4	0x02FD0	[7:4]	0x0	4KB region count.
		[3:0]	0x4	JEP106 continuation code for Arm.
SMMU_PMCG_PIDR5	0x02FD4	[7:0]	0x00	Reserved.
SMMU_PMCG_PIDR6	0x02FD8	[7:0]	0x00	Reserved.
SMMU_PMCG_PIDR7	0x02FDC	[7:0]	0x00	Reserved.
SMMU_PMCG_PIDR0	0x02FE0	[7:0]	0x87	Part number[7:0].
SMMU_PMCG_PIDR1	0x02FE4	[7:4]	0xB	JEP106 ID code[3:0] for Arm.
		[3:0]	0x4	Part number[11:8].
SMMU_PMCG_PIDR2	0x02FE8	[7:4]	0x1	MMU-600AE revision. The value 0x1 indicates major product revision r1.
		[3]	0b1	The component uses a manufacturer identity code that JEDEC allocates, according to the JEP106 specification.
		[2:0]	0b011	JEP106 ID code[6:4] for Arm.
SMMU_PMCG_PIDR3	0x02FEC	[7:4]	MAX[0x0,ecorevnum]	MMU-600AE minor revision. The value 0x0 indicates minor product revision p0.
		[3:0]	0x0	CMOD. This field is not used.
SMMU_PMCG_CIDR0	0x02FF0	[7:0]	0x0D	Preamble.
SMMU_PMCG_CIDR1	0x02FF4	[7:0]	0x90	
SMMU_PMCG_CIDR2	0x02FF8	[7:0]	0x05	
SMMU_PMCG_CIDR3	0x02FFC	[7:0]	0xB1	

3.7 TCU microarchitectural registers

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

This section contains the following subsections:

3.7.1 TCU_CTRL

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

The TCU_CTRL characteristics are:

Usage constraints

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

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

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

Configurations

This register exists in all TCU configurations.

Attributes

Offset `0x08E00`

Type RW

Reset `0x00000000`

Width 32

The following figure shows the bit assignments.

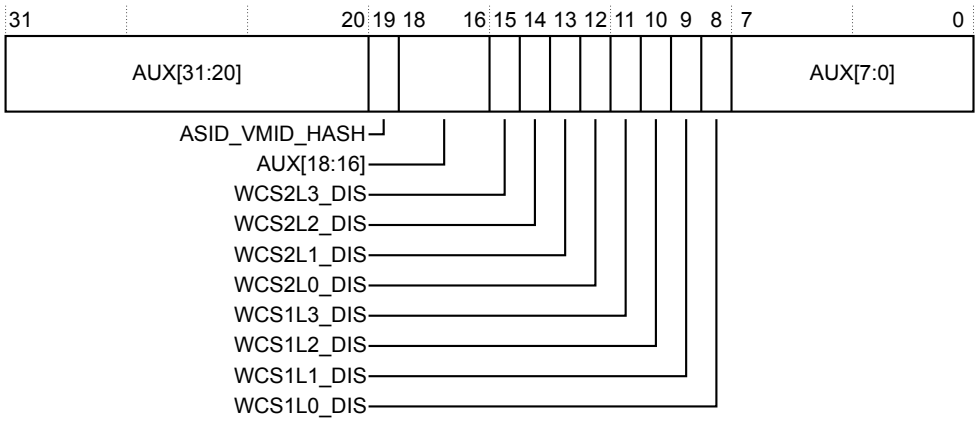

Figure 3-1 TCU_CTRL register bit assignments

The following table shows the bit descriptions.

Table 3-15 TCU_CTRL register bit descriptions

Bits	Name	Description
[31:20]	AUX[31:20]	Leave each of these bits as 0.
[19]	ASID_VMID_HASH	0 VMID and input address are used for walk cache indexing. Transactions with the same VMID and input address, but different ASID, use the same walk cache index. Walk cache utilization is poor if such transactions are common. 1 ASID, VMID, and input address are used for walk cache indexing. Transactions with the same VMID and input address, but different ASID, use different walk cache indexes. This improves walk cache utilization if different ASIDs are used for the same input address and VMID, but invalidation performance is worse for invalidations that do not provide an ASID because the whole cache must be walked instead of invalidating based on a specific index.
[18:16]	AUX[18:16]	Leave each of these bits as 0.
[15]	WCS2L3_DIS	Walk cache disable: 0 Stage 2 level 3 walk cache is enabled. 1 Stage 2 level 3 walk cache is disabled.
[14]	WCS2L2_DIS	Walk cache disable: 0 Stage 2 level 2 walk cache is enabled. 1 Stage 2 level 2 walk cache is disabled.
[13]	WCS2L1_DIS	Walk cache disable: 0 Stage 2 level 1 walk cache is enabled. 1 Stage 2 level 1 walk cache is disabled.
[12]	WCS2L0_DIS	Walk cache disable: 0 Stage 2 level 0 walk cache is enabled. 1 Stage 2 level 0 walk cache is disabled.

Table 3-15 TCU_CTRL register bit descriptions (continued)

Bits	Name	Description
[11]	WCS1L3_DIS	Walk cache disable: `0` Stage 1 level 3 walk cache is enabled. `1` Stage 1 level 3 walk cache is disabled.
[10]	WCS1L2_DIS	Walk cache disable: `0` Stage 1 level 2 walk cache is enabled. `1` Stage 1 level 2 walk cache is disabled.
[9]	WCS1L1_DIS	Walk cache disable: `0` Stage 1 level 1 walk cache is enabled. `1` Stage 1 level 1 walk cache is disabled.
[8]	WCS1L0_DIS	Walk cache disable: `0` Stage 1 level 0 walk cache is enabled. `1` Stage 1 level 0 walk cache is disabled.
[7:0]	AUX[7:0]	Leave each of these bits as `0`.

3.7.2 TCU_QOS

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

The **AxQOS** value that is associated with each transaction does not take account of other transactions that are blocked behind it. For example, although higher priority translations are normally progressed before lower priority translations, a low-priority translation table walk might prevent the TCU from issuing a translation table walk with a higher priority.

The TCU_QOS characteristics are:

Usage constraints

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

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

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

Configurations

This register exists in all TCU configurations.

Attributes

Offset `0x08E04`

Type RW

Reset `0x00000000`

Width 32

The following figure shows the bit assignments.

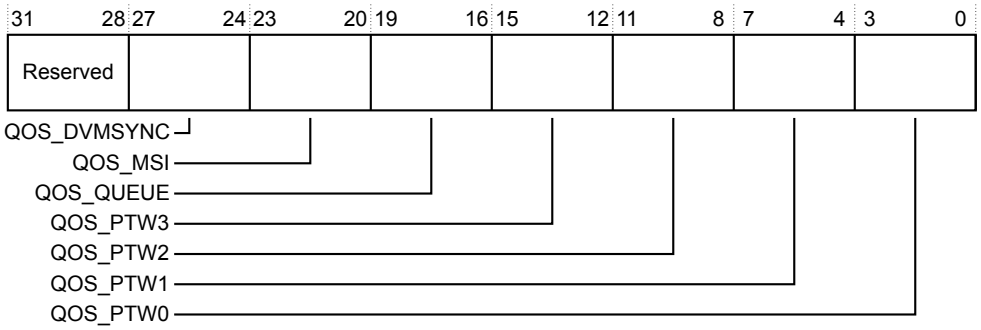

Figure 3-2 TCU_QOS register bit assignments

The following table shows the bit descriptions.

Table 3-16 TCU_QOS register bit descriptions

Bits	Name	Description
[31:28]	-	Reserved.
[27:24]	QOS_DVMSYNC	The **AxQOS** value that is used for DVM Sync Completion messages.
[23:20]	QOS_MSI	The **AxQOS** value that is used for MSIs.
[19:16]	QOS_QUEUE	The **AxQOS** value that is used for queue accesses.
[15:12]	QOS_PTW3	The **AxQOS** value that is used for translation table walks for translations where TCU_NODE_CTRLn.PRIORITY = 3 for the requesting node.
[11:8]	QOS_PTW2	The **AxQOS** value that is used for translation table walks for translations where TCU_NODE_CTRLn.PRIORITY = 2 for the requesting node.
[7:4]	QOS_PTW1	The **AxQOS** value that is used for translation table walks for translations where TCU_NODE_CTRLn.PRIORITY = 1 for the requesting node.
[3:0]	QOS_PTW0	The **AxQOS** value that is used for translation table walks for translations where TCU_NODE_CTRLn.PRIORITY = 0 for the requesting node.

3.7.3 TCU_CFG

This is the TCU Configuration Information register.

Its characteristics are:

Usage constraints

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

Configurations

This register exists in all TCU configurations.

Attributes

Offset `0x08E08`

Type RO

Reset See register bit assignments.

Width 32

The following figure shows the bit assignments.

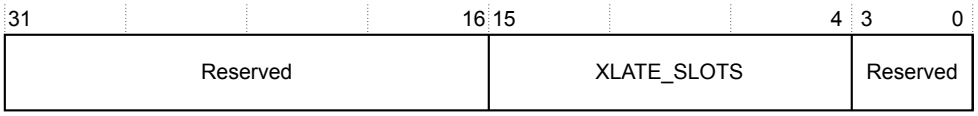

Figure 3-3 TCU_CFG register bit assignments

The following table shows the bit descriptions.

Table 3-17 TCU_CFG register bit descriptions

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

3.7.4 TCU_STATUS

TCU_STATUS is the TCU Status Information register.

Its characteristics are:

Usage constraints

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

Configurations

This register exists in all TCU configurations.

Attributes

Offset `0x08E10`

Type RO

Reset `0x00000000`

Width 32

The following figure shows the bit assignments.

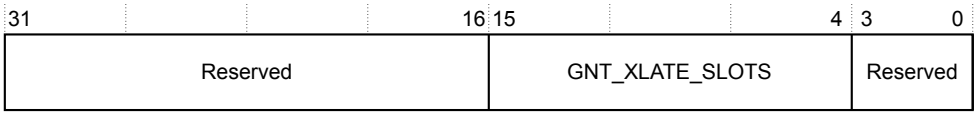

Figure 3-4 TCU_STATUS register bit assignments

The following table shows the bit descriptions.

Table 3-18 TCU_STATUS register bit descriptions

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

3.7.5 TCU_SCR

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

The TCU_SCR characteristics are:

Usage constraints

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

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

Configurations

This register exists in all TCU configurations.

Attributes

Offset `0x08E18`

Type RW

Reset See register bit assignments.

Width 32

The following figure shows the bit assignments.

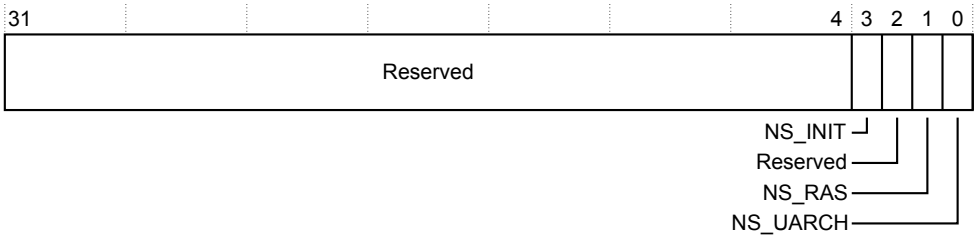

Figure 3-5 TCU_SCR register bit assignments

The following table shows the bit descriptions.

Table 3-19 TCU_SCR register bit descriptions

Bits	Name	Description
[31:4]	-	Reserved.
[3]	NS_INIT	Non-secure register access to SMMU_S_INIT. When this bit is set to 0, Non-secure accesses to the SMMU_S_INIT register are RAZ/WI. The **sec_override** input signal defines the reset value of this bit.
[2]	-	Reserved.

Table 3-19 TCU_SCR register bit descriptions (continued)

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

3.7.6 TCU_NODE_CTRL*n*

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

The TCU_NODE_CTRL*n* characteristics are:

Usage constraints

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

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

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

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

Configurations

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

`0x09000` + (4 × *n*)

Attributes

Offset `0x09000-0x093FC`

Type RW

Reset `0x00000000`

Width 32

The following figure shows the bit assignments.

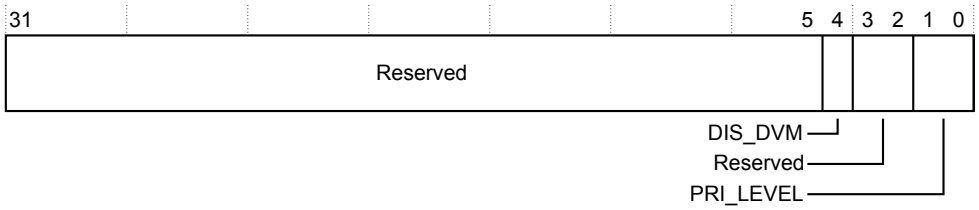

Figure 3-6 TCU_NODE_CTRL register bit assignments

The following table shows the bit descriptions.

Table 3-20 TCU_NODE_CTRL register bit descriptions

Bits	Name	Description
[31:5]	-	Reserved.
[4]	DIS_DVM	Disable DVM. When this bit is set to 1, the corresponding node does not participate in DVM invalidation. Software should set this bit to 1 if all the following are true: • The node is slow to respond to invalidations that are issued over DTI • Software has knowledge that the node does not require to be part of the DVM domain • Software has knowledge that invalidations for the node can be issued using the Command queue **Note** This bit is ignored for connected DTI-ATS masters, because they never participate in DVM invalidation.
[3:2]	-	Reserved.
[1:0]	PRI_LEVEL	Priority level. This field indicates the priority level of the corresponding node. Translation requests from a node with a higher priority level are normally progressed before requests from a node with a lower priority level.

3.7.7 TCU_NODE_STATUS*n*

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

The TCU_NODE_STATUS*n* characteristics are:

Usage constraints

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

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

Configurations

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

`0x09400` + (4 × *n*)

Attributes

Offset `0x09400-0x097FC`

Type RO

Reset 0x00000000

Width 32

The following figure shows the bit assignments.

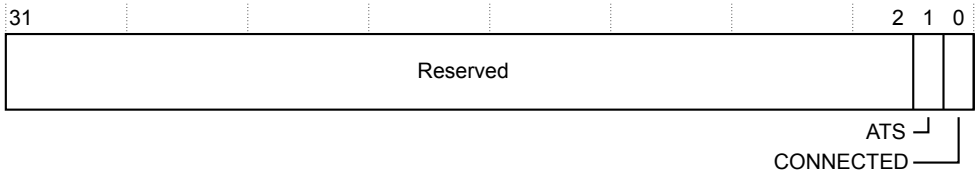

Figure 3-7 TCU_NODE_STATUS register bit assignments

The following table shows the bit descriptions.

Table 3-21 TCU_NODE_STATUS register bit descriptions

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

3.8 TCU RAS registers

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

This section contains the following subsections:

3.8.1 TCU_ERRFR

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

The TCU_ERRFR characteristics are:

Usage constraints

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

Configurations

This register exists in all TCU configurations.

Attributes

Offset `0x08E80`

Type RO

Reset `0x00000081`

Width 32

The following figure shows the bit assignments.

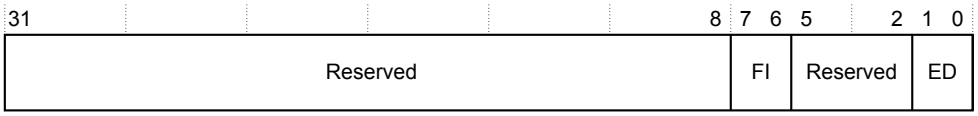

Figure 3-8 TCU_ERRFR register bit assignments

The following table shows the bit descriptions.

Table 3-22 TCU_ERRFR register bit descriptions

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

3.8.2 TCU_ERRCTLR

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

The TCU_ERRCTLR characteristics are:

Usage constraints

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

Configurations

This register exists in all TCU configurations.

Attributes

Offset `0x08E88`

Type RW

Reset `0x00000008`

Width 32

The following figure shows the bit assignments.

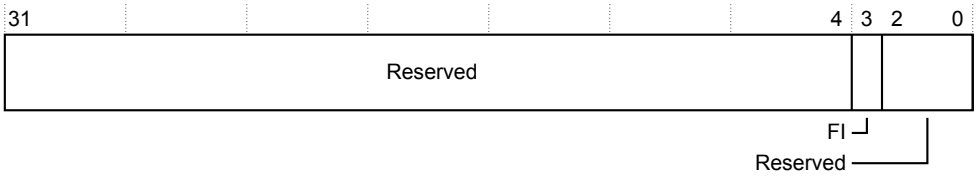

Figure 3-9 TCU_ERRCTLR register bit assignments

The following table shows the bit descriptions.

Table 3-23 TCU_ERRCTLR register bit descriptions

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

3.8.3 TCU_ERRSTATUS

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

The TCU_ERRSTATUS characteristics are:

Usage constraints

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

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

Configurations

This register exists in all TCU configurations.

Attributes

Offset `0x08E90`

Type RW

Reset `0x00000000`

Width 32

The following figure shows the bit assignments.

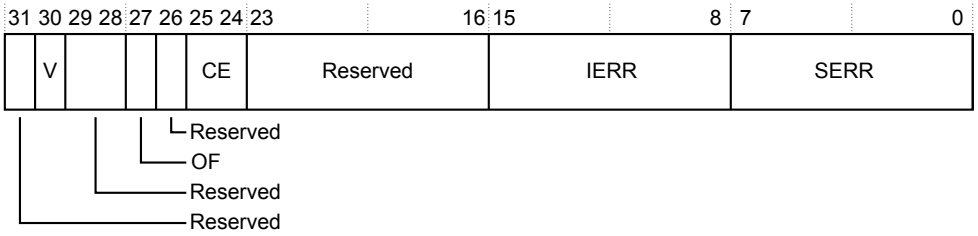

Figure 3-10 TCU_ERRSTATUS register bit assignments

The following table shows the bit descriptions.

Table 3-24 TCU_ERRSTATUS register bit descriptions

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

Table 3-24 TCU_ERRSTATUS register bit descriptions (continued)

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

3.8.4 TCU_ERRGEN

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

Note

The errors that are injected using this mechanism are correctable errors but are not reported in the FMU error records. See:

- *4.5.4 Safety Mechanism table* on page 4-112.
- *Injecting an error into a Safety Mechanism* on page 4-115.

The TCU_ERRGEN characteristics are:

Usage constraints

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

Configurations

This register exists in all TCU configurations.

Attributes

Offset `0x08EC0`

Type RW

Reset `0x00000000`

Width 32

The following figure shows the bit assignments.

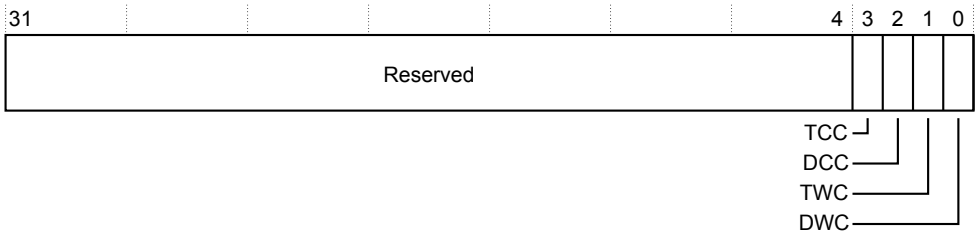

Figure 3-11 TCU_ERRGEN register bit assignments

The following table shows the bit descriptions.

Table 3-25 TCU_ERRGEN register bit descriptions

Bits	Name	Description
[31:4]	-	Reserved.
[3]	TCC	Configuration cache tag parity error: **0** No tag parity error is written to the configuration cache. **1** Entries that are written to the configuration cache include a tag parity error. A fault occurs when the entry is used.
[2]	DCC	Configuration cache data parity error: **0** No data parity error is written to the configuration cache. **1** Entries that are written to the configuration cache include a data parity error. A fault occurs when the entry is used. **Note** Tag parity errors mask data parity errors. When testing data parity error functionality, ensure that software does not set this bit and the TCC bit at the same time.
[1]	TWC	Walk cache tag parity error: **0** No tag parity error is written to the walk cache. **1** Entries that are written to the walk cache include a tag parity error. A fault occurs when the entry is used.
[0]	DWC	Walk cache data parity error: **0** No data parity error is written to the walk cache. **1** Entries that are written to the walk cache include a data parity error. A fault occurs when the entry is used. **Note** Tag parity errors mask data parity errors. When testing data parity error functionality, ensure that software does not set this bit and the TWC bit at the same time.

3.9 TBU component and peripheral ID registers

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

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

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

Register	Offset	Bits	Value	Description
SMMU_PIDR4	0x00FD0	[7:4]	0x0	4KB region count.
		[3:0]	0x4	JEP106 continuation code for Arm.
SMMU_PIDR5	0x00FD4	[7:0]	0x00	Reserved.
SMMU_PIDR6	0x00FD8	[7:0]	0x00	Reserved.
SMMU_PIDR7	0x00FDC	[7:0]	0x00	Reserved.
SMMU_PIDR0	0x00FE0	[7:0]	0x88	Part number[7:0].
SMMU_PIDR1	0x00FE4	[7:4]	0xB	JEP106 ID code[3:0] for Arm.
		[3:0]	0x4	Part number[11:8].
SMMU_PIDR2	0x00FE8	[7:4]	0x1	MMU-600AE major revision. The value 0x1 indicates major product revision r1.
		[3]	0b1	The component uses a manufacturer identity code that JEDEC allocates, according to the JEP106 specification.
		[2:0]	0b011	JEP106 ID code[6:4] for Arm.
SMMU_PIDR3	0x00FEC	[7:4]	MAX[0x0,ecorevnum]	MMU-600AE minor revision. The value 0x0 indicates minor product revision p0.
		[3:0]	0x0	CMOD. This field is not used.
SMMU_CIDR0	0x00FF0	[7:0]	0x0D	Preamble.
SMMU_CIDR1	0x00FF4	[7:0]	0xF0	
SMMU_CIDR2	0x00FF8	[7:0]	0x05	
SMMU_CIDR3	0x00FFC	[7:0]	0xB1	

3.10 TBU PMU component and peripheral ID registers

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

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

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

Register	Offset	Bits	Value	Description
SMMU_PMCG_PMAUTHSTATUS	0x02FB8	[7:0]	0x00	No authentication interface is implemented.
SMMU_PMCG_PIDR4	0x02FD0	[7:4]	0x0	4KB region count.
		[3:0]	0x4	JEP106 continuation code for Arm.
SMMU_PMCG_PIDR5	0x02FD4	[7:0]	0x00	Reserved.
SMMU_PMCG_PIDR6	0x02FD8	[7:0]	0x00	Reserved.
SMMU_PMCG_PIDR7	0x02FDC	[7:0]	0x00	Reserved.
SMMU_PMCG_PIDR0	0x02FE0	[7:0]	0x88	Part number[7:0].
SMMU_PMCG_PIDR1	0x02FE4	[7:4]	0xB	JEP106 ID code[3:0] for Arm.
		[3:0]	0x4	Part number[11:8].
SMMU_PMCG_PIDR2	0x02FE8	[7:4]	0x1	MMU-600AE major revision. The value 0x1 indicates major product revision r1.
		[3]	0b1	The component uses a manufacturer identity code that JEDEC allocates, according to the JEP106 specification.
		[2:0]	0b011	JEP106 ID code[6:4] for Arm.
SMMU_PMCG_PIDR3	0x02FEC	[7:4]	MAX[0x0,ecorevnum]	MMU-600AE minor revision. The value 0x0 indicates minor product revision p0.
		[3:0]	0x0	CMOD. This field is not used.
SMMU_PMCG_CIDR0	0x02FF0	[7:0]	0x0D	Preamble.
SMMU_PMCG_CIDR1	0x02FF4	[7:0]	0x90	
SMMU_PMCG_CIDR2	0x02FF8	[7:0]	0x05	
SMMU_PMCG_CIDR3	0x02FFC	[7:0]	0xB1	

3.11 TBU microarchitectural registers

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

This section contains the following subsections:

3.11.1 TBU_CTRL

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

Its characteristics are:

Usage constraints

When TBU_SCR.NS_UARCH = 0, Non-secure accesses to this register are RAZ/WI. See *3.11.2 TBU_SCR* on page 3-92.

Configurations

This register exists in all TBU configurations.

Attributes

Offset `0x08E00`

Type RW

Reset `0x00000000`

Width 32

The following figure shows the bit assignments.

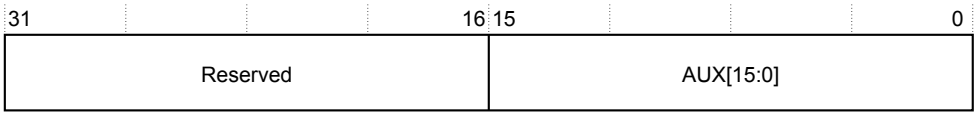

Figure 3-12 TBU_CTRL register bit assignments

The following table shows the bit descriptions.

Table 3-28 TBU_CTRL register bit descriptions

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

3.11.2 TBU_SCR

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

Its characteristics are:

Usage constraints

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

Configurations

This register exists in all TBU configurations.

Attributes

Offset `0x08E18`

Type RW

Reset `0x00000000`

Width 32

The following figure shows the bit assignments.

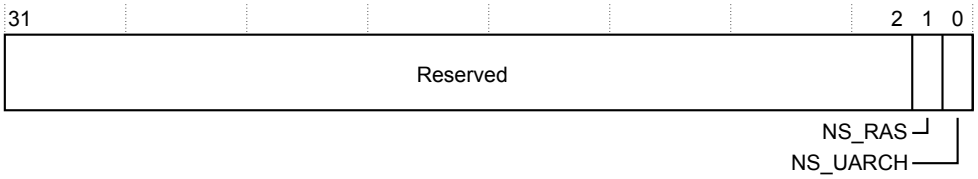

Figure 3-13 TBU_SCR register bit assignments

The following table shows the bit descriptions.

Table 3-29 TBU_SCR register bit descriptions

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

3.12 TBU RAS registers

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

This section contains the following subsections:

3.12.1 TBU_ERRFR

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

The TBU_ERRFR characteristics are:

Usage constraints

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

Configurations

This register exists in all TBU configurations.

Attributes

Offset `0x08E80`

Type RO

Reset `0x00000081`

Width 32

The following figure shows the bit assignments.

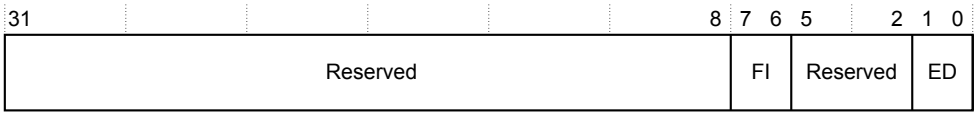

Figure 3-14 TBU_ERRFR register bit assignments

The following table shows the bit descriptions.

Table 3-30 TBU_ERRFR register bit descriptions

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

3.12.2 TBU_ERRCTLR

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

The TBU_ERRCTLR characteristics are:

Usage constraints

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

Configurations

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

Attributes

Offset 0x08E88

Type RW

Reset 0x00000008

Width 32

The following figure shows the bit assignments.

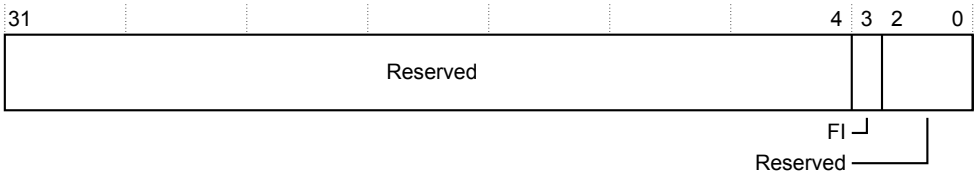

Figure 3-15 TBU_ERRCTLR register bit assignments

The following table shows the bit descriptions.

Table 3-31 TBU_ERRCTLR register bit descriptions

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

3.12.3 TBU_ERRSTATUS

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

The TBU_ERRSTATUS characteristics are:

Usage constraints

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

Configurations

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

Attributes

Offset 0x08E90

Type RW

Reset 0x00000000

Width 32

The following figure shows the bit assignments.

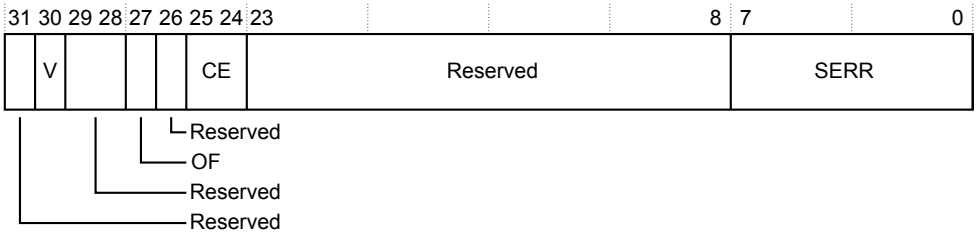

Figure 3-16 TBU_ERRSTATUS register bit assignments

The following table shows the bit descriptions.

Table 3-32 TBU_ERRSTATUS register bit descriptions

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

3.12.4 TBU_ERRGEN

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

Note

The errors that are injected using this mechanism are correctable errors but are not reported in the FMU error records. See:

- *4.5.4 Safety Mechanism table* on page 4-112.
- *Injecting an error into a Safety Mechanism* on page 4-115.

The TBU_ERRGEN characteristics are:

Usage constraints

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

Configurations

This register exists in all TBU configurations.

Attributes

Offset `0x08EC0`

Type RW

Reset `0x00000000`

Width 32

The following figure shows the bit assignments.

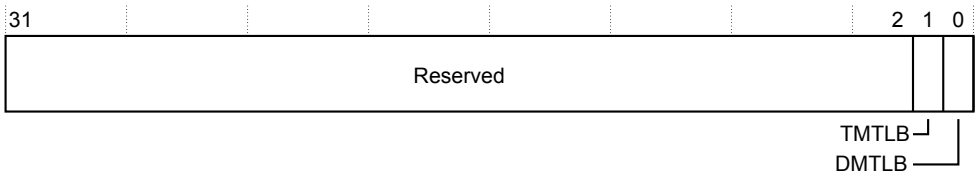

Figure 3-17 TBU_ERRGEN register bit assignments

The following table shows the bit descriptions.

Table 3-33 TBU_ERRGEN register bit descriptions

Bits	Name	Description
[31:2]	-	Reserved.
[1]	TMTLB	Main TLB tag parity error: **0** No tag parity error is written to the Main TLB. **1** Entries that are written to the Main TLB include a tag parity error. A fault occurs when the entry is used.
[0]	DMTLB	Main TLB data parity error: **0** No data parity error is written to the Main TLB. **1** Entries that are written to the Main TLB include a data parity error. A fault occurs when the entry is used. **Note** Tag parity errors mask data parity errors. When testing data parity error functionality, ensure that software does not set this bit and the TMTLB bit at the same time.

Chapter 4
Functional Safety

This chapter describes the *Functional Safety* (FuSa) detection features unique to MMU-600AE.

It contains the following sections:

4.1 Overview

The MMU-600AE is a version of the MMU-600, using the same code base, with *Functional Safety* (FuSa) detection and correction features added. The original MMU-600 logic and functionality are unchanged.

The following sections focus only on the added FuSa features and logic unique to the MMU-600AE.

This section contains the following subsection:

4.1.1 The MMU-600AE Safety Mechanisms

The MMU-600AE provides built-in SMs.

The following figure shows the distribution of the main SMs in the MMU-600AE.

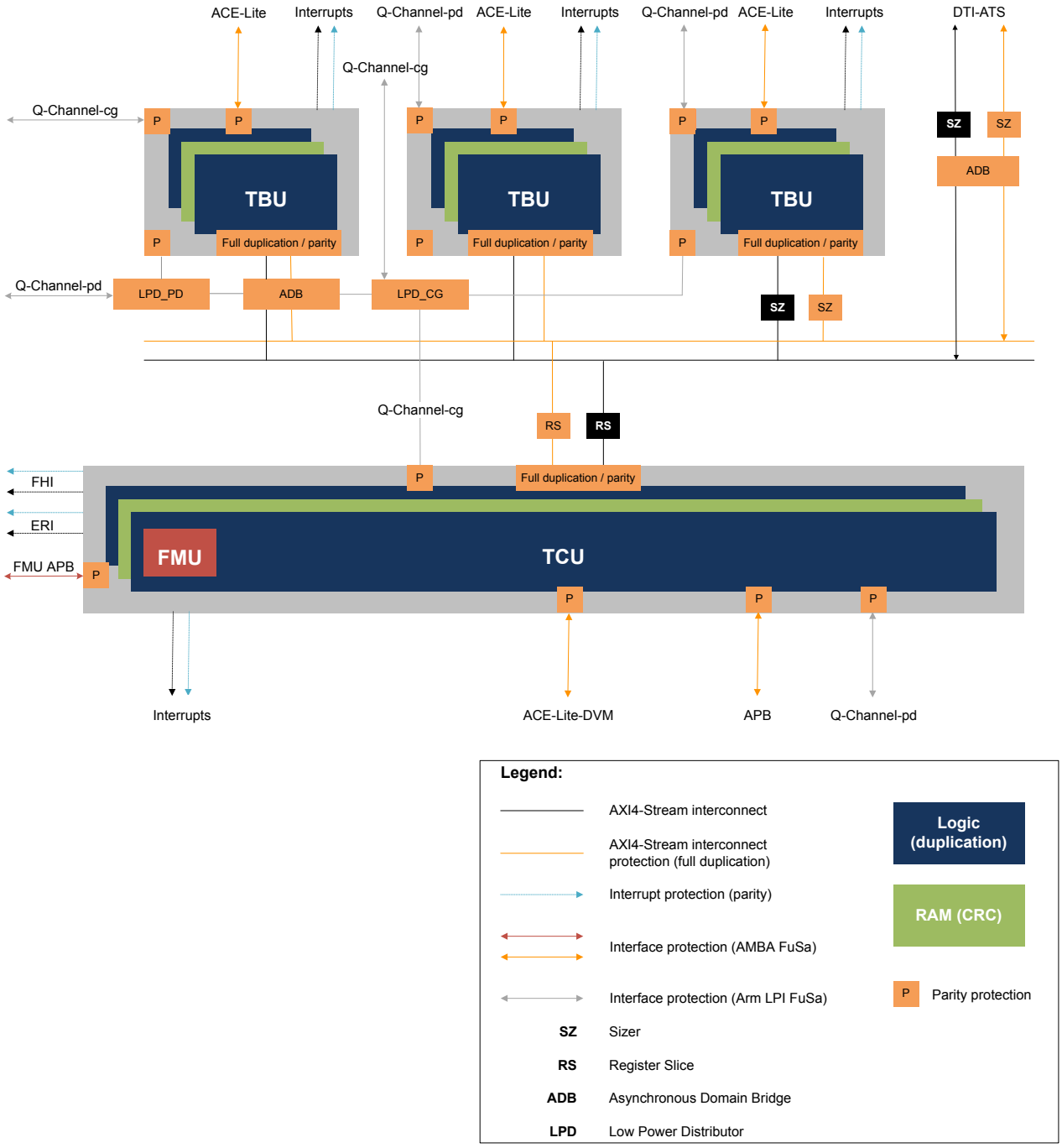

Figure 4-1 Safety Mechanism distribution

The MMU-600AE contains the following main SMs:

Lockstep logic protection

The logic is protected with duplicated logic running in lockstep. See *4.6 Lockstep protection* on page 4-131.

RAM protection

The RAMs are shared between the duplicated blocks and are protected with 8-bit CRC, which covers the address and data. See *4.7 RAM protection* on page 4-133.

AXI4-Stream interconnect protection

The AXI4-Stream interconnect that connects the MMU-600AE blocks is protected by end-to-end duplication.

The following components are protected with partial duplication:

- *AMBA Domain Bridge* (ADB). It has special logic to ensure that the primary and redundant domains are synchronized, and that the outputs have the correct temporal delay.

The following components, which are part of the DTI interconnect, are protected with full duplication:

- DTI Switch
- DTI register slice

Note

It is the responsibility of the integrator to instantiate a redundant instance of these components and stitch the duplicate DTI interconnect, except for the ADB.

See:

- *4.8.2 AXI4-Stream interface parity protection* on page 4-137.
- The *Integrating the TCU, TBU, LPD, PCIe ATS, and DTI AXI4-Stream interconnect* section of the *Arm® CoreLink™ MMU-600AE System Memory Management Unit Technical Reference Manual*.

AMBA external interface protection

All external interfaces are protected with AMBA Parity Extension. AMBA Parity Extension protects point-to-point connections consisting of wires and buffers only, and no gates.
ACE and APB external ports are also included. See:

- *4.8.1 ACE-Lite interface parity protection* on page 4-137.
- *4.8.3 APB interface parity protection* on page 4-138.

Interrupt outputs parity protection

The interrupt outputs from the MMU-600AE are protected with parity protection and are compatible with the Arm CoreLink GIC-600AE Generic Interrupt Controller IP. There is one parity bit for each interrupt output. See the *Interrupt output protection* section of the *Arm® CoreLink™ MMU-600AE System Memory Management Unit Technical Reference Manual*.

Q-Channel protection

The Q-Channel is protected by asynchronous parity. See *4.10 Q-Channel protection* on page 4-142.

Systematic fault watchdog

The MMU-600AE contains a watchdog-based PING/ACK mechanism. This mechanism protects against systematic errors on the interconnect that connects the various MMU-600AE TBU blocks. It works by engaging a hardware mechanism in the TCU that pings each TBU in a round-robin fashion and waits for a response. If the mechanism does not receive a response within the programmable timeout window, it reports a fault. See *4.11 Systematic fault watchdog protection* on page 4-149.

Clocks and resets

The clocks and resets are duplicated. The clocks operate with a Temporal Delay of two cycles. That is, the primary logic operates two cycles ahead of the redundant logic. See the *Clocks and resets* section of the *Arm® CoreLink™ MMU-600AE System Memory Management Unit Configuration and Integration Manual*.

Fault Management Unit

The *Fault Management Unit* (FMU) has the following functions:

- Processes faults that the SMs in the TCU and TBUs detect.
- Records the fault syndrome in the Error Records and reports the fault using the *Error Recovery Interrupt* (ERI) and *Fault Handling Interrupt* (FHI).
- Provides fault injection and clearing for each SM.

The FMU talks to an external Safety Island, using a second APB port that is dedicated to the FMU. The APB port is added for FuSa purposes so that faults can be reported even when the MMU-600 primary logic functionality is either unreliable or inaccessible.
See the *Fault Management Unit* section of the *Arm® CoreLink™ MMU-600AE System Memory Management Unit Technical Reference Manual*.

4.2 FuSa I/Os

Ports have been added for Functional Safety Fault Detection and Control.

This section refers to new interfaces, such as the APB interface, and the protection for existing interfaces. For more information, see *4.8 External interface protection* on page 4-136.

This section contains the following subsections:

4.2.1 Non-architected FuSa ports

The following FuSa ports have been added for *Fault Detection and Control* (FDC).

The following table shows the non-architected FuSa ports.

Table 4-1 Non-architected FuSa ports

Port	Direction	Blocks	Description
aclk_fdc	Input	TCU/TBU	Clock for redundant logic and SMs.
aresetn_fdc	Input	TCU/TBU	Reset for redundant logic and SMs.
mbistresetn_fdc	Input	TCU/TBU	Redundant **nmbistreset**. Both resets must assert together.
fmu_resetn_fdc	Input	TCU	Resets FMU Error Records. Main reset does not reset error records.
dftrstdisable_fdc	Input	TCU/TBU	Prevents reset from asserting when reset generation FDC flops are scanned.
dftcgen_fdc	Input	TCU/TBU	Forces FDC clock gate enable to ensure scanned flops obtain a clock.
dftramhold_fdc	Input	TCU/TBU	Redundant port for **dftramhold**.
fmu_err_out	Output	TBU	Fault indication to the TCU block from a TBU block.
ras_irpt_chk	Output	TCU/TBU	**chk** bit for legacy MMU-600 *Programmer's View* (PV) RAS Interrupt from System MMU.
pmu_irpt_chk	Output	TCU/TBU	**chk** bit for legacy MMU-600 Interrupt from System MMU.
event_q_irpt_s_chk	Output	TCU	**chk** bit for legacy MMU-600 Interrupt from System MMU.
event_q_irpt_ns_chk	Output	TCU	**chk** bit for legacy MMU-600 Interrupt from System MMU.
pri_q_irpt_ns_chk	Output	TCU	**chk** bit for legacy MMU-600 Interrupt from System MMU.
cmd_sync_irpt_ns_chk	Output	TCU	**chk** bit for legacy MMU-600 Interrupt from System MMU.
cmd_sync_irpt_s_chk	Output	TCU	**chk** bit for legacy MMU-600 Interrupt from System MMU.

Table 4-1 Non-architected FuSa ports (continued)

Port	Direction	Blocks	Description
global_irpt_ns_chk	Output	TCU	**chk** bit for legacy MMU-600 Interrupt from System MMU.
global_irpt_s_chk	Output	TCU	**chk** bit for legacy MMU-600 Interrupt from System MMU.
evento_chk	Output	TCU	**chk** bit for legacy MMU-600 **evento** output.
fmu_fault_int	Output	TCU	FHI Interrupt from TCU FMU to Safety Island.
fmu_fault_int_chk	Output	TCU	**chk** bit for FHI Interrupt from TCU FMU to Safety Island.
fmu_err_int	Output	TCU	ERI Interrupt from TCU FMU to Safety Island.
fmu_err_int_chk	Output	TCU	**chk** bit for ERI Interrupt from TCU FMU to Safety Island.
freq[TCUCFG_FUSA_FCHAN_COUNT−1:0]	Input	TCU	Fault Indicator Request from ADB/LPD to TCU.
fack[TCUCFG_FUSA_FCHAN_COUNT−1:0]	Output	TCU	Fault Indicator Ack from TCU to ADB/LPD.
freq_chk[TCUCFG_FUSA_FCHAN_COUNT−1:0]	Input	TBU	Redundant Fault Indicator Request from ADB/LPD to TCU.
fack_chk[TCUCFG_FUSA_FCHAN_COUNT−1:0]	Output	TBU	Redundant Fault Indicator Ack from TCU to ADB/LPD.
fmu_err_in[TCUCFG_FUSA_TBU_FAULT_WIRE_COUNT−1:0]	Input	TBU	Fault indicator input from individual TBUs.
pcie_mode_chk	Input	TBU	Redundant tie-offs, opposite polarity of original tie-offs.
ns_sid_high_chk	Input	TBU	Redundant tie-offs, opposite polarity of original tie-offs.
s_sid_high_chk	Input	TBU	Redundant tie-offs, opposite polarity of original tie-offs.
cmo_disable_chk	Input	TBU	Redundant tie-offs, opposite polarity of original tie-offs.
max_tok_trans_chk	Input	TBU	Redundant tie-offs, opposite polarity of original tie-offs.
utlb_roundrobin_chk	Input	TCU/TBU	Redundant tie-offs, opposite polarity of original tie-offs.
sec_override_chk	Input	TCU	Redundant tie-offs, opposite polarity of original tie-offs.
sup_cohacc_chk	Input	TCU	Redundant tie-offs, opposite polarity of original tie-offs.
sup_btm_chk	Input	TCU	Redundant tie-offs, opposite polarity of original tie-offs.

Table 4-1 Non-architected FuSa ports (continued)

Port	Direction	Blocks	Description
sup_sev_chk	Input	TCU	Redundant tie-offs, opposite polarity of original tie-offs.
sup_oas_chk	Input	TCU	Redundant tie-offs, opposite polarity of original tie-offs.

4.2.2 Q-Channel FuSa ports

The following interfaces add **_chk** bits, as specified in the Arm Q-Channel Parity Extensions.

For more information, see *4.10 Q-Channel protection* on page 4-142.

Table 4-2 Q-Channel FuSa ports

Port	Direction	Blocks	Description
qreqn_chk_pd	Input	TCU/TBU	Redundant **qreqn** port for Q-Channel power controller
qactive_chk_pd	Output	TCU/TBU	Redundant **qactive** port for Q-Channel power controller
qaccept_chk_pd	Output	TCU/TBU	Redundant **qaccept** port for Q-Channel power controller
qdeny_chk_pd	Output	TCU/TBU	Redundant **qdeny** port for Q-Channel power controller
qreqn_chk_cg	Input	TCU/TBU	Redundant **qreqn** port for Q-Channel clock controller
qactive_chk_cg	Output	TCU/TBU	Redundant **qactive** port for Q-Channel clock controller
qaccept_chk_cg	Output	TCU/TBU	Redundant **qaccept** port for Q-Channel clock controller
qdeny_chk_cg	Output	TCU/TBU	Redundant **qdeny** port for Q-Channel clock controller

4.2.3 AMBA interface FuSa ports

The following interfaces add **_chk** bits, as the AMBA Parity Extensions specify.

For more information, see *4.8 External interface protection* on page 4-136.

The APB port was added for FDC between the FMU block residing in the TCU and the Safety Island in the SoC.

Table 4-3 AMBA interface FuSa ports

Port	Granularity	Description
APB interface	TCU	APB4 interface added for FMU as the *4.8 External interface protection* on page 4-136 section describes.
AXI4-Stream AMBA parity	TCU/TBU	AMBA Parity added to all external AXI4-Stream interfaces as the *4.8 External interface protection* on page 4-136 section describes.
ACE AMBA parity	TCU/TBU	AMBA Parity added to all external ACE-Lite interfaces as the *4.8 External interface protection* on page 4-136 section describes.

4.3 Clocks and resets

The MMU-600AE clocks and resets are identical to those of the MMU-600, except for:

- The added redundant clock and reset.
- The added **fmu_resetn** and **fmu_resetn_fdc** signals.

The following figure shows how the redundant clock and reset is used by the FDC logic. The extra **aresetn_fdc** and **aclk_fdc** signals provide redundancy in the clock and reset trees while guarding against faults on the tree branches. If a fault occurs on a branch in either the primary or FDC clock trees, the *Dual LockStep* (DLS) comparators detect the fault.

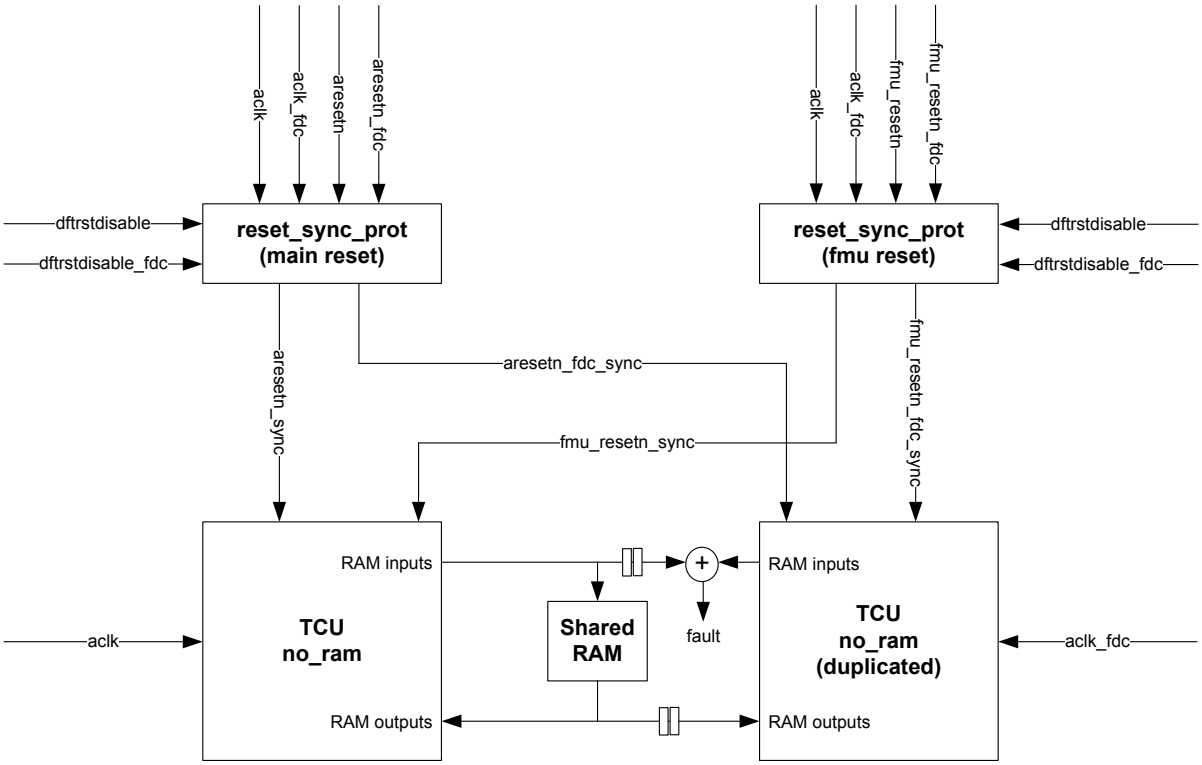

Figure 4-2 MMU-600AE block resets

The following conditions apply to a TCU block with DLS:

- Internal **_sync** resets are asynchronous-assert.
- Internal **_sync** resets are synchronous-deassert.
- The **aresetn_fdc_sync** and **fmu_resetn_fdc_sync** signals are deasserted two cycles after the non-FDC signals.

The **fmu_resetn** and **fmu_resetn_fdc** signals reset only the FMU registers. Asserting only the **aresetn** and **aresetn_fdc** signals resets all registers except for the FMU PV registers. This enables software to view the contents of the registers after reset. Both **aresetn** and **fmu_resetn**, and their **_fdc** counterparts, should be reset during a Cold reset or when debug information from the FMU PV registers is not required.

For more information on integrating the FuSa clocks and resets, see the *FuSa clocks and resets* section of the *Arm® CoreLink™ MMU-600AE System Memory Management Unit Configuration and Integration Manual*.

4.4 DFT protection

Functional Safety Mechanisms have been added to protect the MBIST and ATPG Scan logic from faults during functional mode.

This section contains the following subsections:

4.4.1 MBIST

The MBIST wrapper logic built into the MMU-600AE is duplicated, which lets the mechanism detect faults in this logic.

For example, it can detect a fault on a RAM address bit, due to the comparators at the inputs of the shared RAMs.

The MBIST interface inputs in the following table can cause mission-mode errors, so they are protected.

Table 4-4 Protected MBIST inputs

Signal	Protection	Notes
mbistreq	Assertion detection	If **mbistreq** is asserted when the MMU block is in functional mode, the MMU detects and reports it. If this happens in MBIST mode, it is assumed the fault is ignored and cleared by software, using a reset or the FMU clearing mechanism. Alternatively, to prevent the fault from asserting, software can disable the SM before entering MBIST mode.
nmbistresetn	Duplication	The reset is duplicated. The duplicated reset is **nmbistresetn_fdc**. For a reset to occur, both **nmbistresetn** and **nmbistresetn_fdc** must be asserted. For more information, see *Resets* of the *FuSa clocks and resets* section of the *Arm® CoreLink™ MMU-600AE System Memory Management Unit Configuration and Integration Manual*.

Note

The MBIST interface pins themselves are unchanged from the MMU-600.

The other MBIST inputs, including **mbistaddr** and **mbistindata**, are benign and cause no harm if they experience faults during functional mode.

If faults occur on the MBIST controller or MBIST signals, it is assumed that the MBIST controller detects them.

4.4.2 ATPG Scan

All DFT and ATPG input ports are duplicated.

These duplicate ports allow the SoC integrator to have separate scan chains for **clk** and **clk_fdc**, if wanted. If the scan chains are shared by **clk** and **clk_fdc** flops, drive the duplicate ports in the same way at the same time.

The following table summarizes the duplicate ports.

Table 4-5 Duplicate ATPG input ports

clk scan input	clk_fdc scan input	Description
dftrstdisable	**dftrstdisable_fdc**	Prevents reset from asserting when reset generation flops are scanned.
dftcgen	**dftcgen_fdc**	Forces clock gate enable to ensure scanned flops get a clock
dftramhold	**dftramhold_fdc**	Asserting prevents RAM access during ATPG. This can reduce coverage for logic in the RAM shadow.

4.4.3 LBIST

Arm has verified that a third-party LBIST controller can be instanced and used to control the scan chains and obtain additional latent fault coverage or diagnostic information.

4.5 Fault Management Unit

The FMU is located in the TCU. It processes faults that the TCU and TBU SMs detect.

The FMU implements the following functionality:

- Receives errors signaling from all SMs within other MMU blocks.
- Routes all errors to the Safety Island, if enabled.
- Maintains Error Records for each MMU block, which are stored in registers.
- The Error Records and other registers are accessible through a dedicated, safe APB4 interface.
- It allows software to enable or disable an SM within an MMU block.
- Enables software error recovery testing by allowing error injection in an SM within an MMU block.
- Retains Error Records across functional reset.

The following figure shows the FMU interconnections.

Note

This is a dedicated APB port for the FMU, which is different than the APB port accessing the programmable registers of MMU-600.

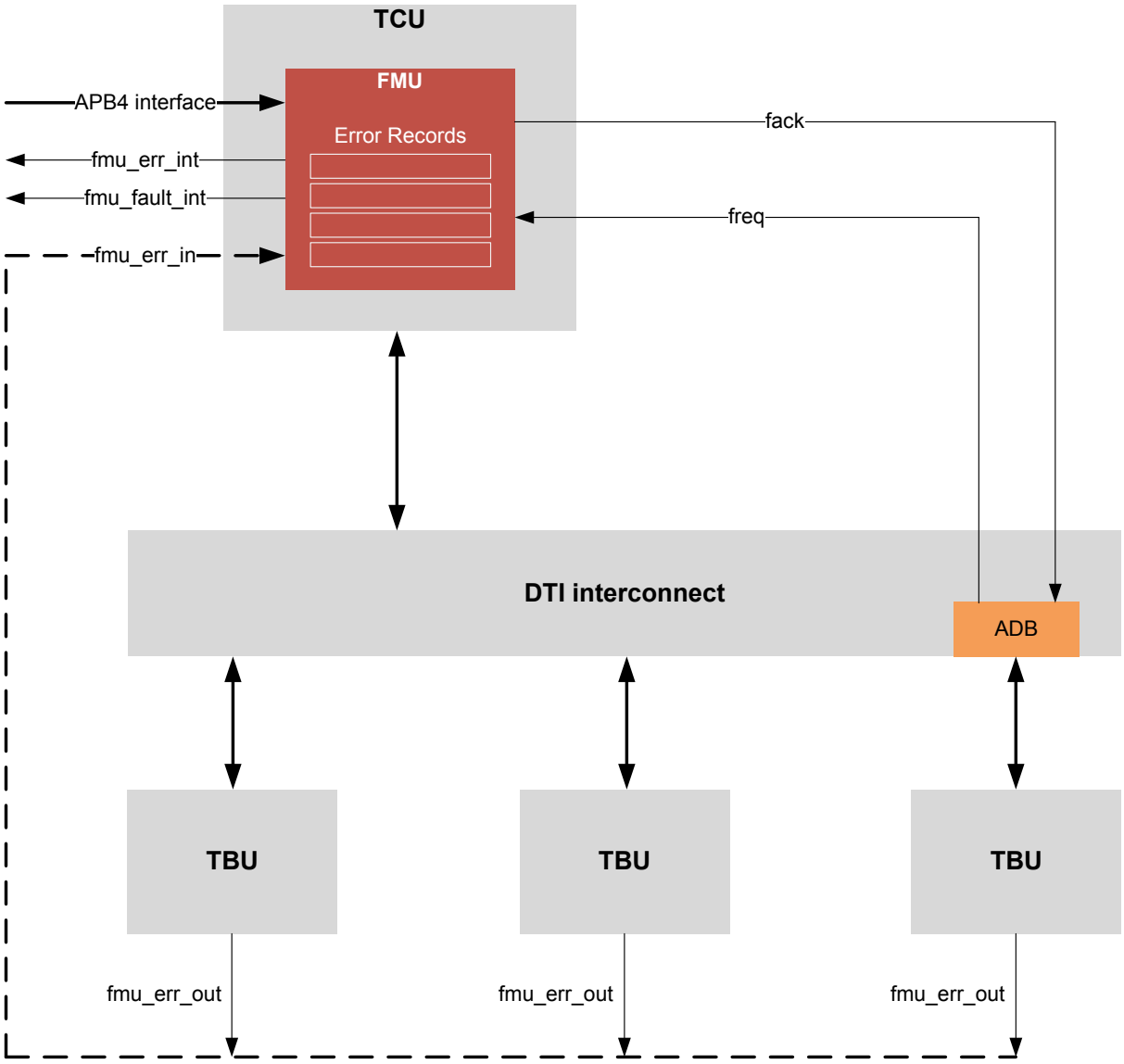

Figure 4-3 FMU in MMU block

This section contains the following subsections:

- *4.5.1 Error signaling to the FMU* on page 4-110.
- *4.5.2 Error signaling from the FMU* on page 4-111.
- *4.5.3 Error Record table* on page 4-111.
- *4.5.4 Safety Mechanism table* on page 4-112.
- *4.5.5 Software interaction* on page 4-116.
- *4.5.6 Lock and key mechanism* on page 4-117.
- *4.5.7 Ping mechanism* on page 4-118.
- *4.5.8 Correctable Error enable* on page 4-120.
- *4.5.9 Programmer's View* on page 4-120.

4.5.1 Error signaling to the FMU

The MMU-600AE deploys numerous SMs in each MMU TCU and TBU block to protect them from transient or permanent errors.

Each SM outputs an error signal which then sends those errors to the FMU residing in the TCU.

The TBU uses the existing AXI4-Stream interconnect to report errors that any SM within the block to the TCU detects.

In addition to reporting errors through the AXI4-Stream interconnect, each TBU has an output, **fmu_err_out**, which indicates an actual uncorrected error within its block. Corrected errors never raise **fmu_err_out**, even if configured to report as uncorrected. See *4.5.8 Correctable Error enable* on page 4-120. The **fmu_err_out** signal must be connected to the **fmu_err_in** input of the TCU to provide a redundant path for error signaling from the TBUs to the FMU residing in the TCU. The **fmu_err_in** wire stays asserted by the TBU until the error recovery software clears the error.

Smaller components without an AXI4-Stream interface needing to report errors do so through the F-Channel interface. The TCU can be connected to 16 such interfaces. The MMU-600AE has modified the ADB and LPD components with an F-Channel interface for this reason. For more information, see *4.8.4 F-Channel* on page 4-138.

4.5.2 Error signaling from the FMU

When an SM detects an error, it is sent to the FMU.

If enabled, the FMU signals the error to the outside world using the level-sensitive interrupt error ports:

- *Error Recovery Interrupt* (ERI), **fmu_err_int**.
- *Fault Handling Interrupt* (FHI), **fmu_fault_int**.

Error reporting through the ERI or FHI is enabled through the *FMU_ERR<n>CTLR, Error Record Control Register* on page 4-122.

Note

These error interrupts are disabled after reset by **fmu_aresetn** and software must enable them at startup.

Detected Uncorrectable Errors can be reported as ERI, FHI, or both when enabled. Detected Correctable Errors can be reported as FHI when enabled. FMU_ERR<n>CTLR.FI and FMU_ERR<n>CTLR.UI control this reporting. The grouping of the errors into these two categories can be helpful in redirecting these errors to different error recovery handlers based on the criticality of the errors or other factors that are known at the system level.

Reset

The **fmu_aresetn** signal resets the FMU Error Records and FMU_ERRGSR. The **aresetn** signal resets the main FMU logic and all other registers, including locking the register file.

This enables software to reset the MMU while keeping the Error Records intact for later interrogation and debug. For more information, see the *Resets* section of the *Arm® CoreLink™ MMU-600AE System Memory Management Unit Configuration and Integration Manual*.

Note

If the FMU Error Records are not reset, but the rest of the system is reset, then an error record with FMU_ERR<n>STATUS.V==1 continues to assert an interrupt if the appropriate control bit, FMU_ERR<n>CTLR.UI or FMU_ERR<n>CTLR.FI enables the interrupt. The system reset signal does not affect the interrupt, and when the system comes out of reset, then the interrupt is still asserted.

If the FMU error records are not reset, then when the system comes out of reset, any new RAS errors that occur from system reset are logged if enabled by FMU_ERR<n>CTLR.ED.

4.5.3 Error Record table

The MMU-600AE faults are recorded in Error Records.

Each MMU block that the *MMU block IDs* table in *Functional PV and FMU PV* on page 4-112 shows has its own error record where the index of the error record is the block ID. Each error record contains several registers and the table *FMU PV registers* in *Programmer's View registers* on page 4-121 describes their offsets.

Accessing FMU Error Records using the FMU APB

These FMU Error Records are accessed only by the dedicated FMU APB interface on the TCU.

This interface is separated from the main functional APB interface to minimize the chance of main logic or interface faults affecting the FDC logic.

Functional PV and FMU PV

Non-FMU registers, such as those used to configure and control the MMU functions, are accessed through the main APB interface on the TCU.

For any CRC or parity faults in:

- The TBU MTLB Tag and Data RAMs.
- The TCU CCB and WCB Tag and Data RAMs.

The MMU-600 PV detects and reports these faults as Corrected Errors in the MMU-600 PV RAS registers. These faults are also Safety Mechanisms TBU SMID:7-8 and TCU SMID:7-10 and an attempt is made to report them independently in the FMU as either Corrected Errors or Uncorrected Errors depending on the corresponding FMU_ERR<n>CTLR.CE_EN field. See:

- *FMU_ERR<n>CTLR, Error Record Control Register* on page 4-122.
- *4.5.4 Safety Mechanism table* on page 4-112.

This method enables the host processor and a dedicated FDC processor to act on faults separately and perform different tasks. If this functionality is not required, the host software can disable the interrupts that are associated with the functional Error Records.

Table 4-6 MMU block IDs

Block ID	MMU block
0	TCU.
1	Fault Channels. LPD and ADB faults are reported in this Error Record.
2	TBU0.
3	TBU1.
4	TBU2.
...	...
61	TBU59.
62	TBU60.
63	TBU61.

The number of Error Records that are supported depends on the number of TBU blocks that are connected to the TCU. For unused TBU blocks, the Error Record registers become RAZ.

For more information about these Error Records, see *4.5.9 Programmer's View* on page 4-120.

4.5.4 Safety Mechanism table

If an uncorrectable error occurs, then the system might behave unpredictably.

For faults in RAMs, unpredictable behavior can include, but is not limited to:

- Data corruption.
- System lock-up in any of the upstream or downstream systems of the TBUs and DTI masters.
- Security violation.

For interface errors, unpredictable behavior can include any of the above behaviors. In addition, depending on the interface, the unpredictable behavior can also include, but is not limited to:

- Potentially, using and/or reporting bad data and addresses to access the FMU register file.
- No response to ping packets.
- FMU_ERRSTATUS.idle never returning to 1, that is, the idle state. See *FMU_ERR<n>STATUS, Error Record Primary Status Register* on page 4-123.

The following table shows the *Safety Mechanisms* (SMs) inside the MMU blocks.

Table 4-7 Safety Mechanisms

Block	SM identifier	SM description
TCU	0	Reserved.
	1	TCU Dual LockStep error. See *4.6 Lockstep protection* on page 4-131.
	2	TCU DTI AXI4-Stream interface error. See *4.8.2 AXI4-Stream interface parity protection* on page 4-137 and *4.9 Integrating the TCU, TBU, LPD, PCIe ATS, and DTI AXI4-Stream interconnect* on page 4-141.
	3	TCU APB interface error. See *4.8.3 APB interface parity protection* on page 4-138.
	4	TCU ACE-Lite DVM master interface error. See *4.8.1 ACE-Lite interface parity protection* on page 4-137.
	5	TCU LPI_PD Q-Channel interface error.
	6	TCU LPI_CG Q-Channel interface error.
	7	TCU *Configuration Cache Block* (CCB) Tag RAM CRC error. See *4.5.8 Correctable Error enable* on page 4-120 and *4.7 RAM protection* on page 4-133.
	8	TCU CCB Entry RAM CRC error. See *4.5.8 Correctable Error enable* on page 4-120 and *4.7 RAM protection* on page 4-133.
	9	TCU *Walk Cache Block* (WCB) Tag RAM CRC error. See *4.5.8 Correctable Error enable* on page 4-120 and *4.7 RAM protection* on page 4-133.
	10	TCU WCB Entry RAM CRC error. See *4.5.8 Correctable Error enable* on page 4-120 and *4.7 RAM protection* on page 4-133.
	11	TCU *DTI Buffer* (DTIB) RAM CRC error. Stores messages received over DTI.
	12	FMU Ping Ack error.
	13	FMU APB interface error. See *4.8.3 APB interface parity protection* on page 4-138.
	14	MBIST Req error. This is disabled by default.
	15	Tie-off error. See *4.8.6 Tie-off input protection* on page 4-140.
	16	FMU clock gating override[a]. This is disabled by default.

[a] The scope of FMU clock gating override is limited to the FMU and does not apply to the parent module, that is, the TCU or the TBU.

Table 4-7 Safety Mechanisms (continued)

Block	SM identifier	SM description
TBU	0	Some error in TBU, precise source yet unknown.
	1	TBU Dual LockStep error. See *4.6 Lockstep protection* on page 4-131.
	2	TBU ACE-Lite Slave (TBS) interface error. See *4.8.1 ACE-Lite interface parity protection* on page 4-137.
	3	TBU ACE-Lite Master (TBM) interface error. See *4.8.1 ACE-Lite interface parity protection* on page 4-137.
	4	TBU DTI AXI4-Stream interface error. See *4.8.2 AXI4-Stream interface parity protection* on page 4-137 and *4.9 Integrating the TCU, TBU, LPD, PCIe ATS, and DTI AXI4-Stream interconnect* on page 4-141.
	5	TBU LPI_PD Q-Channel interface error. *Low-Power Interface Power-Down* (LPI_PD) represents a request from the power-controller to alter the power state of the device.
	6	TBU LPI_CG Q-Channel error. *Low-Power Interface Clock Gating* (LPI_CG) might be a software-based or HW-based request to go into a lower power state by clock gating.
	7	TBU MTLB Tag RAM CRC error. MTLB is a TLB-like structure for caching translations. See *4.5.8 Correctable Error enable* on page 4-120 and *4.7 RAM protection* on page 4-133.
	8	TBU MTLB Entry RAM CRC error. See *4.5.8 Correctable Error enable* on page 4-120 and *4.7 RAM protection* on page 4-133.
	9	TBU WBB MFIFO RAM CRC error. WBB is the store buffer for client data.
	10	MBIST Req error. This is disabled by default.
	11	Tie-off error. See *4.8.6 Tie-off input protection* on page 4-140.
	12	FMU clock gating override[a]. This is disabled by default.
F-Channel	id	Index of **freq**[id].

Note

The SMID value 0 for TBU error record[N] indicates that the FMU has detected an actual uncorrected error in the TBU[N-2], as indicated by the **fmu_err_out** of the TBU that is raising an error on the **fmu_err_in** of the FMU. Corrected errors never raise **fmu_err_out**, even if configured to report as uncorrected. See *4.5.1 Error signaling to the FMU* on page 4-110. The safety mechanism that reported this error has still not been determined. The safety mechanism that reported the error is updated after the safety mechanism in the TBU sends this information over the DTI interface to the FMU in the TCU. This information is then updated in the FMU_ERR<n>STATUS.IERR field. See *FMU_ERR<n>STATUS, Error Record Primary Status Register* on page 4-123.

If a software read of FMU_ERR<n>STATUS.IERR returns SMID:0, then the software is expected to read this register again. If repeated reads of this register do not indicate the safety mechanism other than SMID:0, then it might indicate that the DTI interconnect is broken, possibly because of a permanent fault, and is unable to receive DTI messages. The error recovery software does not have the SM information from the TBU that had this fault. The software should therefore perform error recovery by resetting that TBU and the DTI interconnect components.

Enabling and disabling a Safety Mechanism

All SMs are enabled on reset, except for MBIST Req and FMU clock gating override.

To enable or disable an SM, write to the FMU_SMEN register. FMU_SMEN.BLK specifies the MMU block. FMU_SMEN.SMID specifies the SM within the MMU block to be enabled or disabled.

When a block is powered-down and then powered-up again, then the enabled state of the Safety Mechanisms in that block return to the reset value and it might be necessary for software to redisable or reenable them.

Injecting an error into a Safety Mechanism

To inject an error into a *Safety Mechanism* (SM), write to the FMU_SMINJERR register. FMU_SMINJERR.BLK specifies the MMU block, and FMU_SMINJERR.SMID specifies the SM into which to inject the error.

FMU_STATUS.idle protects the FMU_SMINJERR register. See:

- *FMU idle* on page 4-117.
- *FMU_SMINJERR, Safety Mechanism Inject Error Register* on page 4-129.
- *FMU_STATUS, FMU Status Register* on page 4-130.

This method injects only one error. No clearing of error injection is required.

Errors that are injected into the following are always corrected:

- TCU CCB and WCB Tag and Data RAMs, TCU SMID: 7-10.
- TBU MTLB Tag and Data RAMs, TBU SMID: 7-8.

Correction occurs irrespective of whether an attempt is made to report the errors as corrected or uncorrected in the FMU error records, which FMU_ERR<n>CTLR.CE_EN controls. See:

- *4.5.8 Correctable Error enable* on page 4-120.
- *FMU_ERR<n>CTLR, Error Record Control Register* on page 4-122.

Errors that are injected into these correctable RAMs are injected into only the FMU path and therefore:

- The corresponding errors are not reported in the *3.8.3 TCU_ERRSTATUS* on page 3-86 and *3.12.3 TBU_ERRSTATUS* on page 3-95 MMU-600 PV RAS registers. See also *4.7.2 RAM fault reporting* on page 4-134.
- Do not lead to the corrective action of invalidating and refetching those RAM entries.

You can use this mechanism to test the software that is handling the FMU error records without disturbing the software that is handling the MMU-600 PV RAS registers. However, the mechanism is distinguishable from a true error occurring in these RAMs.

For errors that are injected into these correctable RAMs using *3.8.4 TCU_ERRGEN* on page 3-88 and *3.12.4 TBU_ERRGEN* on page 3-96, no attempt is made to report them in the FMU. These errors are only reported in the MMU-600 PV RAS registers. This enables the RAS software to be tested independently of the FMU, but is distinguishable from a true error occurring in these RAMs.

Errors that are injected into other RAMs are never corrected and the MMU-600 PV RAS registers never report them. The MMU-600 PV RAS registers are also unable to inject errors into these RAMs. An attempt is made to report these errors as Uncorrected Errors in the FMU error records. Because the error injection did not corrupt the RAM data, the SMMU continues to function correctly. However, a real error in these RAMs might lead to the outcomes that the beginning of *4.5.4 Safety Mechanism table* on page 4-112 describes.

Errors that are injected into the Ping Ack mechanism (TCU SMID:12) do not give software the opportunity to choose the block ID that is reported in FMU_ERR<n>STATUS.BLKID if it was a real error. This field is unchanged.

Note

Attempting to inject an error into the F-Channel has no effect. To simulate such an error, send a ping message to inject errors along the path that has the ADB/LPD that is connected to the required F-Channel. See *Directed ping* on page 4-119 in *4.5.7 Ping mechanism* on page 4-118.

There might be multiple components along the ping path that have an F-channel interface. If the ping packet is marked as tbu_inject_error, each component attempts to raise an error. Usually, the ping packet travels sufficiently quickly that the first component is recorded in ERR1STATUS.IERR, and subsequent

components cause an overflow in ERR1STATUS. Subsequent components can only be observed if software is fast enough to acknowledge the ERR1STATUS register before the subsequent components report an error.

——— **Caution** ———

Do not inject error into the FMU clock gating override SM.

4.5.5 Software interaction

This section describes software interaction with the FMU.

Initialization

The initialization routine can use the FMU_ERRIDR register to understand the number of implemented Error Records.

It can further iterate over the FMU_ERR<n>FR, Error Record Feature Registers to understand the capabilities of each Error Record. See *FMU_ERR<n>FR, Error Record Feature Register* on page 4-122.

——— **Note** ———

All SMs are enabled on reset which can lead to errors being logged in the Error Records. If the system does not support or does not want to check a particular safety feature, then the software should disable that SM. To disable an SM, write to the *FMU_SMEN, Safety Mechanism Enable Register* on page 4-128 and specify the block ID and the SM ID.

Because most of the safety mechanisms are enabled by default on startup, then they might have already recorded some errors. The initialization routine should analyze the FMU_ERR<n>STATUS registers to deal with any errors that have been found so far. See *FMU_ERR<n>STATUS, Error Record Primary Status Register* on page 4-123.

Clear all logged errors by writing ones to the bits that are asserted in the *FMU_ERR<n>STATUS, Error Record Primary Status Register* on page 4-123.

To enable reporting of errors through the interrupts FHI and/or ERI, write one to FMU_ERR<n>CTLR.FI and FMU_ERR<n>CTLR.UI. See *FMU_ERR<n>CTLR, Error Record Control Register* on page 4-122.

Interrupt handler

When an interrupt, ERI or FHI, is received, the interrupt handling software can identify the Error Record ID, or the MMU block, that is reporting the error by reading the FMU_ERRGSR register.

See *FMU_ERRGSR, Error Group Status Register* on page 4-125.

FMU_ERRGSR[*N*] being one indicates that Error Record *N* holds a valid error. See *FMU_ERR<n>STATUS, Error Record Primary Status Register* on page 4-123.

The FMU_ERR<*N*>STATUS.IERR indicates the Safety Mechanism ID which reported the error.

In the case where this block has reported more than one error of the same type to this Error Record, FMU_ERR<*N*>STATUS.OF is set to 1.

In the case where this block attempts to report another error while it is still reporting an unacknowledged error, then see *Prioritized ERR<n>STATUS registers* on page 4-125.

When the recovery procedure is complete, the errors from this Error Record can be acknowledge by writing an appropriate value to FMU_ERR<n>STATUS. See *FMU_ERR<n>STATUS, Error Record Primary Status Register* on page 4-123 for more information.

FMU idle

The APB port to the FMU is designed to not introduce backpressure by deasserting **pready**. This measure prevents software lockup and makes the Error Records always accessible.

There are several operations which take multiple clock cycles to complete within the FMU. The FMU frees up the APB bus by asserting **pready** to complete the APB transaction. However, it might still be processing the previous request. When software writes one of the following registers inside the FMU, it is required to poll for the FMU_STATUS.IDLE == 1 before issuing another write to these registers. The APB writes, which require a poll of FMU_STATUS.IDLE == 1, are:

- FMU_ERR<n>STATUS.
- FMU_SMEN.
- FMU_SMINJERR.
- FMU_PINGNOW.

Power management

The TBU LP_PD Q-Channel interface can power down the TBU.

Writing to the following registers generates messages to the remote TBU block:

- *FMU_ERR<n>STATUS, Error Record Primary Status Register* on page 4-123.
- *FMU_PINGNOW, Ping Now Register* on page 4-127.
- *FMU_SMEN, Safety Mechanism Enable Register* on page 4-128.
- *FMU_SMINJERR, Safety Mechanism Inject Error Register* on page 4-129.

The software should be aware of the power state of the remote TBUs and does not initiate writes to these registers that are targeting a powered-down TBU. If writes are initiated to the preceding registers, that are targeting a powered-off TBU, the write does not come into effect.

The following apply:

FMU_ERR<n>STATUS	The write is ignored for all purposes. FMU_ERR<n>STATUS is unchanged.
FMU_PINGNOW	The write is ignored for all purposes other than reading back the register. It does not send a PING packet and does not indicate that the FMU is non-idle through FMU_STATUS.
FMU_SMEN	The write is ignored for all purposes.
FMU_SMINJERR	The write is ignored for all purposes.

4.5.6 Lock and key mechanism

The FMU registers are protected against inadvertent writes by a lock and key mechanism.

The FMU registers are in a locked state after reset. If the register file is locked, then any Write-Access to any register other than *FMU_KEY, FMU Key Register* on page 4-126 is ignored.

The register file is unlocked when a write to FMU_KEY occurs that satisfies all of the following:

- Is Secure.
- Is for 32 bits. That is, all write strobes.
- The bottom 8 bits are `0xBE`.

The register file is locked again when a write occurs that satisfies all of the following:

- Is a Secure write.
- Is any width and any write strobes.
- Is to any register except for FMU_KEY.

A write to the *FMU_KEY, FMU Key Register* on page 4-126, when unlocked, leaves the register file unlocked only if the write satisfies the criteria for unlocking the register file. Otherwise, it locks the register file.

If the register file is unlocked, the FMU_KEY register reads as `0x00000BE`. Otherwise, the FMU_KEY register reads as `0x00000000`.

Note

Non-secure accesses never succeed and never affect the locked state of the register file.

Some of the FMU registers are 64-bit registers, but the APB interface width is 32 bits. When in unlocked state, the FMU allows for two consecutive writes to update the same 64-bit register without requiring unlocking again before the second write. In this sequence, both the writes are Secure, with all write strobes to the same register, except that the second write is targeting the other half of that register.

For example, the following sequence would be successful in updating the register contents:

1. Secure write to FMU_KEY with data `0xBE`, all write strobes asserted.
2. 32-bit Secure write to FMU_ERR0CTLR[63:32] addr `0x0C`, all write strobes asserted.
3. 32-bit Secure write to FMU_ERR0CTLR[31:0] addr `0x08`, all write strobes asserted.

This behavior is permitted to allow for the case when the APB interconnect splits a single 64-bit register access and presents it to the FMU in any order.

4.5.7 Ping mechanism

This section describes the MMU FMU ping mechanism.

Background ping

The FMU provides a background ping mechanism to detect network issues between the TCU and the remote TBU blocks. Using the ping mechanism, the TCU sequentially sends a ping message over the AXI4-Stream network to a remote TBU block, one at a time. It then starts a timer and expects a PING_ACK message back from that remote block. If PING_ACK is not received within the expected interval from the intended remote TBU block, the FMU indicates a PING_ACK timeout error.

The ping mechanism can help identify the following issues:

- Permanent deadlock that permanent Stuck-At Faults on valid and ready bits cause.
- Congestion in the network that exceeds the FMU_PINGCTLR.ping_timeout_value.
- Systematic issues in the network that cause misrouting of messages.
- Connectivity issue of remote blocks to the TCU.

The background ping by MMU FMU can be enabled by writing to the *FMU_PINGCTLR, Ping Control Register* on page 4-126. FMU_PINGCTLR.ping_timeout_value defines the timeout in the MMU FMU clock.

(FMU_PINGCTLR.ping_interval_diff + FMU_PINGCTLR.ping_timeout_value) defines the interval at which the FMU pings the next remote block, `ping_interval`.

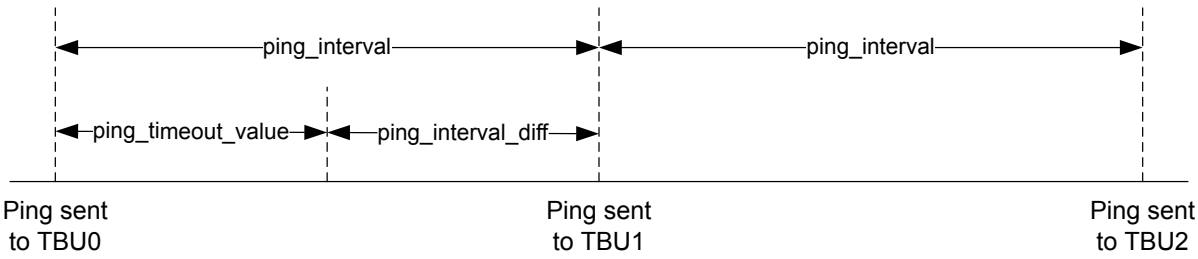

Figure 4-4 Ping mechanism parameters

Note

It is expected that the background ping using *FMU_PINGCTLR, Ping Control Register* on page 4-126 and directed ping using *FMU_PINGNOW, Ping Now Register* on page 4-127 are used mutually exclusively. When background pings are enabled, do not write FMU_PINGNOW.enable = 1.

Before generating directed pings using the FMU_PINGNOW register, turn off background ping by setting FMU_PINGCTLR.enable = 0 and waiting long enough for the last PING_ACK to be returned.

When the FMU indicates a PING_TIMEOUT error, you can obtain the remote TBU block ID by the reading the *FMU_ERR<n>STATUS, Error Record Primary Status Register* on page 4-123.

Note

To conserve operational power of the TCU, the TCU accepts the Q-Channel handshake to enter low powerdown state, if requested by the clock controller.

When the TCU is in the low-power clock gated state, it does not send background ping messages to the TBUs and does not report PING_ACK violations.

When the TCU is not in the low-power clock gated state, the FMU resumes background pings.

Directed ping

The software can also send directed ping messages to a specific block using the *FMU_PINGNOW, Ping Now Register* on page 4-127. This process can help debug PING_ACK violations that are received from background pings.

Use the following recommended software procedure to initiate a directed software ping:

1. To disable background pings, write FMU_PINGCTLR.enable = 0.
2. To clear all flags, write all zeros into the *FMU_PINGNOW, Ping Now Register* on page 4-127.
3. To initiate a directed ping, write:
   a. FMU_PINGNOW.enable = 1.
   b. FMU_PINGNOW.ping_ack_received = 0.
   c. FMU_PINGNOW.blk_id of the TBU block to which the ping is to be sent.
4. Poll for FMU_PINGNOW.ping_ack_received == 1.
5. Optionally, set Error Injection bits to test TBU or TCU integration, software, or both.

The PINGNOW feature can be used to send an erroneous packet from the TCU to a targeted TBU or from a targeted TBU to the TCU. Using this feature enables the TCU or TBU integrator to verify the AXI4-Stream and F-Channel connections between the TBUs and the TCU.

Injecting an error on a TCU ping message and on the subsequent TBU PING_ACK message causes mismatches along the PING/PING_ACK route through the interconnect. After injecting a PINGNOW error, you can read the TCU Error Records and verify that the expected SM errors are reported along the PING or PING_ACK route, for example by the receiving block and by any ADB components along the path.

When writing to the FMU_PINGNOW register and FMU_PINGNOW.enable is written as 1:

- A single ping is sent for each write to a present block.
- If another ping is sent before a previous PING_ACK has been received, then:
  - If sent to the same destination, then the first ping back sets FMU_PINGNOW.ping_ack_received.
  - If sent to a different destination, then the first PING_ACK is silently discarded if or when received because it does not match the programmed FMU_PINGNOW.blk_id.
- An attempt to send a ping to a *not-present* block does not launch a ping and FMU_PINGNOW.ping_ack_received does not go HIGH.

- If FMU_PINGNOW.tcu_inject_error == 1, an error is injected on the outgoing PING packet on the TCU->TBU interface. The receiving TBU block and the ADB, if present, detect the erroneous payload and report it as a fault.
- If FMU_PINGNOW.tbu_inject_error == 1, an error is injected on the outgoing PING_ACK packet by the TBU on the TBU->TCU interface. The receiving TCU block and the ADB, if present, detect the erroneous payload and report it as a fault.

4.5.8 Correctable Error enable

By default, the FMU reports all errors as Uncorrectable Errors.

The RAMs are protected using CRC code. Some RAMs that act as caches on a CRC error perform invalidation and refetch of that cache line. The following RAMs act as caches:

- TCU CCB Tag RAM.
- TCU CCB EntryTag RAM.
- TCU WCB Tag RAM.
- TCU WCB Entry RAM.
- TBU MTLB Tag RAM.
- TBU MTLB Entry RAM.

Because this error is always corrected, CRC errors from these RAMs can be considered to be reported as correctable.

To program the FMU to consider these RAM errors as correctable for reporting purposes, set FMU_ERR<n>CTLR.CE_EN = 1. These Correctable Errors then update FMU_ERR<n>STATUS.CE. Whatever the value of FMU_ERR<n>CTLR.CE_EN, the errors are always corrected for these RAMs.

See:
- *FMU_ERR<n>CTLR, Error Record Control Register* on page 4-122.
- *FMU_ERR<n>STATUS, Error Record Primary Status Register* on page 4-123.

4.5.9 Programmer's View

The MMU-600 memory map that is used to address the legacy MMU functional logic is unchanged on the MMU-600AE.

The MMU-600AE uses a separate and independent memory map for the FDC PV. This section describes theMMU-600AE FDC memory map and PV.

The following information applies to both the FMU and functional MMU-600 registers:
- The base address is not fixed, and can be different for any particular system implementation. The offset of each register from the base address is fixed.
- Do not attempt to access reserved or unused address locations. Attempting to access these locations can result in UNPREDICTABLE behavior.
- Unless otherwise stated in the accompanying text:
  - Do-Not-Modify UNDEFINED register bits.
  - Ignore UNDEFINED register bits on reads.
  - All register bits are reset to 0 by a system or Cold reset.
- Access type is described as follows:
  - *Read and Write* (RW).
  - *Read-Only* (RO).
  - *Write-Only* (WO).
  - *Read-As-Zero* (RAZ).
  - *Writes Ignored* (WI).
- Bit positions that are described as reserved are:
  - RAZ/WI in an RW register.
  - RAZ in a RO register.
  - WI in a WO register.

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

The FMU PROG APB4 interface is 32 bits wide. Strobed write is also not supported.

The FMU PROG APB4 port permits only Secure access to the FMU. This is performed by checking the **PPROT**[1] signal during an access. If the access fails the security check, then PSLVERR is returned.

Programmer's View registers

The MMU-600AE includes FMU registers for the Error Records, ping mechanism, and Safety Mechanisms.

The system attempt to record errors in MMU block index *n* from the *MMU block IDs* table in *Functional PV and FMU PV* on page 4-112 in error record *n*.

The following table shows the FMU PV registers.

Table 4-8 FMU PV registers

Offset	Access	Size	Register	Description
`0x000` + 64 × n	RO	64	FMU_ERR<n>FR	Error Record Feature Register per MMU block. See *FMU_ERR<n>FR, Error Record Feature Register* on page 4-122.
`0x008` + 64 × n	RW	64	FMU_ERR<n>CTLR	Error Record Control Register per MMU block. See *FMU_ERR<n>CTLR, Error Record Control Register* on page 4-122.
`0x010` + 64 × n	RW	64	FMU_ERR<n>STATUS	Error Record Primary Status Register per MMU block. See *FMU_ERR<n>STATUS, Error Record Primary Status Register* on page 4-123.
`0xE00`	RO	64	FMU_ERRGSR	Error Group Status Register. See *FMU_ERRGSR, Error Group Status Register* on page 4-125.
`0xEA0`	RW	32	FMU_KEY	FMU Key Register. See *FMU_KEY, FMU Key Register* on page 4-126.
`0xEA4`	RW	32	FMU_PINGCTLR	Error Ping Control Register. See *FMU_PINGCTLR, Ping Control Register* on page 4-126.
`0xEA8`	RW	32	FMU_PINGNOW	Error Ping Now Register. See *FMU_PINGNOW, Ping Now Register* on page 4-127.
`0xEB0`	WO	32	FMU_SMEN	Safety Mechanism Enable Register. See *FMU_SMEN, Safety Mechanism Enable Register* on page 4-128.
`0xEB4`	WO	32	FMU_SMINJERR	Safety Mechanism Inject Error Register. See *FMU_SMINJERR, Safety Mechanism Inject Error Register* on page 4-129.
`0xEC0`	RW	64	FMU_PINGMASK	FMU Ping Mask Register. See *FMU_PINGMASK, Ping Mask Register* on page 4-129.
`0xF00`	RO	32	FMU_STATUS	Fault Management Status Register. See *FMU_STATUS, FMU Status Register* on page 4-130.
`0xFC8`	RO	32	FMU_ERRIDR	Error Record ID Register. See *FMU_ERRIDR, Error Record ID Register* on page 4-130.

If the number of connected TBUs, configured by the `TCUCFG_FUSA_TBU_FAULT_WIRE_COUNT` parameter, is greater than or equal to 54, then MMU-600AE uses 64KB page sizes instead of the normal 4KB page sizes.

The following table shows the updated register offsets if the number of TBUs is greater than or equal to 54.

Table 4-9 Updated register offsets if the number of TBUs is greater than or equal to 54

Register	Fewer than 54 TBUs	54 or more TBUs
FMU_ERRGSR, Error Group Status Register on page 4-125	0xE00	0xE000
FMU_KEY, FMU Key Register on page 4-126	0xEA0	0xE0A0
FMU_PINGCTLR, Ping Control Register on page 4-126	0xEA4	0xE0A4
FMU_PINGNOW, Ping Now Register on page 4-127	0xEA8	0xE0A8
FMU_SMEN, Safety Mechanism Enable Register on page 4-128	0xEB0	0xE0B0
FMU_SMINJERR, Safety Mechanism Inject Error Register on page 4-129	0xEB4	0xE0B4
FMU_PINGMASK, Ping Mask Register on page 4-129	0xEC0	0xE0C0
FMU_STATUS, FMU Status Register on page 4-130	0xF00	0xE100
FMU_ERRIDR, Error Record ID Register on page 4-130	0xFC8	0xE1C8

FMU_ERR<n>FR, Error Record Feature Register

This register defines which of the common architecturally defined features are implemented and, of the implemented features, which are software programmable.

The following table shows the bit descriptions.

Table 4-10 Error Record Feature Register bit descriptions

Bits	Name	Reset value	Type	Function
[63:8]	-	All zeros	Res0	Reserved.
[7:6]	FI	2'b10	RO	Fault Handling Interrupt. Feature is controllable using ERR<n>CTLR.FI.
[5:4]	UI	2'b10	RO	Error Recovery Interrupt for Uncorrected Errors. Feature is controllable using ERR<n>CTLR.UI.
[3:2]	-	2'b00	Res0	Reserved.
[1:0]	ED	2'b10	RO	Error reporting and logging. Feature is controllable using ERR<n>CTLR.ED.

FMU_ERR<n>CTLR, Error Record Control Register

For this error record:

- FMU_ERR<n>CTLR.ED controls whether the record logs an error.
- FMU_ERR<n>CTLR.CE_EN controls whether correctable errors are reported as uncorrectable.
- FMU_ERR<n>CTLR.FI and FMU_ERR<n>CTLR.UI control the interrupts that are sent when the error record is reporting an error, as indicated by FMU_ERR<n>STATUS.V == 1. A change to these fields asserts or deasserts the level-sensitive interrupt as appropriate.

This register is only reset by the **fmu_aresetn** signal. See *Reset* on page 4-111 in *4.5.2 Error signaling from the FMU* on page 4-111.

The following table shows the bit descriptions.

Table 4-11 Error Record Control Register bit descriptions

Bits	Name	Reset value	Type	Function
[63:4]	-	All zeros	Res0	Reserved.
[3]	FI	`1'b0`	RW	*Fault Handling Interrupt* (FHI) enable. This controls whether an FHI is generated for all detected and logged (FMU_ERR<n>CTLR.ED == 1) errors that are reported through this error record. That is: • Correctable errors, whether reported as CEs or UEs. See *4.7.2 RAM fault reporting* on page 4-134. • Uncorrectable errors.
[2]	UI	`1'b0`	RW	*Error Recovery Interrupt* (ERI) enable. This controls whether an ERI is generated for all detected, logged (FMU_ERR<n>CTLR.ED == 1), RAS errors reported through this error record as UEs. That is: • Correctable errors that are reported as uncorrectable (FMU_ERR<n>CTLR.CE_EN == 0). See *4.7.2 RAM fault reporting* on page 4-134. • Uncorrectable errors. **Note** An error that is reported as a UE might generate both an ERI and an FHI.
[1]	CE_EN	`1'b0`	RW	Correctable Error enable: `0` Treats Correctable Errors as Uncorrectable Errors (default). `1` Treats Correctable Errors and Uncorrectable Errors differently, and reports them separately. See *4.5.8 Correctable Error enable* on page 4-120.
[0]	ED	`1'b1`	RW	Error reporting and logging enable.

FMU_ERR<n>STATUS, Error Record Primary Status Register

This register indicates information relating to the recorded errors.

Poll the FMU_STATUS register after a write to this register to ensure that the effect of the write is complete. FMU_STATUS.idle == 1 indicates that the effect of a write is complete. See *FMU_STATUS, FMU Status Register* on page 4-130.

Until the write takes effect, that is, FMU_STATUS.idle == 1 then:

- The corresponding bit of FMU_ERRGSR might still report as 1.
- Any interrupts caused by this record might still be asserted.
- Any new error that occurs is treated as a second error recording on top of this error and causes an overflow to be set. See *Prioritized ERR<n>STATUS registers* on page 4-125.
- Any read of this register might return the old value, or if a new error has been recorded, then the newly recorded value.

This register is only reset by the **fmu_aresetn** signal. See *Reset* on page 4-111 in *4.5.2 Error signaling from the FMU* on page 4-111. Do not write to an FMU_ERR<n>STATUS corresponding to a powered-off block. See *Power management* on page 4-117.

The following table shows the bit descriptions.

Table 4-12 Error Record Primary Status Register bit descriptions

Bits	Name	Reset value	Type	Function
[63:40]	-	All zeros	Res0	Reserved.
[39:32]	BLKID	All zeros	RO	If a PING_ACK timeout error occurs, this field indicates the block ID of the remote MMU block that caused the error. Only valid for Error Record 0. Only valid when FMU_ERR<n>STATUS.V == 1 and FMU_ERR<n>STATUS.IERR == `8'd12`. See *FMU_ERR<n>STATUS, Error Record Primary Status Register* on page 4-123. This field is not updated when a PING_ACK timeout error is reported as a result of a software error injection using the *FMU_SMINJERR, Safety Mechanism Inject Error Register* on page 4-129.
[31]	-	`1'b0`	RO	Reserved.
[30]	V	`1'b0`	RW	Status Register valid. This bit is set if the record represents that one or more errors have occurred. If this bit is clear, then all other fields except for FMU_ERR<n>STATUS.SERR are UNKNOWN. For write behavior, see the Note at the end of this section.
[29]	UE	`1'b0`	RW	Uncorrected Error. If FMU_ERR<n>STATUS.V == 1 and FMU_ERR<n>STATUS.UE == 1 then one or more uncorrected errors occurred. For write behavior, see the Note at the end of this section.
[28]	-	`1'b0`	Res0	Reserved.
[27]	OF	`1'b0`	RW	Record has overflowed. Valid if FMU_ERR<n>STATUS.V==1, else UNKNOWN. For write behavior, see the Note at the end of this section. See *Prioritized ERR<n>STATUS registers* on page 4-125.
[26]	-	All zeros	Res0	Reserved.
[25:24]	CE	`2'b00`	RW	Corrected Error bit: `2'b00` No errors. `2'b10` One or more errors were corrected. Valid if FMU_ERR<n>STATUS.V == 1, else UNKNOWN. For write behavior, see the Note at the end of this section.
[23:22]	-	`2'b0`	RO	Reserved.

Table 4-12 Error Record Primary Status Register bit descriptions (continued)

Bits	Name	Reset value	Type	Function
[21:20]	UET	`2'b11`	RO	Uncorrected Error type. These bits read as: `2'b11` When FMU_ERR<n>STATUS.V == 1 and FMU_ERR<n>STATUS.UE == 1. UNKNOWN Otherwise.
[19:16]	-	All zeros	RO	Reserved.
[15:8]	IERR	`8'd0`	RO	Safety Mechanism ID code. See *Table 4-7 Safety Mechanisms* on page 4-113 for Safety Mechanism ID encodings. Valid if FMU_ERR<n>STATUS.V == 1, else UNKNOWN.
[7:0]	SERR	`8'd0`	RO	Reads as zero if FMU_ERR<n>STATUS.V == 0. Otherwise, reads as 1.

Note

The V, UE, CE, and OF fields in this register are write-one-to-clear, but there is an interaction between the bits.

A write to this register is ignored, unless after the write, the V, UE, OF, and CE fields are all zero. That is, either the:

- Field was zero to begin with.
- Write-one-to-clear cleared the field to zero.

The CE field is a 2-bit field but only ever reports `0b00` or `0b10`.

The value that is written to CE[0] is IGNORED for all purposes.

All other named fields are read-only and any value that is written to them is IGNORED for all purposes.

Prioritized ERR<n>STATUS registers

When a CE (Correctable Error) is followed by a UE (Uncorrectable Error), the following occurs:

1. The status registers are updated to reflect the SM ID of the Uncorrectable Error.
2. The UE is set along with the CE bit.
3. OF (Overflow) is not set in this case. OF is set only when either:
   - UE is set and another Uncorrectable Error is received.
   - CE is set and another Correctable Error is received.

Note

The MMU-600AE has separate UE and CE pipelines to avoid head of line UE blocking by a CE.

FMU_ERRGSR, Error Group Status Register

This register shows the status of the MMU-600AE Error Records.

If an error record is being acknowledge by a write to *FMU_ERR<n>STATUS, Error Record Primary Status Register* on page 4-123, then while FMU_STATUS.idle == 1, the corresponding bit in this register might read as 1.

Bit n in this register corresponds to ERR<n>STATUS.V in error record n.

This register is only reset by the **fmu_aresetn** signal. See *Reset* on page 4-111 in *4.5.2 Error signaling from the FMU* on page 4-111.

The following table shows the bit descriptions.

Table 4-13 Error Group Status Register bit descriptions

Bits	Name	Function
[63:0]	S	Indicates the status of Error Record n: `0` The Error Record is not reporting any errors. `1` The Error Record is reporting one or more errors. Records corresponding to non-existent TBUs Read-As-Zero.

FMU_KEY, FMU Key Register

Writing the correct key to this register enables the next write to any other writable register to succeed. This register reads as 0 if the register file is locked. This register reads as `0xBE` if the register file is unlocked.

See *4.5.6 Lock and key mechanism* on page 4-117 for more information about the lock-key mechanism.

The following table shows the bit descriptions.

Table 4-14 FMU Key Register bit descriptions

Bits	Name	Reset value	Type	Function
[31:8]	-	All zeros	Res0	Reserved.
[7:0]	KEY	`8h'00`	RW	The required key to write to FMU registers.

FMU_PINGCTLR, Ping Control Register

This register configures the error ping timing interval.

Do not change FMU_PINGMASK while background ping is enabled, that is, FMU_PINGCTLR.enable == 1.

The following table shows the bit descriptions.

Table 4-15 Ping Control Register bit descriptions

Bits	Name	Reset value	Type	Function
[31:16]	ping_interval_diff	All zeros	RW	Equal to (`ping_interval` − `ping_timeout_value`) in SMMU clock cycles.
[15:4]	ping_timeout_value	All zeros	RW	Timeout threshold value for ping timeouts in SMMU clock cycles. The minimum supported value is 4.

Table 4-15 Ping Control Register bit descriptions (continued)

Bits	Name	Reset value	Type	Function
[3:1]	-	All zeros	Res0	Reserved.
[0]	enable	`1'b0`	RW	Enables the TCU ping engine. The TCU sends ping messages to each remote component, and expects a PING_ACK back within the specified timeout. If the PING_ACK is not received within the specified timeout, then the TCU records this situation as an error. The TCU sequentially moves to the next block and sends another ping message after `ping_interval`. If pings are enabled, then the setting in the *FMU_PINGMASK, Ping Mask Register* on page 4-129 must unmask at least one TBU block.

FMU_PINGNOW, Ping Now Register

This register specifies the remote SMMU block to send the ping request to, and monitors whether that block has acknowledged the ping.

For more information on the use of this register, see *Directed ping* on page 4-119 in *4.5.7 Ping mechanism* on page 4-118.

Poll the FMU_STATUS register after a write to this register to ensure that the effect of the write is complete. FMU_STATUS.idle == 1 indicates that the effect of a write is complete. See *FMU_STATUS, FMU Status Register* on page 4-130.

Do not send a PING to a powered-off block. See *Power management* on page 4-117.

The following table shows the bit descriptions.

Table 4-16 Ping Now Register bit descriptions

Bits	Name	Reset value	Type	Function
[31:12]	-	All zeros	Res0	Reserved.
[11]	tbu_inject_error	`1'b0`	RW	`1` Inject an error on the PING_ACK response packet that is sent from the remote SMMU block to the FMU. This action causes errors along the route of the PING_ACK through the interconnect. The presence of errors helps to confirm that the interconnect path from the specified remote SMMU block to the FMU has been properly connected.
[10]	tcu_inject_error	`1'b0`	RW	`1` Inject an error on the PING data packet that is sent from the FMU to the remote SMMU block. This action causes errors along the route of the PING route through the interconnect. The presence of errors helps to confirm that the interconnect path from the FMU to the specified remote SMMU block has been properly connected.
[9]	ping_ack_received	`1'b0`	RW	Indicates whether a PING_ACK has been received: `0` PING_ACK has not been received. `1` PING_ACK has been received from the SMMU block that was pinged.

Table 4-16 Ping Now Register bit descriptions (continued)

Bits	Name	Reset value	Type	Function
[8]	enable	`1'b0`	RW	Ping enable: `0` Does not initiate a ping. Allows software to clear the status of this register without initiating another ping. `1` Initiates a ping to the SMMU block specified in FMU_PINGNOW.block_id.
[7:0]	block_id	`8'd0`	RW	Block identifier. Sends a ping request to the specified SMMU block. See *Table 4-6 MMU block IDs* on page 4-112 for block ID encodings.

FMU_SMEN, Safety Mechanism Enable Register

This register enables or disables particular SMs inside a specified SMMU block. All SMs in the SMMU blocks are enabled at reset.

Poll the FMU_STATUS register after a write to this register to ensure that the effect of the write is complete. FMU_STATUS.idle == 1 indicates that the effect of a write is complete. See *FMU_STATUS, FMU Status Register* on page 4-130.

Do not attempt to enable or disable a powered-off block. See *Power management* on page 4-117.

Note

If a block is powered-off and then powered-on again, the enabled state of the Safety Mechanism returns to the default reset state. See *Enabling and disabling a Safety Mechanism* on page 4-114 in *4.5.4 Safety Mechanism table* on page 4-112.

The following table shows the bit descriptions.

Table 4-17 Safety Mechanism Enable Register bit descriptions

Bits	Name	Reset value	Type	Function
[31:24]	SMID	X	WO	Safety Mechanism identifier. See *Table 4-7 Safety Mechanisms* on page 4-113 for Safety Mechanism ID encodings.
[23:16]	-	`0x8'd0`	Res0	Reserved.
[15:8]	BLK	X	WO	Block identifier. See Error Records.
[7:1]	-	`7'd0`	Res0	Reserved.
[0]	EN	X	WO	Safety Mechanism enable.

Note

This feature cannot be used for the following:

- BLK = TCU, SMID = 0.
- BLK = Fault channel block.
- BLK = TBU, SMID = 0.

FMU_SMINJERR, Safety Mechanism Inject Error Register

This register injects one error into the specified SM inside an SMMU block. Only one error is injected, and no explicit clearing of this mechanism is required.

Poll the FMU_STATUS register after a write to this register to ensure that the effect of the write is complete. FMU_STATUS.idle == 1 indicates that the effect of a write is complete. See *FMU_STATUS, FMU Status Register* on page 4-130.

Do not attempt to inject an error into a powered-off block. See *Power management* on page 4-117.

For more information, see:

- *4.5.4 Safety Mechanism table* on page 4-112.
- *Injecting an error into a Safety Mechanism* on page 4-115.

The following table shows the bit descriptions.

Table 4-18 Safety Mechanism Inject Error Register bit descriptions

Bits	Name	Reset value	Type	Function
[31:24]	SMID	X	WO	Safety Mechanism identifier. See *Table 4-7 Safety Mechanisms* on page 4-113 for Safety Mechanism ID encodings.
[23:16]	-	`8'd0`	Res0	Reserved.
[15:8]	BLK	X	WO	Block identifier. See Error Records.
[7:0]	-	`8'd0`	Res0	Reserved.

Note

This feature cannot be used for the following:

- BLK = TCU, SMID = 0.
- BLK = Fault channel block.
- BLK = TBU, SMID = 0.

FMU_PINGMASK, Ping Mask Register

This register configures the ping mask.

Do not change FMU_PINGMASK while background ping is enabled, that is, FMU_PINGCTLR.enable == 1.

It is not necessary to mask off a powered-off block in FMU_PINGMASK before powering it off. The SMMU automatically stops sending background pings to a powered-off block as the block performs the powerdown handshake.

The following table shows the bit descriptions.

Table 4-19 Ping Mask Register bit descriptions

Bits	Name	Reset value	Type	Function
[63:0]	ping_mask	All zeros	RW	Ping mask. Bit position corresponds to MMU block ID. To make the FMU skip specific TBU blocks while generating background ping messages, write a 1 to the corresponding bit: **Bit [0]** TCU **Bit [1]** Fault Channel **Bit [2]** TBU0 **Bit [2+n]** TBUn For unpopulated MMU blocks, the corresponding bits have no effect. The same applies to bit[0] and bit[1] because the FMU does not ping the TCU or the Fault Channel components.

FMU_STATUS, FMU Status Register

This register monitors whether the FMU is idle.

The following table shows the bit descriptions.

Table 4-20 FMU Status Register bit descriptions

Bits	Name	Reset value	Type	Function
[31:1]	-	All zeros	Res0	Reserved.
[0]	idle	`1'b1`	RO	Indicates whether the FMU is idle: `0` FMU is busy processing the previous command. `1` FMU is idle.

FMU_ERRIDR, Error Record ID Register

This register defines the highest numbered index of the Error Records in this group.

The following table shows the bit descriptions.

Table 4-21 Error Record ID Register bit descriptions

Bits	Name	Reset value	Type	Function
[31:16]	-	All zeros	Res0	Reserved.
[15:0]	NUM	2 + number of connected TBUs	RO	1 + highest numbered index of the Error Records in this group.

4.6 Lockstep protection

The MMU-600AE logic is protected by redundant lockstep checking.

The exceptions to this are:

- The RAMs, which are shared.
- The internal AXI4-Stream interconnect, which uses full duplication.

The following figure shows the lockstep for the TCU.

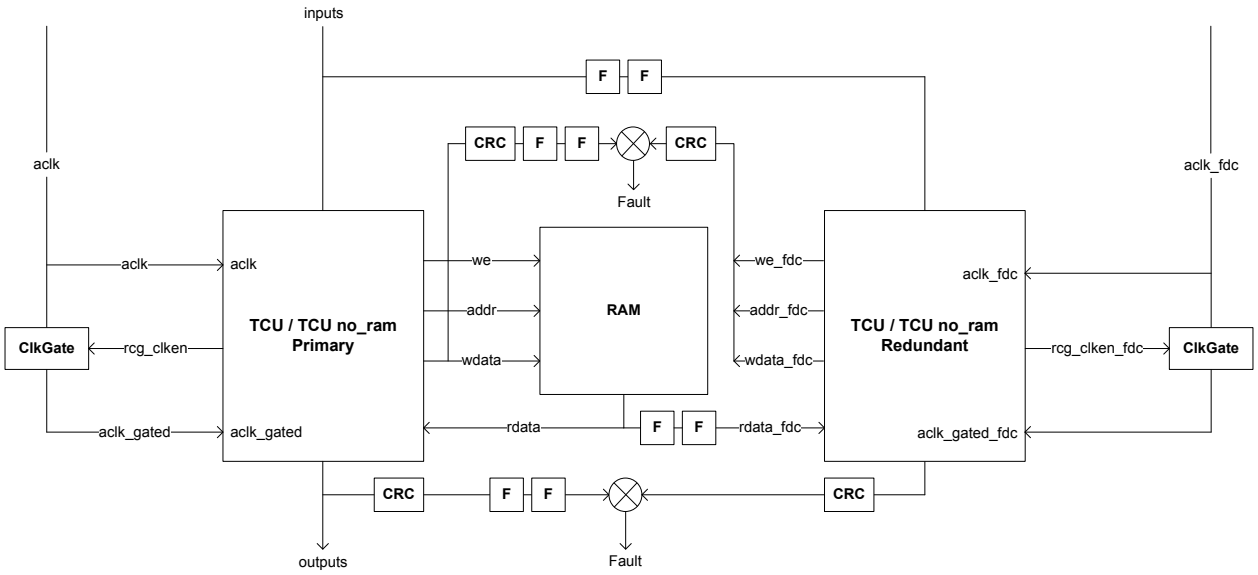

Figure 4-5 TCU lockstep

The lockstep has a standard Temporal Delay of two cycles, with RAM sharing and comparators. The comparators are shown by a circle with an X in the middle. To save power, CRC is used to compress the outputs.

The entire `noram` hierarchy is duplicated, with the comparators instanced in the block top level. The clock gate and reset synchronizers must also be duplicated in the top level.

The clocking is also duplicated. To provide redundancy in the reset and clock trees, the master (primary) and checker (shadow) logic are clocked by a separate clock and separate reset. If a branch of the reset of clock tree fails in the master domain, it is detected by the checker domain, and vice versa.

This section contains the following subsections:

- *4.6.1 Comparators* on page 4-131.
- *4.6.2 Non-resettable flops* on page 4-132.
- *4.6.3 Reset* on page 4-132.
- *4.6.4 Error injection* on page 4-132.

4.6.1 Comparators

The lockstep comparators consist of an XOR tree. The same parameterized comparator component is instanced throughout the design to promote uniformity and allow the implementation to be changed.

The comparators are known to be power hungry. Therefore, qualification is used wherever possible so they only check the outputs when necessary. For example, an AXI bus comparator checks the data only when the valid bits are asserted. This methodology is necessary to:

- Prevent flagging on benign glitches when nothing is reading the bus.
- Prevent false error from being asserted due to UNKNOWN values on the bus, from RAMs or from uninitialized datapath flops.

Comparator duplication option

The comparators can be duplicated by setting a parameter to aid in latent fault diagnostic coverage goals.

Duplicating the comparators provides passive latent fault coverage, preventing the need to achieve coverage through LBIST or software STL library means. The main trade-off is power and area, but partners should check timing results as well. The option adds one additional gate into the comparator paths.

To duplicate the comparators, set `FUSA_COMP_DUP`=1 in instantiation.

Note

All comparators in MMU-600AE can be duplicated, including lockstep and CRC comparators.

4.6.2 Non-resettable flops

Non-resettable flops from MMU-600 were made resettable in cases where false output comparator errors can occur due to non-deterministic states of those flops.

4.6.3 Reset

Logic has been added to the MMU-600AE to guarantee a proper reset for lockstep logic.

For more information on reset assumptions and requirements related to lockstep logic and FuSa, see the *FuSa clocks and resets* section in the *Arm® CoreLink™ MMU-600AE System Memory Management Unit Configuration and Integration Manual*.

4.6.4 Error injection

The FMU can be used to inject a fault into a fixed input of the lockstep comparators.

The main purpose is to test connectivity and software. This is not meant to be an exhaustive test of the comparator XOR tree. For this purpose, the comparators can be duplicated by setting the `FUSA_COMP_DUP` parameter, as described in *Comparator duplication option* on page 4-132.

4.7 RAM protection

For RAM errors that can be corrected, independent of the FMU_ERR<n>STATUS.CE_EN bit, the legacy MMU-600 PV reports some extra information about the RAM fault.

See *3.8.3 TCU_ERRSTATUS* on page 3-86 and *3.12.3 TBU_ERRSTATUS* on page 3-95.

For correctable and uncorrectable RAM faults, the FMU PV is limited to reporting the RAM in which the fault occurred.

Errors are corrected by invalidating the cache lines that are associated with the error.

The address is not protected on the MMU-600. The MMU-600AE adds 8-bit CRC protection on both RAM data and RAM address bits. The CRC code is created from the combination of the data and the address being written to. When a memory location is read, the hardware calculates the CRC code, based on the data that is read from the RAM and the address that is used to read the data. If this calculated CRC code does not match the CRC code that was stored with the data, a CRC fault is flagged and correction is initiated, if applicable.

The CRC protection scheme can detect the following errors:

- All *Single-Bit data Errors* (SBEs).
- All *Double-Bit data Errors* (DBEs) on data widths of 128 bits or less.
- 99.6% of all DBEs on data widths greater than 128 bits.
- 99.6% of all *Multi-Bit data Errors* (MBEs).
- 99.6% of incorrect address errors.

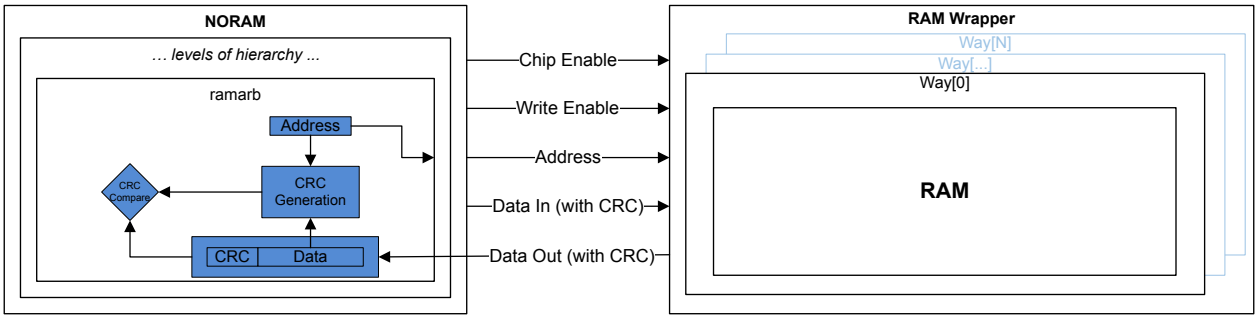

Figure 4-6 RAM CRC read path

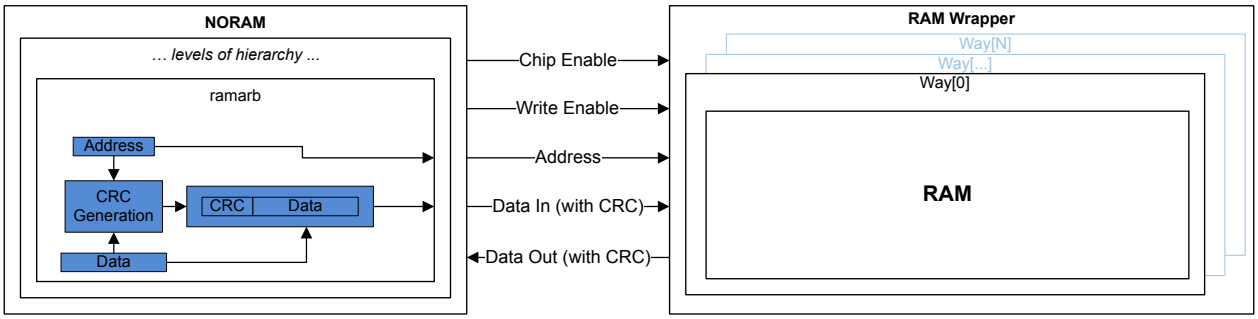

Figure 4-7 RAM CRC write path

This section contains the following subsections:

- *4.7.1 RAM fault correction* on page 4-134.
- *4.7.2 RAM fault reporting* on page 4-134.
- *4.7.3 RAM fault control* on page 4-134.

- *4.7.4 RAM fault severity* on page 4-134.
- *4.7.5 Address protection* on page 4-134.

4.7.1 RAM fault correction

On the MMU-600, faulty Tag and Entry RAM caches are invalidated based on a parity error. On the MMU-600AE, these faulty caches are invalidated based on a CRC or parity error.

If the faulty RAM is acting as a buffer and not a cache, TCU Translation Request Buffer and TBU Write Buffer, the fault is not corrected. The uncorrected faulty data is used and can lead to some of the unpredictable effects that *4.5.4 Safety Mechanism table* on page 4-112 describes.

4.7.2 RAM fault reporting

For the correctable RAMs, CRC or parity faults are always corrected by invalidating the cache entry and refetching:

- TCU CCB and WCB Tag and Data RAMs, TCU SMID: 7-10.
- TBU MTLB Tag and Data RAMs, TBU SMID: 7-8.

An attempt is made to report a CRC or a parity fault as a correctable error in the MMU-600 PV RAS registers. The PV RAS registers supply extra syndrome information for these errors in TCU_ERRSTATUS.IERR and TBU_ERRSTATUS.IERR. See *3.8.3 TCU_ERRSTATUS* on page 3-86 and *3.12.3 TBU_ERRSTATUS* on page 3-95.

An attempt is made to report a CRC failure as a correctable or uncorrectable error in the FMU error records. See *4.5.8 Correctable Error enable* on page 4-120.

Note

Parity failure, but not CRC failure, might detect rare *Multi-Bit Errors* (MBEs). In which case, the error is only reported in the MMU-600 PV RAS registers. However, the CRC catches 99.6% of all MBEs and therefore this situation is extremely rare.

Errors in all other RAMs are uncorrectable, and an attempt is made only to report them in the FMU error records. No extra syndrome information is available other than the SMID that identifies the RAM.

The MMU-600 uses faulty data from uncorrectable RAMs and this can lead to some of the unpredictable effects that *4.5.4 Safety Mechanism table* on page 4-112 describes.

For both correctable and uncorrectable faults, both real faults and faults that software injects differ slightly. See *Injecting an error into a Safety Mechanism* on page 4-115.

4.7.3 RAM fault control

RAM CRC fault reporting can be enabled or disabled using the FMU_SMEN register.

This register only affects reporting of a CRC error. It does not affect the reporting of the legacy parity error in the MMU-600 PV.

4.7.4 RAM fault severity

You can select whether a RAM fault is reported to the Safety Island through either an ERI or FHI interrupt.

See *4.5 Fault Management Unit* on page 4-109 for more information.

4.7.5 Address protection

Because the RAM is shared, address protection must consider the protection of address decoders within the RAM decoder macro itself. Otherwise, faults within the RAM macro address decoder can cause *Common Mode failure* (CMF).

The CRC code is created from the combination of the data and the address that is being written to.

When a memory location is read, the hardware calculates the CRC code based on the data that is read from the RAM and the address that is used to read the data. If this calculated CRC code does not match the CRC code that was stored with the data, a CRC fault is flagged.

The RAM type determines whether the fault is corrected.

- If the faulty RAM is a cache, correction is initiated by invalidating the failed cache location.
- If the faulty RAM is a buffer, for example the TCU Translation Request Buffer or TBU Write Buffer, the fault is not corrected.

4.8 External interface protection

All external bus interfaces are protected as defined by the AMBA Parity Extensions for *point-to-point* protection. These interfaces include the ACE-Lite, AXI4-Stream, and APB external interfaces.

The following figure shows the distribution of interface protection within the MMU-600AE. The ACE-Lite, AXI4-Stream, and APB external interfaces are shown as bidirectional orange arrows.

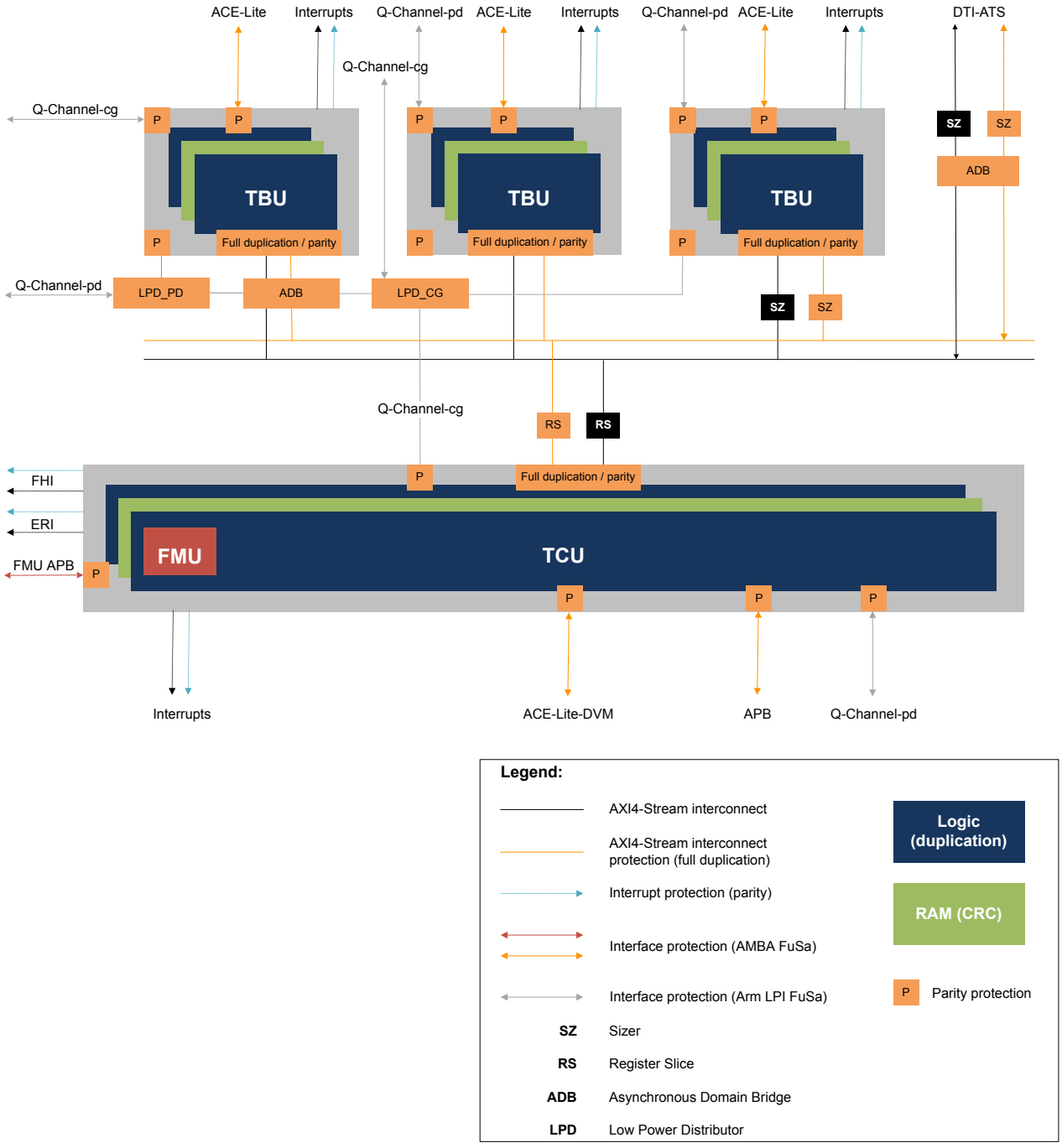

Figure 4-8 Interface protection distribution

Note

The ADB and Register Slice do not support parity on the AXI4-Stream interface. Therefore, only full duplication on AXI4-Stream is supported in the example topology.

Point-to-point protection

Point-to-point protection is sufficient for wires and buffers that cannot cause multiple-bit faults. An example of an interconnect component that could cause multiple-bit faults is a switch. A single fault on a switch mux input can switch the wrong data, causing multiple bits to fail.

This section contains the following subsections:

4.8.1 ACE-Lite interface parity protection

MMU-600AE supports ACE-Lite interface parity protection for point-to-point connections from MMU-600AE to another functionally safe IP or FuSa interconnect. If a parity fault is detected, and the safety mechanism is enabled, the FMU attempts to report an error in the FMU error records, potentially issuing an FHI and/or an ERI interrupt. The FMU attempts to report an error in a TBU or TCU error record. The MMU uses the faulty data.

Note

If this protection is not required, it can be disabled through the MMU-600AE FMU PV.

Assumptions of use for FuSa purposes

Arm expects that:

- MMU-600AE is directly connected to the far-end IP with only wires and repeater buffers.
- No complex logic gates, such as ADBs or cross bar switches, exist in the path because they could be a source of *Multiple Bit Errors* (MBEs).
- The far-end IP checks the parity bits that MMU-600AE generates.
- The far-end IP generates the incoming parity bits, as *ACE-Lite interface parity protection* in the *Arm® CoreLink™ MMU-600AE System Memory Management Unit Configuration and Integration Manual* describes.

4.8.2 AXI4-Stream interface parity protection

MMU-600AE AXI4-Stream interfaces support AMBA Parity Extension protection for point-to-point connections from MMU-600AE to another FuSa IP or FuSa interconnect. If a parity fault is detected, and the safety mechanism is enabled, the FMU attempts to report an error in the FMU error records, potentially issuing an FHI and/or an ERI interrupt. The FMU attempts to report an error in a TBU or TCU error record. The MMU uses the faulty data.

Note

If this protection is not required, it can be disabled through the MMU-600AE FMU PV.

Assumptions of use for FuSa purposes

Arm expects that:

- MMU-600AE is directly connected to the far-end IP with only wires and repeater buffers.
- No complex logic gates, such as ADBs or cross bar switches, exist in the path because they could be a source of MBEs.
- The far-end IP checks the parity bits that MMU-600AE generates.
- Far-end IP generates the incoming parity bits, as *AXI4-Stream interface parity protection* in the *Arm® CoreLink™ MMU-600AE System Memory Management Unit Configuration and Integration Manual* describes.

4.8.3 APB interface parity protection

The MMU-600AE has two APB interfaces, one for the MMU-600 PV register file, and the other to the FMU register file. The MMU-600AE supports APB interface parity protection on both APB point-to-point connections from MMU-600AE to another FuSa IP or FuSa interconnect. If a parity fault is detected, and the safety mechanism is enabled, the FMU attempts to report an error in the FMU error records, potentially issuing an FHI and/or an ERI interrupt. The FMU attempts to report an error in a TBU or TCU error record. The MMU uses the faulty data.

Note

If this protection is not required, it can be disabled through the MMU-600AE FMU PV. Disable this protection when using an interconnect that does not generate AMBA parity.

Assumptions of use for FuSa purposes

Arm expects that:

- MMU-600AE is directly connected to the far-end IP with only wires and repeater buffers.
- No complex logic gates, such as ADBs or cross bar switches, exist in the path because they could be a source of MBEs.
- The far-end IP checks the parity bits that MMU-600AE generates.
- Far-end IP generates the incoming parity bits, as *APB interface parity protection* in the *Arm® CoreLink™ MMU-600AE System Memory Management Unit Configuration and Integration Manual* describes.

4.8.4 F-Channel

The *Fault Channel* (F-Channel) communicates faults from FuSa blocks and components to the FMU using a Fault Channel interface.

This interface consists of a REQ/ACK four-phase handshake protocol, similar to that of the Q-Channel interface, that is designed to work across asynchronous domains. The phases of this protocol are:

1. The **freq** signal originating from the APD or LPD (in the case of the MMU-600AE) is asserted.
2. The **freq** signal is held until the **fack** response is received from the FMU.
3. When the **fack** response is received, the FuSa block deasserts its **freq** signal.
4. When the **freq** signal is deasserted by the FuSa block, the FMU deasserts its **fack** signal.

Table 4-22 F-Channel signals

Signal	Origin	Destination	Description
freq	ADB/LPD	FMU	Request to report a fault to the FMU.
fack	FMU	ADB/LPD	Acknowledgment that the FMU received the fault.
freq_chk	ADB/LPD	FMU	Asynchronous redundancy.
fack_chk	FMU	ADB/LPD	Asynchronous redundancy.

The following figure shows the F-Channel and FMU fault wire connections.

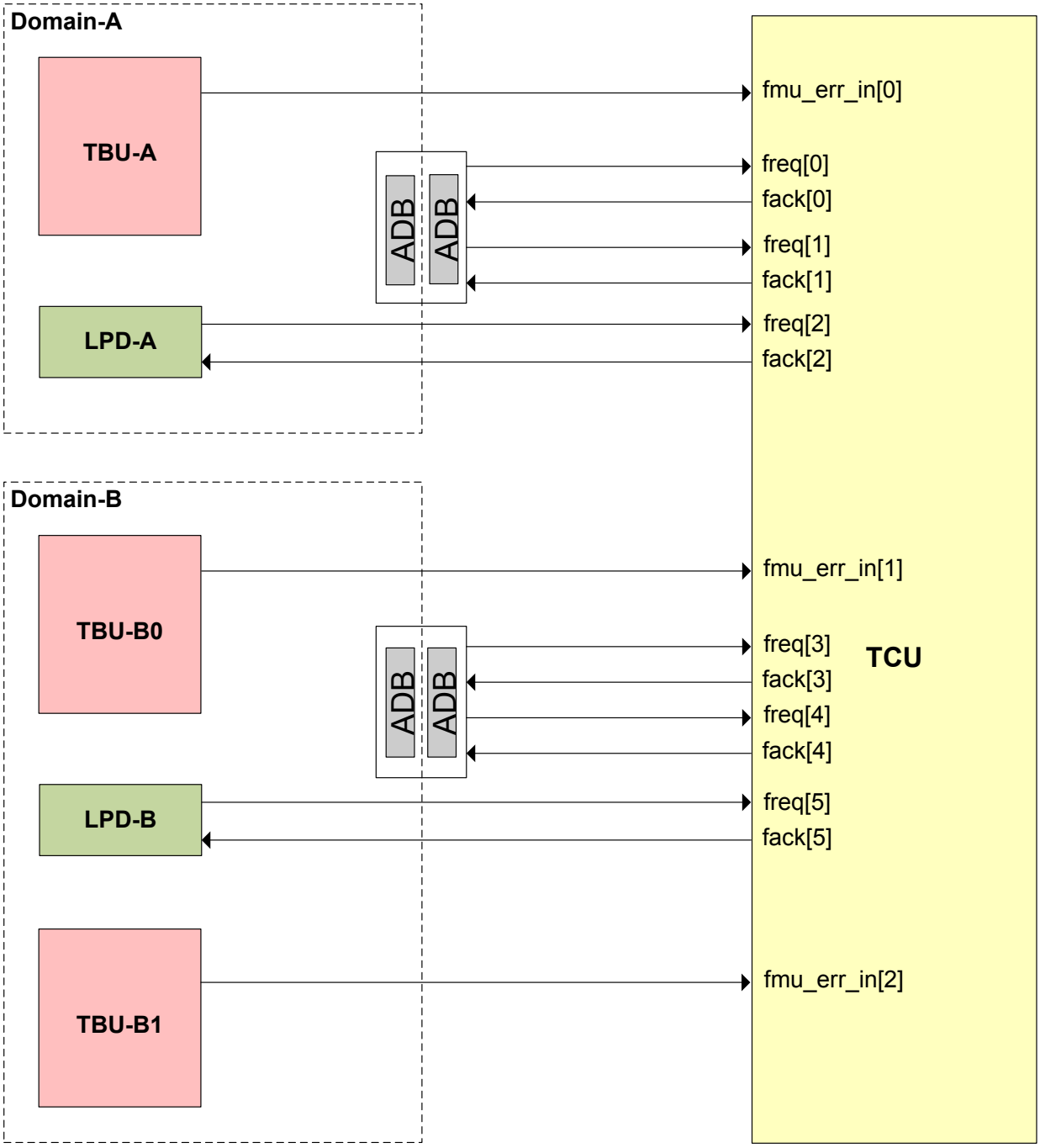

Figure 4-9 F-Channel and FMU fault wire connections

The related parameter settings for these connections are:

- `TCUCFG_FUSA_FCHAN_COUNT`=6.
- `TCUCFG_FUSA_TBU_FAULT_WIRE_COUNT`=3.

Note

For clarity, the **freq_chk** and **fack_chk** connections are not shown.

4.8.5 Interrupt output protection

Each single interrupt output is protected with an inverted **chk** parity bit. The **chk** bit is launched on the same clock cycle as the interrupt bit that it is protecting.

This format is compatible with the Arm CoreLink GIC-600AE Generic Interrupt Controller interrupt input protection. If you do not require this protection, you can leave the output unconnected.

4.8.6 Tie-off input protection

The TBU and TCU employ “tie-off” or strap bits, which affect the out-of-reset behavior of the MMU.

These bits are protected by inverted duplication, which means that **port_chk**[x:0] = ~**port**[x:0]. These tie-off inputs are expected to be static during and after reset. These inputs can be:

- Tied off.
- Connected to fuses or straps.
- Driven by other logic as required by the SoC.

Example 4-1 TCU tie-off input

For the TCU tie-off input **ecorevnum**[3:0], the **chk** bits are **ecorevnum_chk**[3:0].
If **ecorevnum**[3:0] == `0b0010`, **ecorevnum_chk**[3:0] must be `0b1101`.

———— **Note** ————

For multi-bit ports, this protection is different from parity.

————————————

4.8.7 Interfacing to unsafe interfaces

If a TBU or TCU is connected to an interface that is not safe, that is, the interface does not provide the ***chk*** or ***fdc*** signals, then you must comply with some guidelines.

The guidelines that you must comply with for unsafe interfaces are as follows:

- No input signal must be floating, that is, X or Z
- Tie unused ***validchk*** and ***wakeupchk*** input signals to `1`
- Tie all other unused ***chk*** input signals to `0`
- Tie all other unused ***fdc*** input signals to their original counterpart, for example, tie **aclk_fdc** to **aclk**
- Tie unused **Freq** and **fmu_err_in** to `0`
- Set the `TCUCFG_FUSA_FCHAN_COUNT` and `TCUCFG_FUSA_TBU_FAULT_WIRE_COUNT` parameter to `1`, if you are not using F-Channel error reporting
- Set the `TBUCFG_FUSA_DTI_FULL_DUP_PROT` and `TCUCFG_FUSA_DTI_FULL_DUP_PROT` parameters to `0` for point-to-point protection if the DTI interface is unsafe
- Set the `TCUCFG_FUSA_DISABLE_PQCHAN_PROT` parameter to `1`, if you do not have redundant P-Channel or Q-Channel protection
- Set the `TCUCFG_FUSA_DISABLE_SYSCO_PROT` parameter to `1`, if you do not have redundant SYSCO interface protection, ACE interface

4.9 Integrating the TCU, TBU, LPD, PCIe ATS, and DTI AXI4-Stream interconnect

The MMU-600AE supports the following options for protecting the DTI AXI4-Stream interfaces on the TCU and TBU:

Duplicated DTI AXI4-Stream interfaces

The TCU and TBU have duplicated DTI AXI4-Stream ports which can be used to protect the interconnect by duplication. Like the MMU-600, the MMU-600AE provides AXI4-Stream components that the SoC integrator can use to build their own interconnect. A modified ADB that supports AXI4-Stream duplication is provided with the MMU-600AE, which is used when the duplicated interconnect crosses asynchronous boundaries.

Single DTI AXI4-Stream interface with protection

The single DTI AXI4-Stream interface is protected by AMBA parity point-to-point interface. The TCU and TBU blocks can be connected to an interconnect or other IP that supports AMBA Parity Extensions for protecting point-to-point connections. When in this mode, the MMU-600AE generates the parity for the interface outputs, checks the parity for interface inputs, and flags a fault if there is mismatch.

Note

If the DTI AXI4-Stream interconnect IP does not support protection, then the safety mechanism inside the TBU (SMID: 4) and TCU (SMID:2) should be disabled during system initialization to prevent errors from being reported for these interfaces.

When a TBU comes out of reset, it exchanges messages with the TCU. If the parity protection has not been disabled by that time, then it attempts to report an error:

- For the TBU, in FMU error record (tbu_index + 2) with UE = 1, IERR = SMID:4
- For the TCU, in FMU error record 0 with UE = 1, IERR = SMID:2.

However, because the TCU and TBU continue to use the messages (despite the parity error) then the TBU connects correctly. Software can then later disable the SM (TBU SMID:4, TCU SMID:2) and discard any of the parity errors that the connection messages generated.

For more information, see the *FuSa integration* section of the *Arm® CoreLink™ MMU-600AE System Memory Management Unit Configuration and Integration Manual*.

4.10 Q-Channel protection

This section describes the Q-Channel logic and connections.

The following figure shows a top-level example of an MMU topology with AXI4-Stream interconnect connections.

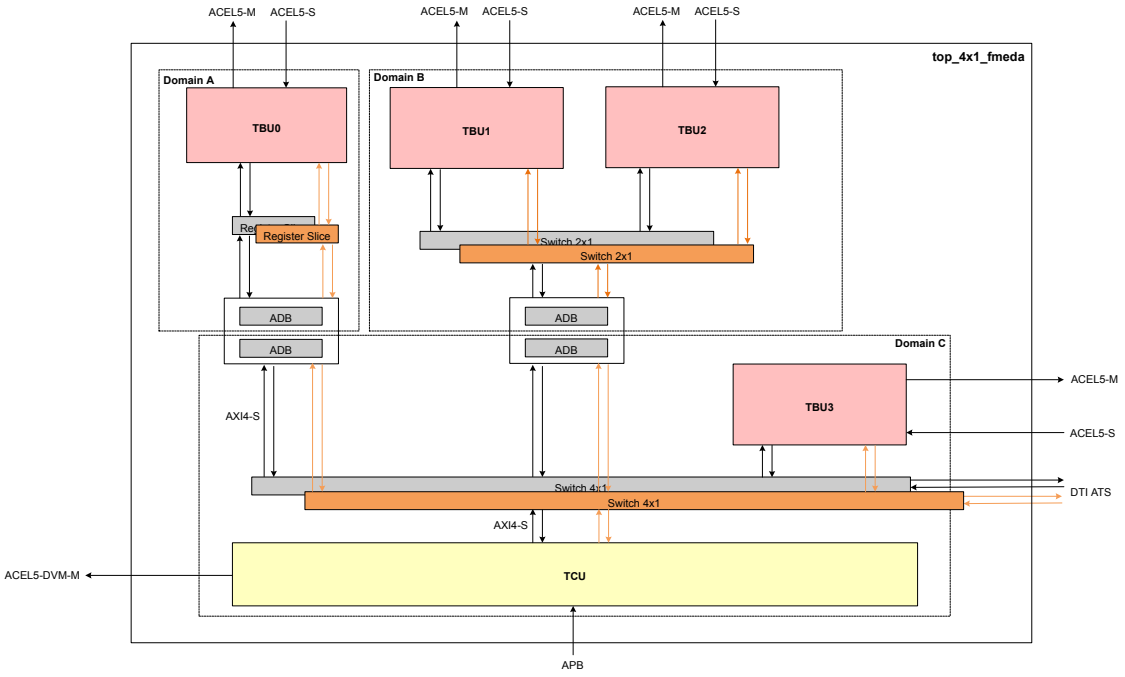

Figure 4-10 MMU topology with AXI4-Stream interconnect connections

The following figure shows a top-level example of an MMU topology with corresponding Q-Channel connections.

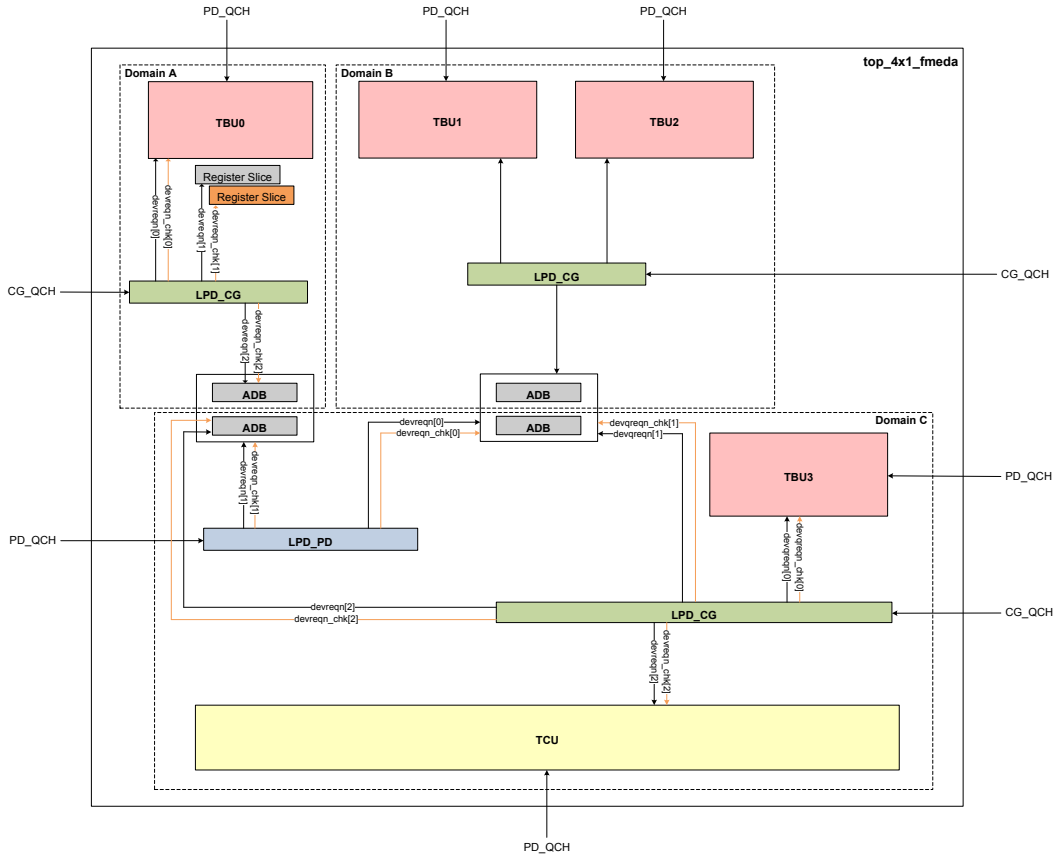

Figure 4-11 MMU topology with corresponding Q-Channel connections

The Q-Channels are protected with additional AMBA LPI specified redundant **chk** bits with reverse polarity. Due to the four-phase asynchronous nature of the Q-Channel, signals are checked individually. With four-phase handshaking, all assertions must be held until handshaking feedback is received. Therefore, transient assertions are treated as faults which are filtered for reliability by the MMU-600AE protection logic. The protection logic prevents these faults from reaching mission mode logic and causing errors. Permanent or *Stuck-At Faults* (SAF) are detected and flagged.

The following figure is a high-level block diagram of a Q-Channel example employed by the MMU blocks. The figure shows that **qreqn** and **qreqn_chk** are synchronized separately and then passed through redundant sig_prot blocks where the transient filtering and the stuck-at checker counters live. The Q-Channel outputs are passed to the external power controller (or internal MMU LPD) with a temporal delay of no more than two cycles. This variation is allowed by the Q-Channel AMBA LPI extensions.

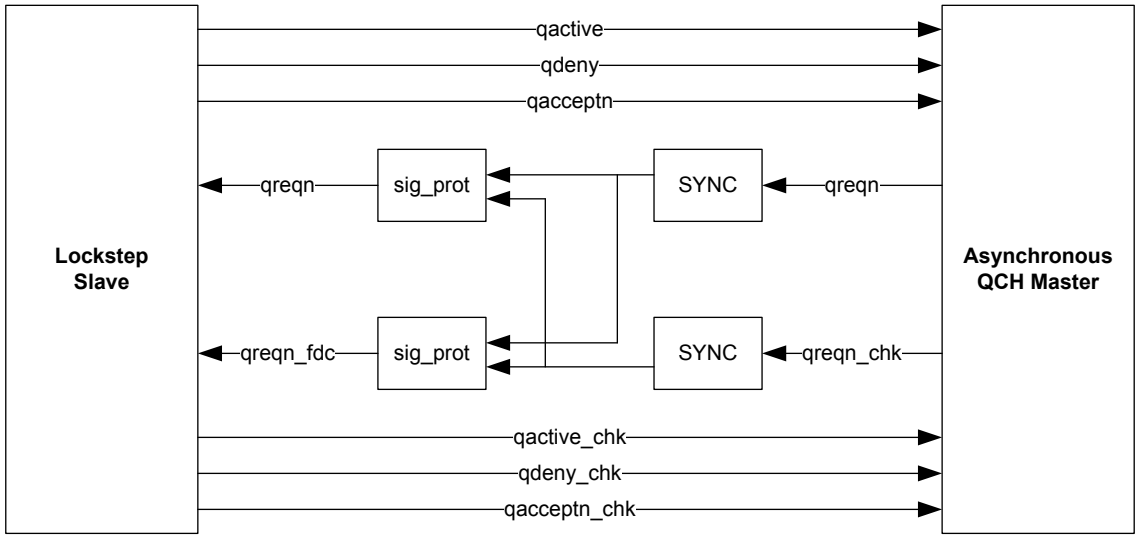

Figure 4-12 Q-Channel protection connections

This section contains the following subsections:

- *4.10.1 Q-Channel signaling* on page 4-144.
- *4.10.2 Q-Channel acceptance* on page 4-145.
- *4.10.3 Q-Channel denial* on page 4-145.
- *4.10.4 CHK bit timing* on page 4-145.
- *4.10.5 Transient faults* on page 4-146.
- *4.10.6 Stuck-at faults* on page 4-147.
- *4.10.7 Disabling Q-Channel Safety Mechanisms* on page 4-148.

4.10.1 Q-Channel signaling

The following figure shows the Q-Channel device and controller signal mappings, including the added **chk** signals.

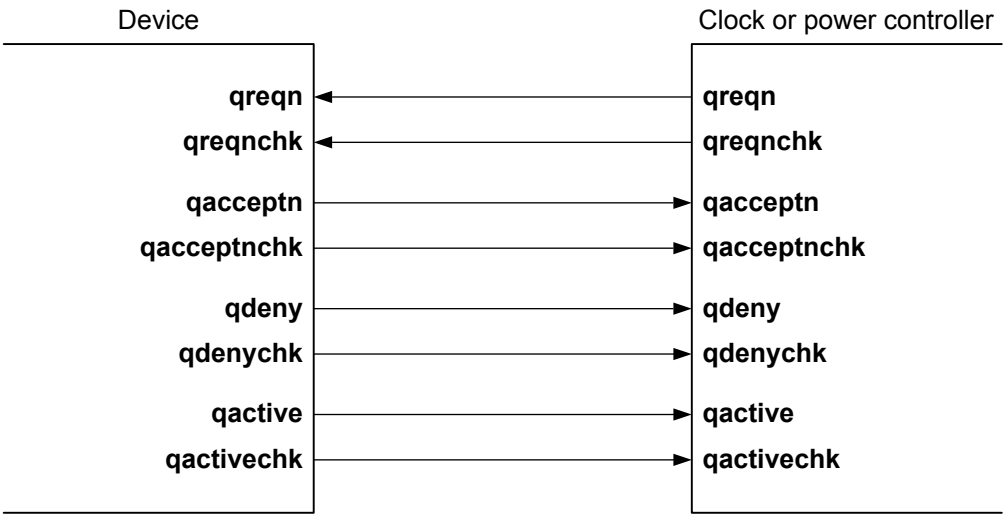

Figure 4-13 Q-Channel device and controller signal mappings

There is one new **chk** bit with inverted polarity for each Q-Channel signal.

4.10.2 Q-Channel acceptance

The following figure shows the opposite polarity of the **chk** bits during the Q-Channel entry, acceptance, and exit sequence.

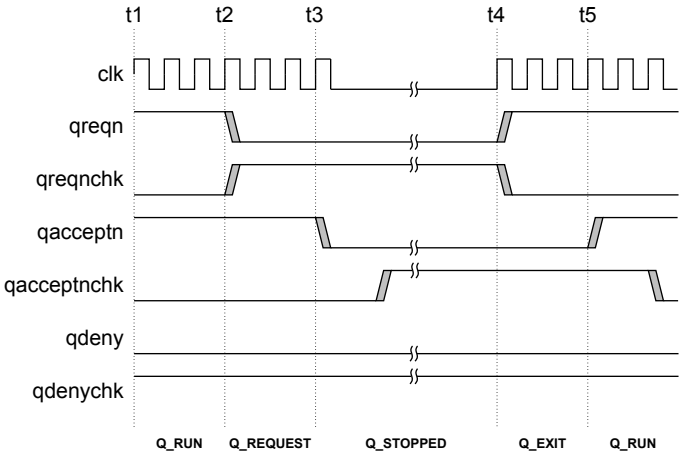

Figure 4-14 Q-Channel acceptance

4.10.3 Q-Channel denial

The following figure shows the opposite polarity of the **chk** bits during the Q-Channel denial sequence.

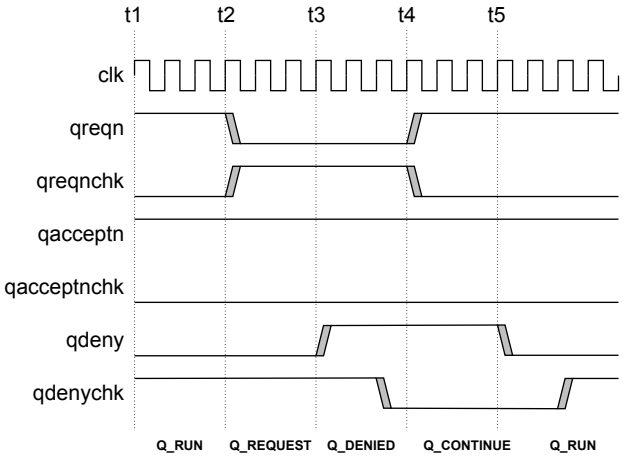

Figure 4-15 Q-Channel denial

4.10.4 CHK bit timing

There is a hard timing requirement that is determined by the *Stuck-At Fault* (SAF) detection logic.

The skew of **qreqn** and **qreqn_chk** must be less than the maximum skew that the SAF detection logic allows.

***Clock Ratio* (CR)**

Equal to (MMU clock frequency)/(channel controller clock frequency).

Implementation Skew
Silicon skew due to asynchronous clock domain crossings or other factors.

Temporal Delay Skew
Skew between lockstep primary and redundant logic blocks.

Since the MMU-600AE SAF detector counts to 64 before flagging an SAF, the permitted skew is calculated as follows:

Maximum skew allowed = 64/CR.

Example 4-2 Q-Channel skew calculation

- MMU clock frequency = 1000MHz.
- Q-Channel frequency = 125MHz.

Based on these frequencies, the CR is calculated as follows:

CR = (MMU clock frequency)/(channel controller clock frequency) = 1000MHz/125MHz = 8.

Maximum skew allowed = 64/CR = 64/8 = 8 cycles.

Therefore, the SoC integrator is allowed eight cycles for Implementation Skew and Temporal Delay Skew that originate from the SoC Q-Channel controller.

4.10.5 Transient faults

The following figure shows the normal situation with no fault.

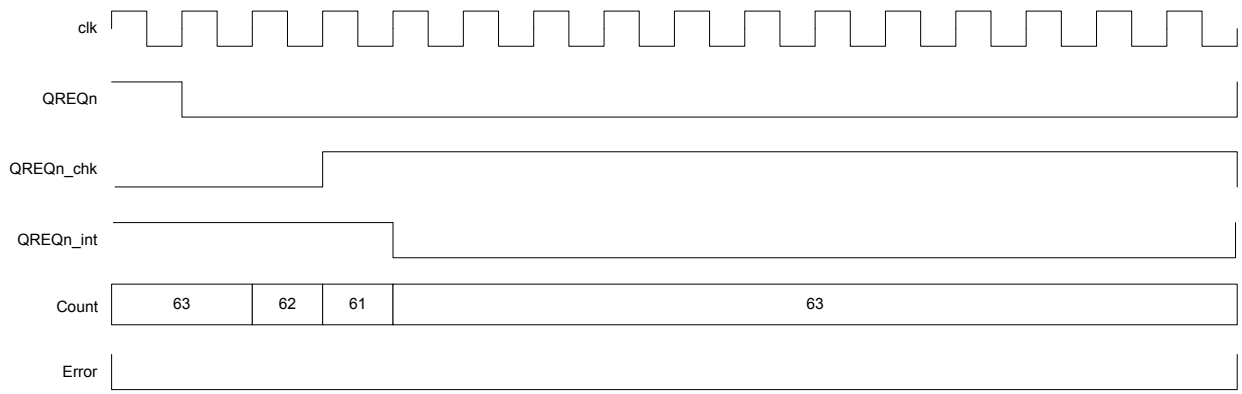

Figure 4-16 Normal assertion of qreqn and qreqn_chk

The following figure shows how a transient fault on **qreqn** is filtered.

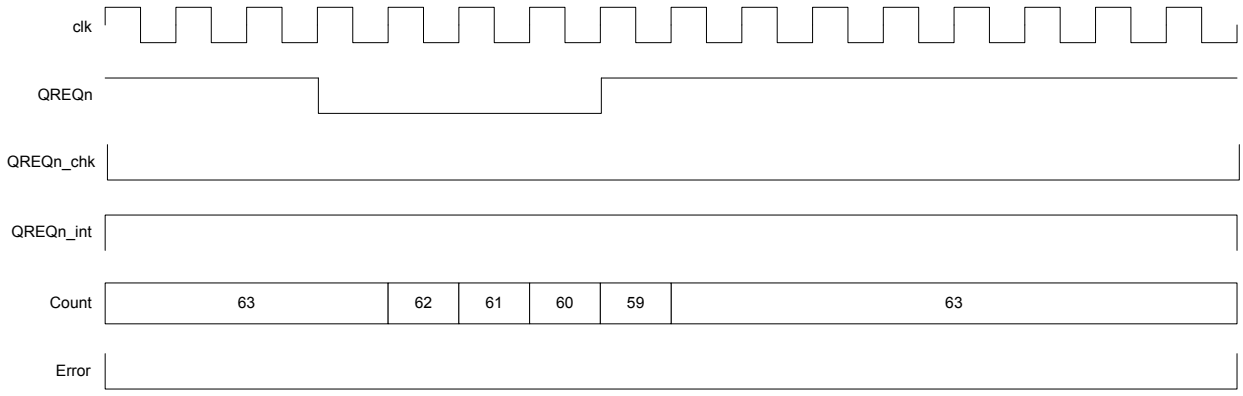

Figure 4-17 Transient fault on qreqn

The following figure shows how a transient fault on **qreqn_chk** is filtered.

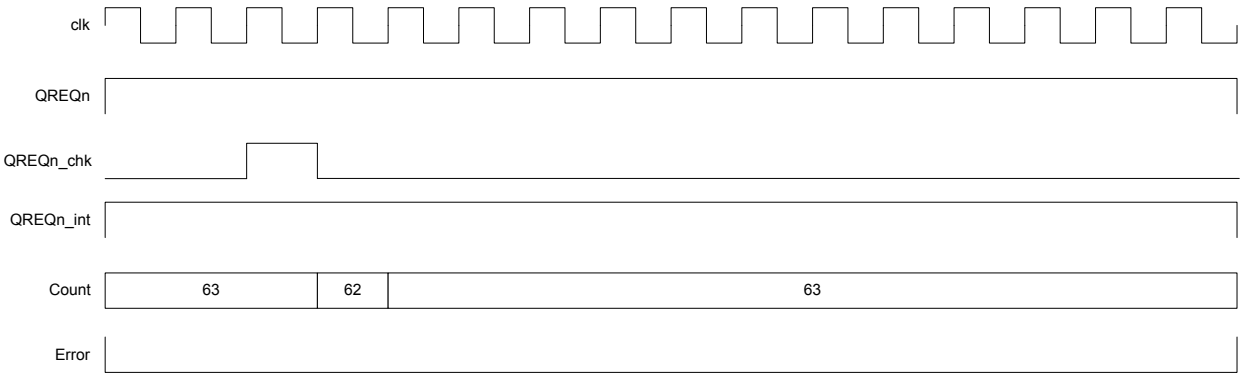

Figure 4-18 Transient fault on qreqn_chk

The output of the filtering logic, **qreqn_int**, does not assert. The figures depict a version of **qreqn** and **qreqn_chk** after they pass synchronizer cells. The counter depicts the operation of the SAF detector. In this example, the SAF detector is set to a value of 63 whenever **qreqn** and **qreqn_chk** are the same polarity. If it detects a polarity difference between **qreqn** and **qreqn_chk**, it starts counting down. If the counter reaches zero, it flags an error.

4.10.6 Stuck-at faults

The following figure shows how the SAF detector detects a stuck-at-one error on **qreqn**.

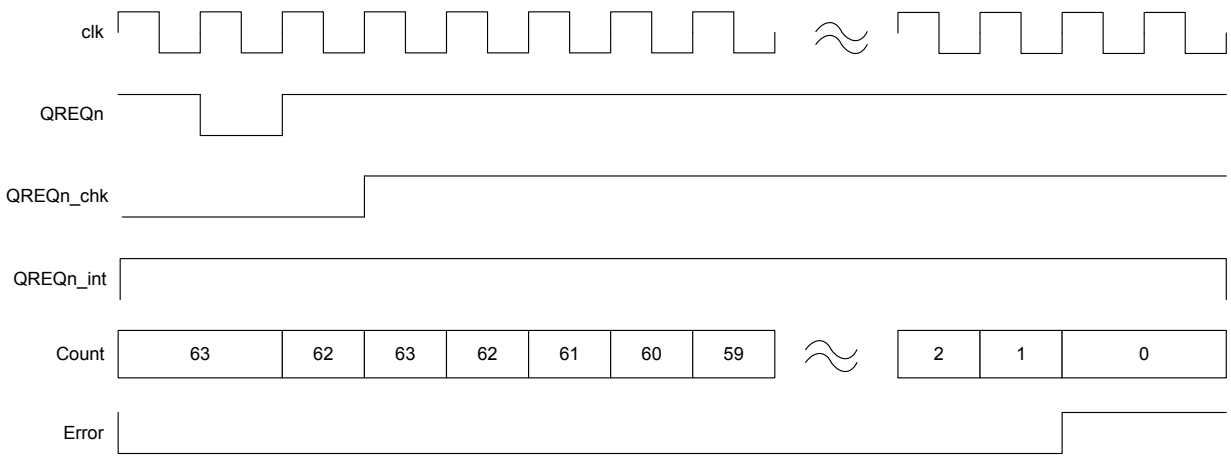

Figure 4-19 Stuck-at-one error on qreqn

The following figure shows how the SAF detector detects a stuck-at-one error on **qreqn_chk**.

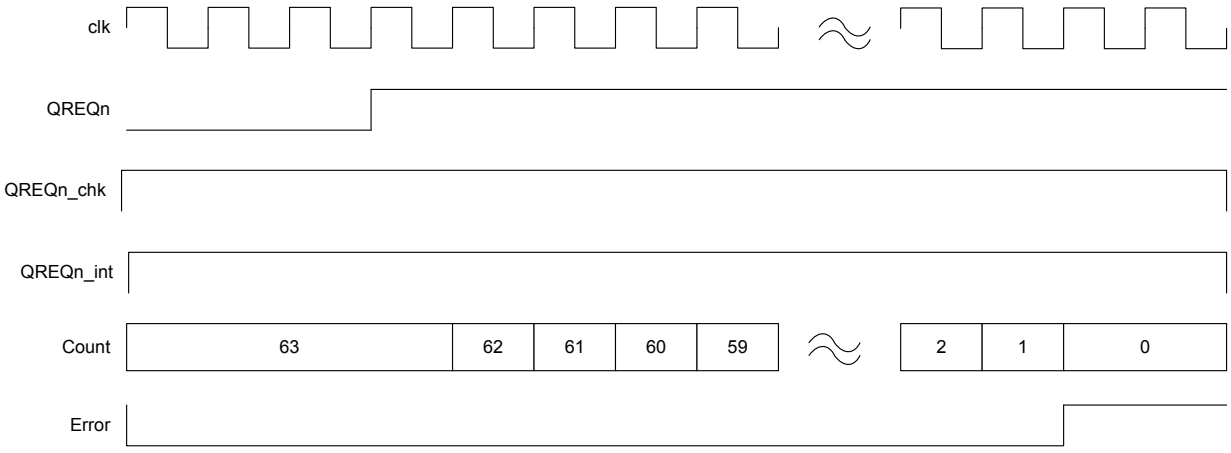

Figure 4-20 Stuck-at-one error on qreqn_chk

4.10.7 Disabling Q-Channel Safety Mechanisms

The FMU_SMEN register cannot disable the Q-Channel SMs. They can be disabled during design time using one of the following methods:

- To disable specific Q-Channel SMs, tie the **qreqn_chk** bit to the value of !**qreqn** on the Q-Channel interface that you want to disable protection for.
- To disable all Q-Channel SMs in the MMU-600AE, set `FUSA_DISABLE_PQCHAN_PROT`=1. Setting this parameter disables the following Q-Channel SMs:
  — TCU SM 5.
  — TCU SM 6.
  — TBU SM 5.
  — TBU SM 6.

4.11 Systematic fault watchdog protection

MMU-600AE contains a watchdog-based PING/ACK mechanism that guards against systematic errors on the interconnect.

It engages a hardware mechanism in the MMU TCU, which pings each MMU block in a round-robin fashion and waits for a response. If a response is not received within a programmable timeout window, a fault is reported. This mechanism can guard against:

- Lockup on the interconnect that connects the MMU blocks.
- Possible lockup on external buses that causes the MMU blocks and internal interconnect to stall.

The source of the lockup might be software issues, DoS issues, or systematic faults in the silicon.

For more information on this feature see *4.5.7 Ping mechanism* on page 4-118.

Appendix A
Signal descriptions

This appendix describes the MMU-600AE external signals.

It contains the following sections:

A.1 Clock and reset signals

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

The following table shows the clock and reset signals.

Table A-1 Clock and reset signals

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

A.2 TCU QTW/DVM interface signals

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

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

Table A-2 TCU QTW/DVM interface signals

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

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

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

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

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

A.3 TCU programming interface signals

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

The following table shows the TCU programming interface signals.

Table A-3 TCU programming interface signals

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

A.4 TCU SYSCO interface signals

The following table shows the TCU SYSCO interface signals.

Table A-4 TCU SYSCO interface signals

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

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

A.5 TCU PMU snapshot interface signals

The following table shows the TCU PMU snapshot interface signals.

Table A-5 TCU PMU snapshot interface signals

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

A.6 TCU LPI_PD interface signals

The following table shows the TCU LPI_PD interface signals.

Table A-6 TCU LPI_PD interface signals

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

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

A.7 TCU LPI_CG interface signals

The following table shows the TCU LPI_CG interface signals.

Table A-7 TCU LPI_CG interface signals

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

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

A.8 TCU DTI interface signals

The following table shows the TCU DTI interface signals.

Table A-8 TCU DTI interface signals

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

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

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

A.9 TCU interrupt signals

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

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

The following table shows the TCU interrupt signals.

Table A-9 TCU interrupt interface signals

Signal	Direction	Description
event_q_irpt_s	Output	Event queue, Secure interrupt. Asserts a Secure interrupt to indicate that the Event queue is not empty or has overflowed.
event_q_irpt_ns	Output	Event queue, Non-secure interrupt. Asserts a Non-secure interrupt to indicate that the Event queue is not empty or has overflowed.
cmd_sync_irpt_ns	Output	SYNC complete, Non-secure interrupt. Asserts a Non-secure interrupt to indicate that the `CMD_SYNC` command is complete.
cmd_sync_irpt_s	Output	SYNC complete, Secure interrupt. Asserts a Secure interrupt to indicate that the `CMD_SYNC` command is complete.
global_irpt_ns	Output	Asserts a global Non-secure interrupt.
global_irpt_s	Output	Asserts a global Secure interrupt.
ras_irpt	Output	Asserts a *Reliability, Availability, and Serviceability* (RAS) interrupt. ——— **Note** ——— The MMU-600AE cannot output RAS interrupts as MSIs. You must connect this output to an interrupt controller. ———
pmu_irpt	Output	Asserts a PMU interrupt. ——— **Note** ——— The MMU-600AE cannot output PMU interrupts as MSIs. You must connect this output to an interrupt controller. ———
pri_q_irpt_ns	Output	Asserts a *Page Request Interface* (PRI) queue interrupt.

A.10 TCU event interface signal

The TCU event interface signal is an event output for connection to processors.

The following table shows the TCU event interface signal.

Table A-10 TCU event interface signal

Signal	Direction	Description
evento	Output	The **evento** signal is asserted for one cycle to indicate an event that enables processors to wake up from the *Wait For Event* (WFE) low-power state. Connect the **evento** signal of the TCU to the event interface of Arm processors. Processors that use the *DynamIQ Shared Unit* (DSU) have a different event handshake mechanism. The mechanism that the DSU uses is the successor to the mechanism that some MMUs use. Arm processors can use the following event mechanisms: • Some processors have an **eventi** input to connect directly to the **evento** output from the MMU. • Some processors, including DSU-based systems, have a **req**/**ack** handshake mechanism that requires the **evento** signal from the MMU to be converted and uses the **eventiack**, **eventireq**, **eventoack**, and **eventoreq** signals. ——— **Note** ——— You can also route the **evento** signal through other interconnects such as the Arm CoreLink CMN-600 Coherent Mesh Network instead of connecting **evento** directly to the processor. These interconnects, like the DSU, only support the newer event mechanism. ——— If the rest of your system uses the newer event mechanism, you must add logic to convert events that the MMU-600AE generates, which uses the older event mechanism. In both mechanisms, in the signal names: **i** Represents events that are inputs to a particular component. **o** Represents events that are outputs from a particular component. ——— **Note** ——— For the signals, the handshake mechanism uses one input and one output in each direction. This is because the acknowledgment of the request operates in the opposite direction to the original request. ——— The MMU-600AE has an event output and therefore only has the **evento** signal. The processor has an input interface to receive the event from the MMU-600AE, and other devices. This input interface uses the **eventiack** and **eventireq** signals, if the processor uses the newer mechanism. The required conversion is from the older mechanism, **eventi** and **evento** signals, to the newer mechanism, **eventiack**, **eventireq**, **eventoack**, and **eventoreq** signals. When connecting the MMU-600AE to a DSU, the only signals to consider are the following: • **evento** signal of the MMU-600AE. • **eventiack** and **eventireq** signals of the DSU. Some processors have an **eventi** input instead. You can use the *Channel Pulse to Channel adapter* that is provided in the CoreSight System-on-Chip SoC-600. See *Chapter 6.11* in the *Arm® CoreSight™ System-on-Chip SoC-600 Technical Reference Manual* for more information about this component. ——— **Note** ——— To use the *Channel Pulse to Channel adapter* from CoreSight System-on-Chip SoC-600, you must be a licensee of the SoC-600 product. If you are not a licensee of SoC-600, you must add your own logic. ———

For more information, see the documentation for your processor or DSU.

A.11 TCU tie-off signals

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

The following table shows the TCU tie-off signals.

Table A-11 TCU tie-off signals

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

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

A.12 TCU and TBU test and debug signals

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

The following table shows the test and debug signals.

Table A-12 Test and debug signals

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

A.13 TBU TBS interface signals

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

The following table shows the TBU TBS interface signals.

Table A-13 TBU TBS interface signals

Signal	Direction	Description
aclk	Input	Clock input.
araddr_s	Input	Read address.
arburst_s	Input	Burst type.
arcache_s	Input	Memory type.
ardomain_s	Input	Shareability domain.
aresetn	Input	Active-LOW reset signal.
arid_s	Input	Read address ID.
arlen_s	Input	Burst length.
arlock_s	Input	Lock type.
arprot_s	Input	Protection type.
arqos_s	Input	*Quality of Service* (QoS).
arready_s	Output	Read address ready.
arregion_s	Input	Region identifier.
arsize_s	Input	Burst size.
armmussid_s	Input	These signals indicate the StreamID, SubstreamID, and ATS translated status of the originating transaction. These signals are defined by the AXI5 Untranslated_Transactions extension.
armmusid_s	Input	
armmussidv_s	Input	
armmusecsid_s	Input	
armmuatst_s	Input	
arvalid_s	Input	Read address valid.
awaddr_s	Input	Write address.
awatop_s	Input	Atomic operation.
awburst_s	Input	Burst type.
awcache_s	Input	Memory type.

Table A-13 TBU TBS interface signals (continued)

Signal	Direction	Description
awdomain_s	Input	Shareability domain.
awid_s	Input	Write address ID.
awlen_s	Input	Burst length.
awlock_s	Input	Lock type.
awprot_s	Input	Protection type.
awqos_s	Input	QoS.
awready_s	Output	Write address ready.
awregion_s	Input	Region identifier.
awsize_s	Input	Burst size.
awmmussid_s	Input	These signals indicate the StreamID, SubstreamID, and ATS translated status of the originating transaction. These signals are defined by the AXI5 Untranslated_Transactions extension.
awmmusid_s		
awmmussidv_s		
awmmusecsid_s		
awmmuatst_s		
awvalid_s	Input	Write address valid.
bid_s	Output	Response ID.
bready_s	Input	Response ready.
bresp_s	Output	Write response.
bvalid_s	Output	Write response valid.
rdata_s	Output	Read data.
rid_s	Output	Read ID.
rlast_s	Output	Read last.
rready_s	Input	Read ready.
rresp_s	Output	Read response.
rvalid_s	Output	Read valid.
wdata_s	Input	Write data.
wlast_s	Input	Write last.

Table A-13 TBU TBS interface signals (continued)

Signal	Direction	Description
wready_s	Output	Write ready.
wstrb_s	Input	Write strobes.
wvalid_s	Input	Write valid.
aruser_s	Input	Read address (AR) channel user signal.
awuser_s	Input	Write address (AW) channel user signal.
wuser_s	Input	Write data (W) channel user signal.
ruser_s	Output	Read data (R) channel user signal.
buser_s	Output	Write response (B) channel user signal.
awakeup_s	Input	Wakeup signal.
arsnoop_s	Input	Transaction type of read transaction.
awsnoop_s[3]	Input	Transaction type of write transaction.
awstashnid_s[10:0]	Input	These signals are defined by the AXI5 Cache_Stash_Transactions extension. If `TBUCFG_STASH` = 0, these signals are ignored.
awstashniden_s	Input	
awstashlpid_s[4:0]	Input	
awstashlpiden_s	Input	

A.14 TBU TBM interface signals

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

The following table shows the TBU TBM interface signals.

Table A-14 TBU TBM interface signals

Signal	Direction	Description
aclk	Input	Clock input.
araddr_m	Output	Read address.
arburst_m	Output	Burst type.
arcache_m	Output	Memory type.
ardomain_m	Output	Shareability domain.
aresetn	Input	Active-LOW reset signal.
arid_m	Output	Read address ID.
arlen_m	Output	Burst length.
arlock_m	Output	Lock type.
arprot_m	Output	Protection type.
arqos_m	Output	*Quality of Service* (QoS).
arready_m	Input	Read address ready.
arregion_m	Output	Region identifier.
arsize_m	Output	Burst size.
armmusid_m	Output	These signals indicate the StreamID of the originating transaction.
armmusecsid_m	Output	
arvalid_m	Output	Read address valid.
awaddr_m	Output	Write address.
awatop_m	Output	Atomic operation.
awburst_m	Output	Burst type.
awcache_m	Output	Memory type.
awdomain_m	Output	Shareability domain.
awid_m	Output	Write address ID.
awlen_m	Output	Burst length.

Table A-14 TBU TBM interface signals (continued)

Signal	Direction	Description
awlock_m	Output	Lock type.
awprot_m	Output	Protection type.
awqos_m	Output	QoS.
awready_m	Input	Write address ready.
awregion_m	Output	Region identifier.
awsize_m	Output	Burst size.
awmmusid_m	Output	These signals indicate the StreamID of the originating transaction. The *Generic Interrupt Controller* (GIC) uses these signals to determine the DeviceID of MSIs that originate from upstream masters.
awmmusecsid_m	Output	
awvalid_m	Output	Write address valid.
bid_m	Input	Response ID.
bready_m	Output	Response ready.
bresp_m	Input	Write response.
bvalid_m	Input	Write response valid.
rdata_m	Input	Read data.
rid_m	Input	Read ID.
rlast_m	Input	Read last.
rready_m	Output	Read ready.
rresp_m	Input	Read response.
rvalid_m	Input	Read valid.
wdata_m	Output	Write data.
wlast_m	Output	Write last.
wready_m	Input	Write ready.
wstrb_m	Output	Write strobes.
wvalid_m	Output	Write valid.
aruser_m	Output	Read address (AR) channel user signal.
awuser_m	Output	Write address (AW) channel user signal.
wuser_m	Output	Write data (W) channel user signal.

Table A-14 TBU TBM interface signals (continued)

Signal	Direction	Description
ruser_m	Input	Read data (R) channel user signal.
buser_m	Input	Write response (B) channel user signal.
awakeup_m	Output	Wakeup signal.
arsnoop_m	Output	Transaction type of read transaction.
awsnoop_m[3]	Output	Transaction type of write transaction.
awstashnid_m[10:0]	Output	These signals are defined by the AXI5 Cache_Stash_Transactions extension. If TBUCFG_STASH = 0, these signals are ignored.
awstashniden_m	Output	
awstashlpid_m[4:0]	Output	
awstashlpiden_m	Output	

A.15 TBU PMU snapshot interface signals

The following table shows the TBU PMU snapshot interface signals.

Table A-15 TBU PMU snapshot interface signals

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

A.16 TBU LPI_PD interface signals

The following table shows the TBU LPI_PD interface signals.

Table A-16 TBU LPI_PD interface signals

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

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

A.17 TBU LPI_CG interface signals

The following table shows the TBU LPI_CG interface signals.

Table A-17 TBU LPI_CG interface signals

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

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

A.18 TBU DTI interface signals

The following table shows the TBU DTI interface signals.

Table A-18 TBU DTI interface signals

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

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

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

A.19 TBU interrupt signals

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

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

The following table shows the TBU interrupt signals.

Table A-19 TBU interrupt signals

Signal	Direction	Description
ras_irpt	Output	RAS interrupt
pmu_irpt	Output	PMU interrupt

A.20 TBU tie-off signals

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

The following table shows the TBU tie-off signals.

Table A-20 TBU tie-off signals

Signal	Direction	Description
ns_sid_high[23:TBUCFG_SID_WIDTH]	Input	Provides the high-order StreamID bits for all transactions with a Non-secure StreamID that pass through the TBU.
s_sid_high[23:TBUCFG_SID_WIDTH]	Input	Provides the high-order StreamID bits for all transactions with a Secure StreamID that pass through the TBU.
max_tok_trans[log2(TBUCFG_XLATE_SLOTS)-1:0]	Input	Indicates the number of DTI translation tokens to request when connecting to the TCU, minus 1.
pcie_mode	Input	You must tie this signal HIGH when the TBU is connected to a PCIe interface. When this signal is HIGH, the TBU behaves as if the PCIe 'No Snoop' property is applied to transactions downstream of the SMMU, as long as the PCIe interface outputs transactions with the following AXI memory types: **Normal Non-Cacheable Bufferable** When 'No Snoop' is set for the transaction **Write-Back** When 'No Snoop' is not set for the transaction This TBU behavior is a requirement of the *Arm Server Base System Architecture*. If this signal is HIGH, the attributes of TBS interface transactions are always combined with the translation attributes, even if stage 1 translation is enabled. That is, the transaction attributes are always calculated as if the DTI_TBU_TRANS_RESP.STRW field is EL1-S2, regardless of the actual STRW value. If this signal is HIGH, the input attribute and shareability override information in the ATTR_OVR field of the DTI_TBU_TRANS_RESP message is ignored. For SMMUv3, PCIe masters do not support this feature.
sec_override	Input	When HIGH, certain registers are accessible to Non-secure accesses from reset, as the TBU_SCR register settings describe. See *3.11.2 TBU_SCR* on page 3-92.
ecorevnum[3:0]	Input	Tie this signal to 0 unless directed otherwise by Arm.

Table A-20 TBU tie-off signals (continued)

Signal	Direction	Description
utlb_roundrobin	Input	Defines the Micro TLB entry replacement policy. When LOW, the Micro TLB uses a *Pseudo Least Recently Used* (PLRU) replacement policy. This policy typically provides the best average performance. When HIGH, the Micro TLB uses a round-robin replacement policy. With this policy, the oldest entry is evicted when the Micro TLB is full. Tie this signal HIGH if you want to prevent newer translations from being evicted, even if older translations have been used more recently. Otherwise, tie this signal LOW.
cmo_disable	Input	To disable cache maintenance operations, tie this signal HIGH. When this signal is HIGH, the following transactions are always aborted with an SLVERR response: • CleanInvalid • CleanShared • CleanSharedPersist • MakeInvalid Cache maintenance operations can sometimes break the requirements of limited sideband channel communication, such as when a master component accesses protected content. You can disable cache maintenance operations in such cases.

Related information

3.7.5 TCU_SCR on page 3-81

A.21 DTI interconnect switch signals

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

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

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

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

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

Table A-22 DTI interconnect switch UP_S*n* interface signals

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

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

Table A-23 DTI interconnect switch DN_M interface signals

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

Table A-23 DTI interconnect switch DN_M interface signals (continued)

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

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

Table A-24 DTI interconnect switch UP_M interface signals

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

A.22 DTI interconnect sizer signals

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

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

Table A-25 DTI interconnect sizer LPI_CG interface signals

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

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

Table A-26 DTI interconnect sizer DN_S interface signals

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

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

Table A-27 DTI interconnect sizer UP_S interface signals

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

Table A-27 DTI interconnect sizer UP_S interface signals (continued)

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

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

Table A-28 DTI interconnect sizer DN_M interface signals

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

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

Table A-29 DTI interconnect sizer UP_M interface signals

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

A.23 DTI interconnect register slice signals

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

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

Table A-30 DTI interconnect register slice LPI_CG interface signals

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

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

Table A-31 DTI interconnect register slice DN_S interface signals

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

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

Table A-32 DTI interconnect register slice UP_S interface signals

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

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

Table A-33 DTI interconnect register slice DN_M interface signals

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

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

Table A-34 DTI interconnect register slice UP_M interface signals

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

Appendix B
Software initialization examples

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

It contains the following sections:

B.1 Initializing the SMMU

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

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

Note

This section does not describe how to create translation tables. See the *Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile* for more information.

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

This section contains the following subsections:

B.1.1 Allocating the Command queue

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

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

Procedure

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

   Note

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

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

   Note

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

B.1.2 Allocating the Event queue

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

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

Procedure

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

   Note

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

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

   Note

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

B.1.3 Configuring the Stream table

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

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

Procedure

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

   If SMMU_IDR0.COHACC = 0, the system does not support coherent access to memory for the TCU. In such cases, you might require extra steps to ensure that the SMMU can observe the written data.

B.1.4 Initializing the Command queue

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

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

Procedure

1. Enable the Command queue by setting the SMMU_CR0.CMDQEN bit to 1.
2. Check that the enable operation is complete by polling SMMU_CR0ACK until CMDQEN reads as 1.

B.1.5 Initializing the Event queue

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

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

Procedure

1. Enable the Event queue by setting the SMMU_CR0.EVENTQEN bit to 1.

2. Check that the enable operation is complete by polling SMMU_CR0ACK until EVENTQEN reads as 1.

B.1.6 Invalidating TLBs and configuration caches

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

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

Non-secure EL1 contexts	Issue `CMD_TLBI_NSNH_ALL`
EL2 contexts	Issue `CMD_TLBI_EL2_ALL`
EL3 contexts	Issue `CMD_TLBI_EL3_ALL`
Secure EL1 contexts	Issue `CMD_TLBI_NH_ALL`

Note

Commands to invalidate Secure TLB entries can only be issued through the Secure Command queue. For a system that implements two security states, Secure software must issue the appropriate command to the Secure Command queue for the first TLB invalidation. If your system does not use Secure software, you can permit Non-secure software to access SMMU_S_INIT by using **sec_override**. See *A.11 TCU tie-off signals* on page Appx-A-164 and *A.20 TBU tie-off signals* on page Appx-A-177.

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

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

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

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

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

B.1.7 Creating a basic Context Descriptor

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

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

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

Table B-1 Configuring the CD

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

Table B-1 Configuring the CD (continued)

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

B.1.8 Creating a Stream Table Entry

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

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

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

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

   `0b100` Stage 1 and Stage 2 bypass.

   `0b101` Stage 1 translation Stage 2 bypass.

   `0b110` Stage 1 bypass Stage 2 translation.

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

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

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

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

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

   STE.S2PS Contains the PA size of the stage 2 PA range.

   STE.S2AA64 Indicates whether the Stage 2 tables are AArch32 or AArch64 format.

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

B.2 Enabling the SMMU

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

To enable the SMMU, carry out the following procedure.

Procedure

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

Appendix C
Revisions

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

It contains the following section:

C.1 Revisions

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

Table C-1 Issue 0000-00 BET

Change	Location	Affects
First release for r0p0 BET.	-	-

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

Change	Location	Affects
Removed references to P-Channel.	Throughout the document.	All revisions.
Updated TBU register address ranges.	*Table 3-3 MMU-600AE memory map* on page 3-69.	All revisions.
Updated TCU registers, page 0 description.	*Table 3-4 MMU-600AE TCU memory map* on page 3-69.	All revisions.
Updated TBU registers, page 0 description.	*Table 3-5 MMU-600AE TBU memory map* on page 3-70.	All revisions.
Corrected all port names.	*Table 4-2 Q-Channel FuSa ports* on page 4-105.	All revisions.
Corrected **nmbistresetn** signal name.	*Table 4-4 Protected MBIST inputs* on page 4-107.	All revisions.
Removed **dftse** signal.	*Table 4-5 Duplicate ATPG input ports* on page 4-108.	All revisions.
Corrected **fmu_fault_int**, **fack**, and **freq** signal names.	*Figure 4-3 FMU in MMU block* on page 4-110.	All revisions.
Corrected signal names.	*Reset* on page 4-111.	All revisions.
Added TBU8 to TBU14.	*Table 4-6 MMU block IDs* on page 4-112.	All revisions.
Updated IDs for TCU tie-off or Interrupt error and TBU tie-off or Interrupt error. Added TCU FMU clock gating override and TBU FMU clock gating override SMs.	*Table 4-7 Safety Mechanisms* on page 4-113.	All revisions.
Added specific key value.	*4.5.6 Lock and key mechanism* on page 4-117.	All revisions.
Updated register summary table to include new registers.	*Table 4-8 FMU PV registers* on page 4-121.	All revisions.
Added CE_EN bitfield.	*FMU_ERR<n>CTLR, Error Record Control Register* on page 4-122.	All revisions.
Added BLKID and CE bitfields. Updated description for V bitfield.	*FMU_ERR<n>STATUS, Error Record Primary Status Register* on page 4-123.	All revisions.
Added new register description.	*FMU_ERRGSR, Error Group Status Register* on page 4-125.	All revisions.
Updated bit range for ping_timeout_value.	*FMU_PINGCTLR, Ping Control Register* on page 4-126.	All revisions.
Added new register description.	*FMU_PINGMASK, Ping Mask Register* on page 4-129.	All revisions.
Updated bit range for idle bitfield.	*FMU_STATUS, FMU Status Register* on page 4-130.	All revisions.
Updated reset value for NUM bitfield.	*FMU_ERRIDR, Error Record ID Register* on page 4-130.	All revisions.
Added new figure.	*Figure 4-9 F-Channel and FMU fault wire connections* on page 4-139.	All revisions.

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

Change	Location	Affects
Added new section.	*4.8.5 Interrupt output protection* on page 4-140.	All revisions.
Added new section.	*4.8.6 Tie-off input protection* on page 4-140.	All revisions.
Updated figures.	*4.10 Q-Channel protection* on page 4-142.	All revisions.
Removed redundant text.	'Configuration and parameters' section that is no longer contained in this document.	All revisions.
Added **rresp_s** signal.	*Table A-13 TBU TBS interface signals* on page Appx-A-166.	All revisions.
Added **rresp_m** signal.	*Table A-14 TBU TBM interface signals* on page Appx-A-169.	All revisions.

Table C-3 Differences between Issue 0000-01 EAC and Issue 0100-00 REL

Change	Location	Affects
Corrections to the minor revision, MAX[`0x1`, **ecorevnum**]	• *2.4.1 SMMUv3 support* on page 2-54. • *3.5 TCU component and peripheral ID registers* on page 3-74. • *3.6 TCU PMU component and peripheral ID registers* on page 3-75. • *3.9 TBU component and peripheral ID registers* on page 3-90. • *3.10 TBU PMU component and peripheral ID registers* on page 3-91.	r1p0.
Updates to the description of TCU prefetching	*2.3.8 TCU prefetch* on page 2-48.	All revisions.
Correction to TCU_ERRFR reset value.	*3.8.1 TCU_ERRFR* on page 3-85.	All revisions.
Correction to TCU ERRCTRL.FI description.	*3.8.2 TCU_ERRCTLR* on page 3-85.	All revisions.
Correction to TBU_ERRFR reset value.	*3.12.1 TBU_ERRFR* on page 3-94.	All revisions.
Correction to TBU ERRCTRL.FI description.	*3.12.2 TBU_ERRCTLR* on page 3-94.	All revisions.
Updated the description, category, and location of the **evento** signal.	*A.10 TCU event interface signal* on page Appx-A-162.	All revisions.
Updated the description of the '*Distributed Virtual Memory* (DVM) messages' section.	*2.3.6 Distributed Virtual Memory (DVM) messages* on page 2-46.	All revisions.
Updated the description of the **cmo_disable** TBU tie-off signal.	*A.20 TBU tie-off signals* on page Appx-A-177.	All revisions.
Updated the description of the TCU memory map.	*3.3 MMU-600AE memory map* on page 3-69.	All revisions.
Improved content descriptions.	Throughout the document.	All revisions.
Added descriptions for the DTI interconnect sizer.	Throughout the document.	All revisions.
Multiple updates that are related to FuSa.	*Chapter 4 Functional Safety* on page 4-98.	All revisions.