Arm® CoreLink™ AHB Cache

Revision: r0p0

Technical Reference Manual

101807_0000_03_en

Arm® CoreLink™ AHB Cache

Technical Reference Manual

Release Information

Document History

Issue	Date	Confidentiality	Change
0000-01	13 December 2019	Confidential	First beta release for r0p0.
0000-02	24 April 2020	Non-Confidential	First early access release for r0p0.
0000-03	06 October 2020	Non-Confidential	First full release for r0p0.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

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™ AHB Cache Technical Reference Manual

Preface

This preface introduces the *Arm® CoreLink™ AHB Cache Technical Reference Manual*.

It contains the following:
- *About this book* on page 7.
- *Feedback* on page 10.

About this book

This book describes the functionality of the components in the Arm® CoreLink™ AHB Cache. It also provides the programming information and the signal descriptions.

Product revision status

The r*x*p*y* identifier indicates the revision status of the product described in this book, for example, r1p2, where:

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

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

Intended audience

This book is written for system designers and programmers who are designing or programming a *System on Chip* (SoC) that uses the AHB Cache.

Using this book

This book is organized into the following chapters:

Chapter 1 Overview
This chapter introduces the AHB Cache.

Chapter 2 Interfaces
This chapter describes the functional interfaces of the AHB Cache.

Chapter 3 Operation
This chapter describes the operation of the AHB Cache.

Chapter 4 Programmers model
This chapter describes the functionality of the AHB Cache from a programming perspective.

Chapter 5 Using software to program the AHB Cache
This chapter provides details on programming the AHB Cache by exploring typical scenarios.

Appendix A Signal descriptions
This appendix describes the AHB Cache interface signals.

Appendix B 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.

monospace
: 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.

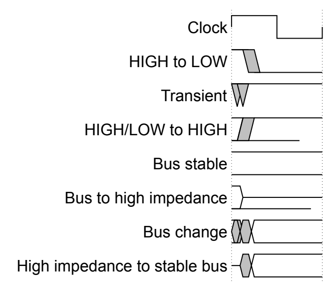

Figure 1 Key to timing diagram conventions

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

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

Arm Publications

Document name	Document ID	Licensee only Y/N
Arm® CoreLink™ AHB Cache Configuration and Integration Manual	101808	Y
Arm® AMBA® 5 AHB Protocol Specification, issue B.b	IHI 0033	N
AMBA® APB Protocol Specification Version 2.0, issue C	IHI 0024	N
AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces, Issue C	IHI 0068	N

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 AHB Cache Technical Reference Manual*.
- The number 101807_0000_03_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

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

Chapter 1
Overview

This chapter introduces the AHB Cache.

It contains the following sections:

1.1 Basic terms

You should be familiar with the terms that are used to describe the AHB Cache and its operation.

This document uses the terms that are defined in the *Arm® AMBA® 5 AHB Protocol Specification, issue B.b* to describe AHB transfers:

- Bufferable
- Modifiable
- Lookup
- Allocate

The following table describes the complex transfer attributes that are used by AHB Cache.

Table 1-1 Complex attributes

Name	Signals asserted	AHB transfer type	Description
Cacheable	**HPROT[4:3]**	A transfer that is both Modifiable and Lookup.	A Cacheable transfer is looked up in the cache. If a transfer is not Cacheable, it is passed through.
Write-Back	**HPROT[4:2]**	A Cacheable Write transfer for which the Bufferable attribute is set.	A Write-Back transfer updates the data in the cache if the cache line is already allocated (or being allocated due to this transfer). The main memory is only updated when the cache line is evicted.
Write-Through	**HPROT[4:3]**	A Cacheable Write transfer for which the Bufferable attribute is not set.	A Write-Through transfer updates the data in the cache if the cache line is already allocated (or being allocated due to this transfer). A Write-Through transfer forwards the transaction to update the data in the main memory.
Early response	**HPROT[2]** or **HPROT[3]**	Writes that are Bufferable or Modifiable.	When a Write-Access is buffered, the AHB Cache can send an early write response to the Write-Access without waiting for the main memory to respond.

The basic operations of the AHB Cache are described in *3.1 Basic operations* on page 3-35.

Other caching terms

Cache line

A cache line is the unit of data transfers between the cache and main memory. The AHB Cache has 32-byte cache lines.

Index

A part of the address of a Cacheable access which is used to select between the sets of lines in the cache.

Set

A group of lines with the same index. For a 4-way cache, each set has four lines.

Tag

A part of the address of a Cacheable access that is stored in one of the four ways of the tag RAM when allocating a cacheline. This is the part of the address that is compared to a maximum of four valid tags stored in the four ways during a lookup.

Streaming

If a cacheable transfer addresses a word that is currently being fetched as part of a linefill, the response and read data is streamed directly to the transfer. In this case, the response and read data does not have to wait for the word to be written to the linefill buffer first.

Linefill

See *3.1.4 Linefill* on page 3-36.

Eviction

See *3.1.5 Eviction* on page 3-36.

Cache hit and cache miss

See *3.1.3 Lookup* on page 3-35.

1.2 About the AHB Cache

The AHB Cache is a configurable cache which improves performance for IoT devices. The AHB Cache is designed to reduce the effect of high latency or slow memory on system performance.

The AHB Cache can help reduce both system memory bandwidth used and access latency by storing recently accessed memory contents for reuse. Reducing system memory accesses may also help save power at system level.

The AHB Cache can be integrated to connect directly to a processor. It can be implemented as a processor cache (data or generic), or a system cache. It can be used for both code and data.

The cache provides AHB5 data interfaces and an APB configuration interface, both with Arm TrustZone® for Armv8-M support. The configuration interface is designed to run at the AHB bus frequency, but it supports running the APB bus on a slower clock by using the **pclken** clock enable input.

The AHB Cache is a non-coherent cache. The AHB5 **HPROT[6]** (Shareable) attribute does not prevent a transfer from being cached or buffered. For more information about how to integrate the AHB Cache into a coherent system, see *Arm® CoreLink™ AHB Cache Configuration and Integration Manual*.

The AHB Cache has the following features:
- AHB5 data interfaces with 32-bit wide data and address bus
- Zero wait state for cache hit accesses
- 4-way set associativity
- TrustZone for Armv8-M support
- Configurable cache size (2KB-64KB)
- 32-byte cache lines
- Write-Through and Write-Back policy support
- Forceable Write-Through policy
- Pseudo-random replacement policy
- AMBA 4 *Low Power Interface* (LPI) Q-Channel interfaces for clock and power management.
- Internal buffers for temporarily storing cache lines

Figure 1-1 AHB Cache overview on page 1-15 shows how the AHB Cache connects to the processor and the system memory.

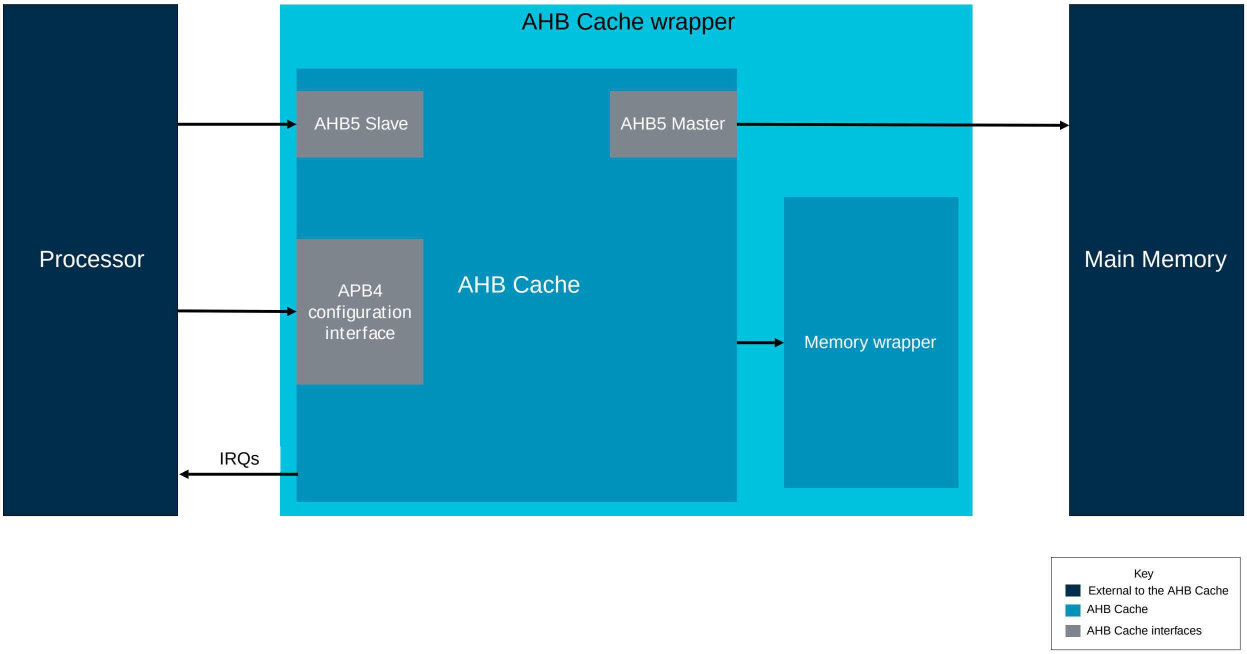

Figure 1-1 AHB Cache overview

Figure 1-2 AHB Cache wrapper on page 1-16 shows the AHB Cache wrapper.

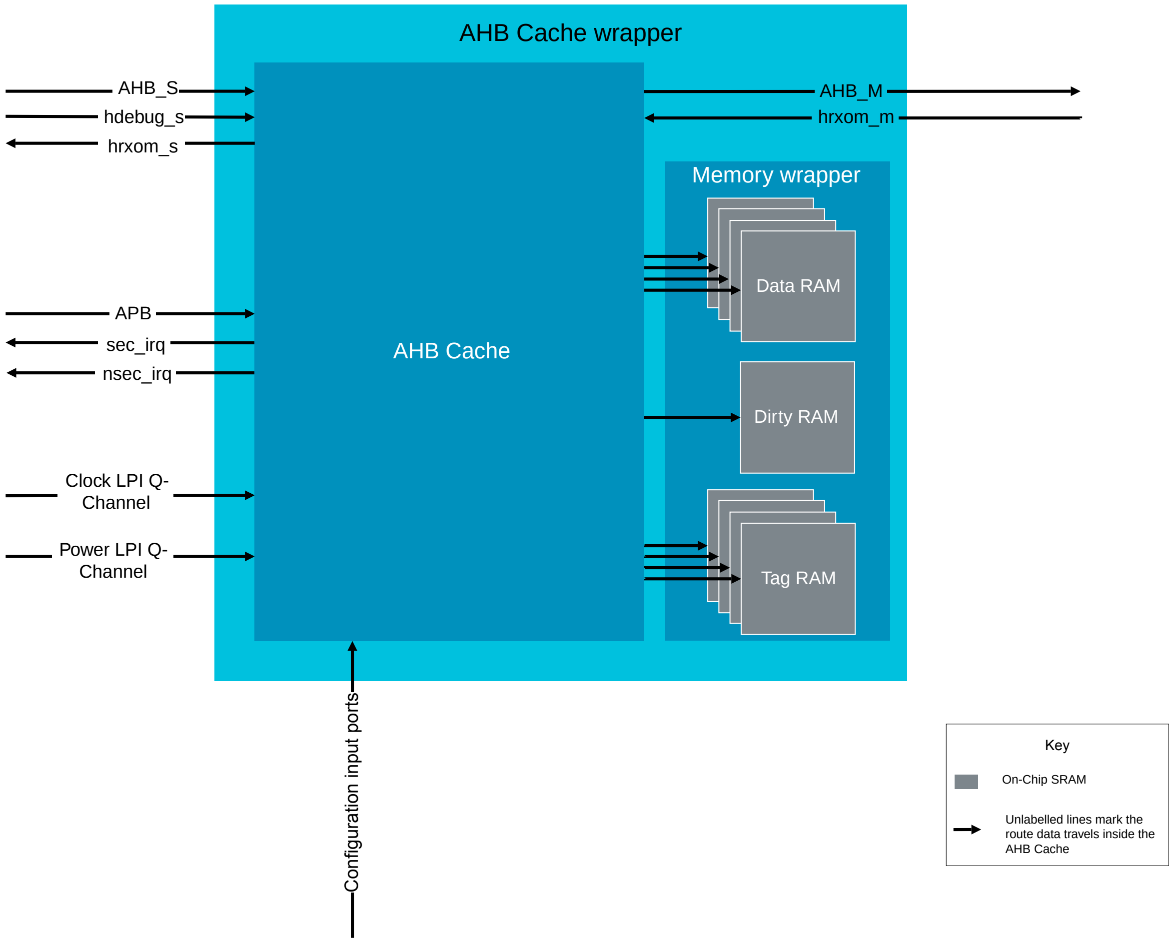

Figure 1-2 AHB Cache wrapper

For the possible configurations of the AHB Cache, see *1.5 Implementations* on page 1-19 and the *Arm® CoreLink™ AHB Cache Configuration and Integration Manual*. The AHB Cache is not designed for safety critical applications.

1.3 Configurable features

The AHB Cache provides configurable features.

The AHB Cache configurable features are:

- Write-Through and Write-Back policy support with forceable Write-Through policy. For more information, see *2.3 APB interface* on page 2-29.
- Configurable *eXecute Only Memory* (XOM) support. For more information, see *2.2.6 XOM* on page 2-27.
- Configurable automatic maintenance. For more information, see *3.3.7 Automatic maintenance features* on page 3-39.
- Configurable AHB and APB violation responses. For more information, see *2.2.6 XOM* on page 2-27 and *2.3 APB interface* on page 2-29.
- Performance monitoring with configurable snapshotting. For more information, see *3.2 Performance monitoring* on page 3-37.
- Configurable automatic **power_on _enable**. For more information, see *3.3.9 power_on_enable* on page 3-43.

1.4 Compatibility

The AHB Cache is compatible with several other products.

Compatible processors

The AHB Cache is compatible with the following Arm Cortex®-M processors:

- Cortex-M0
- Cortex-M0+
- Cortex-M3
- Cortex-M4
- Cortex-M23
- Cortex-M33

Other compatible products

The AHB Cache is compatible with the following products:

- CoreLink SIE-200 System IP for Embedded
- CoreLink PCK-600 Power Control Kit

1.5 Implementations

The AHB Cache can be implemented as a processor cache or a system cache.

Processor cache

The AHB Cache can be implemented as:

- A dedicated data cache
- A dedicated instruction cache
- A generic cache

System cache

A system cache implementation of the AHB Cache can be used to:

- Share memory access among several masters
- Reduce latency when connecting an AHB processor to an AXI subsystem

For more details on how to implement the AHB Cache, see Chapter 2 *System Design with the AHB Cache* in the *Arm® CoreLink™ AHB Cache Configuration and Integration Manual*.

1.6 Compliance

Arm CoreLink AHB Cache is compliant with Arm specifications and protocols:

- *Arm® AMBA® 5 AHB Protocol Specification, issue B.b*
- *AMBA® APB Protocol Specification Version 2.0, issue C*
- *AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces, Issue C*

Note

The AHB Cache supports the Cortex-M0, M0+, M3, M4 processors even though they are not AHB5 compliant. For more information, see section 8.6.2 *Integration with non-AHB5 compliant processors* in the *Arm® CoreLink™ AHB Cache Configuration and Integration Manual*.

1.7 Product documentation

Documentation that is provided with this product includes a *Technical Reference Manual* (TRM) and a *Configuration and Integration Manual* (CIM), together with architecture and protocol information.

For relevant protocol and architectural information that relates to this product, see *Additional reading* on page 8.

The AHB Cache documentation is as follows:

Technical Reference Manual

The TRM describes the functionality and the effects of functional options on the behavior of AHB Cache. 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 the TRM describes are not relevant. If you are programming AHB Cache, contact:

- The implementer to determine:
  - The build configuration of the implementation
  - What integration, if any, was performed before implementing AHB Cache.
- The integrator to determine the signal configuration of the device that you use.

The TRM complements architecture and protocol specifications and relevant external standards. It does not duplicate information from these sources.

Configuration and Integration Manual

The CIM describes:

- The available build configuration options
- How to configure the RTL with the build configuration options
- How to integrate AHB Cache into an SoC
- How to implement AHB Cache into your design
- The processes to validate the configured design

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

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

1.8 Product revisions

This section describes the differences in functionality between product revisions.

r0p0 First release.

Chapter 2
Interfaces

This chapter describes the functional interfaces of the AHB Cache.

It contains the following sections:

2.1 Clocking and reset

The AHB Cache uses a single clock and a single reset.

The clock signal, **clk**, drives all the clocked logic, including the interfaces and the SRAM blocks.

The APB configuration interface uses a clock enable signal, **pclken**, to support APB running on a divided frequency. This enable signal must be periodical and synchronous to the clock, **clk**.

The cache uses a single, active-LOW reset, **resetn**. This reset must be deasserted synchronously with **clk** but it can be asserted asynchronously.

For more information, see the *Arm® CoreLink™ AHB Cache Configuration and Integration Manual*.

Related references

A.1 Clock and reset signals on page Appx-A-118

2.2 AHB interface

The AHB Cache has standard AHB5 data interfaces, which support XOM, locked sequences, and exclusive accesses.

The AHB Cache has an AHB5 Slave interface that receives transfers and serves them from the cache. The AHB Cache is optimized for single cycle hit latency on its AHB5 Slave interface.

The AHB Cache also has an AHB5 Master interface, which:
- Passes through transfers
- Fills data into the cache
- Writes back data from the cache

This section contains the following subsections:

2.2.1 Latency and stalling on the AHB interface

The AHB Cache operates transparently when disabled, without any latency added. When the cache is enabled, the AHB interface responds with some delay to certain transactions.

Normal latencies

Table 2-1 Normal access latencies on page 2-25 shows the normal access latencies. More latency cycles can occur when:
- Maintenance is ongoing.
- Linefill is ongoing.

Table 2-1 Normal access latencies

Added latency cycles	Cacheable transaction	Reason for latency
0	Hit or No-Allocate Write-Through miss	Normal operation
1	No-Allocate Write-Back miss	Forwarding the transfer after lookup
3	Allocate miss	Internal maintenance before linefill is started

AHB interface response to enabling and disabling the cache

When a register access on the APB interface enables or disables the cache, the new setting becomes effective at the next IDLE or NONSEQ transaction on the AHB interface. This operation takes place so that the AHB Cache can avoid changing behavior in the middle of an AHB Burst.

If cache enable maintenance is on when enabling the cache, the AHB interface continues to operate transparently.

If cache disable maintenance is on when disabling the cache, the cache stalls the AHB interface at the next NONSEQ transaction.

The AHB Slave interface remains stalled until the related clean all maintenance is completed. The duration of the maintenance depends on the number of dirty cache lines found in the cache memories.

See *Cache enable maintenance* on page 3-41 and *Cache disable maintenance* on page 3-42 for more information.

Write-Allocate

If a Write-Allocate transaction misses the cache, the cache response depends on whether the XOM configuration is enabled:

- If the XOM configuration is enabled, the transaction is stalled. The Write-Allocate transaction is only responded to after the related (critical) word of the triggered linefill has been received from main memory. While the critical word of the triggered linefill is being fetched, the AHB Slave interface remains stalled. Linefill is started with the critical word.
- If the XOM configuration is disabled, the cache buffers the written data and it does not need to wait for the linefill. The cache responds when the internal maintenance is completed.

2.2.2 Write-Through and Write-Back support

The AHB Cache supports both Write-Through and Write-Back policies.

By default, the cache selects the write policy based on the transfer attributes, as defined by the *Arm® AMBA® 5 AHB Protocol Specification, issue B.b*. For more information about Write-Through and Write-Back transfer attributes, see *1.1 Basic terms* on page 1-12.

Write-Through policy can also be forced if necessary. Forced Write-Through policy makes the cache always select the Write-Through attribute for cacheable transfers. For more information, see *4.4.2 CTRL, control register* on page 4-54.

The cache requires that a cache line is always accessed as either Write-Back or Write-Through only. Bursts must not cross between regions of different types. This is normally the case for a *Memory Protection Unit* (MPU) with a 1KB granularity.

2.2.3 Exclusive access sequences

The AHB interface supports exclusive accesses.

Exclusive accesses are triggered by **hexcl_s** and **hexcl_m**. Exclusive cacheable accesses are looked up in the cache. If the result is a hit, then the line is cleaned and invalidated before the access proceeds. Then the access continues as an exclusive non-cacheable access and is forwarded to the AHB Master interface. For more information, see the *Exclusive access* section of the *Arm® CoreLink™ AHB Cache Configuration and Integration Manual*.

Exclusive accesses do not trigger a linefill.

2.2.4 Locked accesses and locked sequences

The AHB interface supports locked accesses.

The AHB Cache responds to locked transfers as outlined by the *Arm® AMBA® 5 AHB Protocol Specification, issue B.b*.

IDLE transfers

When the cache is enabled, the AHB Cache inserts an unlocked IDLE transfer after the end of a locked sequence, as recommended by the AHB specification. If an unlocked IDLE transfer is already present after the end of the locked sequence, the AHB Cache does not insert another unlocked IDLE transfer.

hmastlock

The AHB Cache propagates **hmastlock** when the cache is transparent. While a locked access is being looked up in the cache, **hmastlock** appears on the AHB master interface.

If the lookup results in a hit and the cache line is dirty, the data needs to be written back before the locked access proceeds. Therefore, **hmastlock** is masked. This behavior produces an empty locked sequence. When the Write-Back is completed, **hmastlock** is reasserted.

Eviction Write-Back Burst

If an AHB locked sequence contains accesses to different addresses, an eviction Write-Back Burst can interrupt it.

Note

Normal use cases of locked transfers in Arm-based systems are:

- Read-modify-write of semaphore data
- Cortex-M3 and Cortex-M4 bit band operations

These use cases are not affected by this limitation because the locked sequence in these cases targets the same address and never crosses a cache line boundary.

When the AHB Cache receives a locked access that hits the cache, the access is stalled until the related cache line is cleaned and invalidated.

It can be the case that a locked sequence crosses a cache line boundary and the subsequent cache line is stored in the cache and dirty. In that situation, the locked sequence is broken at the boundary by the eviction Write-Back Burst and the AHB master interface deasserts **hmastlock**. The sequence is continued with a NONSEQ transaction after the eviction is finished.

hmastlock being deasserted can result in the locked sequence progressing further during the eviction Write-Back Burst without completing the previous access.

2.2.5 Error responses

The AHB Cache operates transparently when disabled, without any latency added.

The AHB Cache forwards the error response from the master interface to the slave interface for:

- Transfers that must be responded from the endpoint. For example, non-cacheable, non-bufferable, and non-modifiable transfers.
- Streaming hits during a linefill

To learn more about error responses to XOM reads, see *2.2.6 XOM* on page 2-27.

Linefill error and data loss

If a linefill receives an error response on the AHB Master interface, data from a Write-Back access can be lost.

If a linefill encounters a bus error, the data from the entire linefill Burst is invalidated. Any data written to this cache line before the bus error occurred is lost. The AHB Cache signals the linefill error through an interrupt, TR_ERR, which is generated by the transfer error.

Further errors are not reported for the given linefill Burst. If a subsequent Burst beat is responded to with an AHB error, it does not trigger the TR_ERR interrupt again. The AHB error is not captured in the IRQINFO registers, and does not overwrite the information saved earlier for the first error. Therefore regardless of the number of errors received during a linefill Burst, the AHB Cache reports only a single interrupt for the first error.

A linefill error does not affect read data.

Related concepts

3.1.4 Linefill on page 3-36

2.2.6 XOM

The AHB interface can be configured to support the configurable *eXecute Only Memory* (XOM) feature.

- Instruction read accesses to a memory location flagged as XOM proceed as normal.
- Data read accesses and writes to XOM regions are not allowed.

The AHB interface supports XOM using **hrxom_s** and **hrxom_m** sideband signals.

hrxom_s and **hrxom_m** follow the same timing as **hruser_s** and **hruser_m**.

The downstream AHB slave must assert the **hrxom_m** signal when a read access hits an XOM region.

The **hrxom_m** signal must be consistent throughout a cache line. The AHB Cache can detect that a memory region belongs to an XOM region when the **hrxom_m** signal is set on its AHB master interface. The XOM attribute is saved for each cache line. On read accesses, the **hrxom_s** signal is driven according to the previously saved XOM attribute or, for streaming purposes, by the **hrxom_m** input signal.

If a data access read (**hprot_s[0]**=1) hits an XOM line already in the cache, the read data is masked and the transfer is responded with an error or an OK response depending on the **ahb_violation_resp**. If a Write-Access hits an XOM line already in the cache, the line is invalidated and the transfer is always forwarded. When such XOM violations are detected by the AHB Cache, the AHB Cache sets the XOM error interrupt flag. For more information, see *4.4.9 SECIRQEN, Secure interrupt enable register* on page 4-62 and *4.4.14 NSECIRQEN, Non-secure interrupt enable register* on page 4-67.

2.2.7 Debug accesses

The AHB Cache supports debug accesses.

Debug accesses must be flagged using the **hdebug_s** input port. **hdebug _s** follows the same timing as **hmaster_s**.

Debug accesses are looked up in the cache, even if they are marked as non-cacheable. The lookup takes place in case the debugger has used the wrong attributes or the MPU overwrote them.

2.3 APB interface

The APB interface provides a software control and status interface for the AHB Cache.

It allows software to complete the following actions:

- Enable and disable the cache
- Configure Non-secure permissions
- Configure forceable Write-Through policy
- Read the statistics registers
- Perform manual maintenance operations
- Handle interrupt enables and interrupt statuses
- Setup and trigger snapshots, when SNAPSHOTTING is configured

Access rules

The APB interface only accepts an access that meets all the following conditions:

- The access is privileged.
- The access is a data access.
- The address is aligned.
- Writes have all strobe bits set.

An access that does not meet the preceding conditions is considered a failed APB access. The cache responds to failed APB accesses as follows:

- Read accesses return zero.
- Writes accesses are ignored.

If the **apb_violation_resp** is set to HIGH, the AHB Cache responds with errors to failed APB accesses by asserting **pslverr**. For more information, see the *AMBA® APB Protocol Specification Version 2.0, issue C*.

The APB interface is security aware. Each register and individual bits have security attributes as described in the *Chapter 4 Programmers model* on page 4-44.

Table 2-2 Secure and Non-secure accesses

Access Type	Permissions
Secure accesses	Secure accesses can access all registers and fields, including registers and fields marked as Non-secure.
Non-secure accesses	Non-secure accesses can only access Non-secure registers and fields. Attempts to access Secure information are ignored: *Reads As Zero, Writes Ignored* (RAZ/WI) or **pslverr** HIGH, depending on **apb_violation_resp**. No interrupts are triggered. **Note** You can configure some Non-secure access permissions using the CTRL register.

Configuring Non-secure access permissions

Only Secure accesses can configure the Non-secure access permission registers. You can configure the CTRL register so that Non-secure accesses are allowed to perform the following actions:

- Checking if the cache is enabled
- Reading Non-secure statistics registers
- Triggering some Non-secure line maintenance features

For more information, see *4.4.2 CTRL, control register* on page 4-54. Non-secure software can read the NSEC_ACCESS register to see which information and features it can access. For more information, see *4.4.3 NSEC_ACCESS, Non-secure access information register* on page 4-56.

2.4 Interrupts

The AHB Cache has two interrupt signals: one for Secure and one for Non-secure interrupt sources.

The AHB Cache can generate interrupts for the following events:

- Maintenance finished
- A maintenance request being ignored
- Interface errors on the AHB Master interface
- Saturation of the statistics counters.
- The AHB Cache being either enabled or disabled
- Access violations

Note

Access violations can be triggered by bus masters that are able to generate speculative accesses.

The following table lists the interrupt signals used by the AHB Cache.

Table 2-3 Interrupt signals

Signal	Description
sec_irq	Secure interrupt
nseq_irq	Non-secure interrupt

Each interrupt has a set of associated registers. If an interrupt source is not enabled, its status register is still set as normal, but it does not contribute to the interrupt signal. Two associated information registers help diagnose the source of the interrupt generated by a transfer error. For interrupt register descriptions, see from section *4.4.7 SECIRQSTAT, Secure interrupt request status register* on page 4-60 to section *4.4.16 NSECIRQINFO2, Non-secure transfer error information register 2* on page 4-69 of the *Chapter 4 Programmers model* on page 4-44.

Related references

A.6 System interface signals on page Appx-A-125

2.5 Low-Power Interface

The AHB Cache has two LPI Q-Channel interfaces to support low-power applications: one for clock and one for power management.

For a description of the signals that are used by the LPI Q-Channel, see *A.2 LPI signals* on page Appx-A-119.

This section contains the following subsections:

2.5.1 Dirty status indicator

The AHB Cache uses a simplified model to track the overall dirty status of the cache.

The simplified model does not track the actual dirty status of individual lines. Instead, it reports if there is potentially any dirty data in the cache, using the CACHE_IS_CLEAN bit in the MAINT_STATUS register. For more information, see *4.4.6 MAINT_STATUS, maintenance status for the cache register* on page 4-59.

Any write to an allocated or soon-to-be allocated cache line sets the dirty status indicator. Once set, this bit remains set and is only cleared by a completed clean all, invalidate all, or clean and invalidate all cache maintenance operation (automatic or manual).

Therefore the AHB Cache can report a dirty status, even if all lines are clean. If the AHB Cache reports a clean status, it guarantees that the cache is clean.

2.5.2 Clock LPI

The clock LPI module signals activity on the AHB Cache and responds to incoming quiescence requests.

The clock LPI module is active and denies quiescence requests in the following situations:

- There is activity on the AHB or APB interface.
- An internal operation (for example, maintenance) is ongoing.
- There is activity on the power Q-Channel.

Otherwise the quiescence request is accepted and the response is synchronized with **pcklen**.

Note

When the AHB Cache is in a clock quiescent state, it can asynchronously request **clk** for itself.

Related references

A.2 LPI signals on page Appx-A-119

2.5.3 Power LPI

The power LPI module responds to incoming quiescence requests.

The power LPI module is active and denies quiescence requests in the following situations:

- There is activity on the AHB or APB interfaces.
- An internal operation, for example, maintenance is ongoing.
- There is any outstanding interrupt.
- The DENY_POWERDOWN bit in the CTRL register is set.

Otherwise the request is accepted and the response is synchronized with **pcklen**.

Note

When the AHB Cache is in power and clock quiescent state, it cannot asynchronously request power for itself.

When the clock LPI module is in a quiescent state, the power LPI module does not respond to requests or update **pwr_qactive**. Instead, the power LPI module waits for the clock request to be granted.

By default, the AHB Cache runs typical maintenance tasks automatically, see *3.3.7 Automatic maintenance features* on page 3-39.

Note

By default, the AHB Cache automatically cleans all cache lines before accepting a quiescence request. However, it is possible to disable automatic cache maintenance through the configuration input, **dis_pwr_down_maint**. We recommend that the cache is cleaned before it enters any retention state, unless optimization requirements demand otherwise.

The AHB Cache does not monitor dirty cache lines for activity reporting.

When the AHB Cache receives a quiescence request, it checks the cache activity and status:
- If the cache status is clean and the cache otherwise idle, then it accepts the quiescence request.
- If the cache status is dirty but otherwise idle, it starts a clean all maintenance process. The AHB Cache delays the response and only denies the request if a dirty line is actually found in the cache memory during the maintenance operation. If no dirty line was found and the maintenance operation is completed, the AHB Cache accepts the request.

Quiescence requests are denied until the powerdown maintenance is completed. The AHB Cache aborts the powerdown maintenance process, if it receives new transfer on the AHB Slave interface.

Before requesting quiescence again, the software can check the MAINT_STATUS register to see if the cache is clean. For more information, see *4.4.6 MAINT_STATUS, maintenance status for the cache register* on page 4-59.

Related references

A.2 LPI signals on page Appx-A-119

2.5.4 Quiescent state

The quiescent state affects the way that the AHB Cache processes transactions on its AHB slave interface.

The AHB Cache is not intended to be on the power, reset, or clock domain border with its AHB or APB interfaces.

By default, the AHB Cache is not expected to receive a transaction on the AHB slave interface while in quiescent state.

If an access is received while the cache is in a quiescent state, the access is stalled while the AHB Cache is wakened. The AHB address phase cannot be stalled according to the *Arm® AMBA® 5 AHB Protocol Specification, issue B.b*. However, the Q-Channel handshakes might not have finished yet, while the AHB Cache might already have stable power and clock. Therefore, the AHB slave interface buffers the address phase of the access to avoid data loss if possible.

If the AHB Cache is powered off after the AHB slave interface has buffered the address phase of the access, the buffered address phase is lost. The transfer is completed with the isolation values of the data phase signals.

If the cache is not reset after entering quiescent state, but returned to functional state instead, then the cache enable status remains unchanged. This process takes place in, for example, retention states. The RAM content is still valid and the previously enabled cache continues to function.

Related references

A.2 LPI signals on page Appx-A-119

Chapter 3
Operation

This chapter describes the operation of the AHB Cache.

It contains the following sections:

3.1 Basic operations

This section describes some of the basic operations performed by the AHB Cache.

This section contains the following subsections:

3.1.1 Cache enable

The AHB Cache must be enabled to start caching accesses. You can enable the AHB Cache using either software or hardware.

Software can enable the cache by setting the ENABLE field of the CTRL register using the APB configuration interface. For more information about using software to enable the AHB Cache, see *5.1 Enable the AHB Cache by using software* on page 5-96.

Hardware can also enable the cache by setting the configuration port **power_on_enable**, which triggers the cache to enable when the LPI interfaces reach the QRUN state for the first time after reset.

If cache enable maintenance is turned on, the AHB Cache runs invalidate all maintenance in the background. When the maintenance is completed, the cache is enabled. For more information about cache enable maintenance, see *Cache enable maintenance* on page 3-41.

If cache enable maintenance is turned off by setting the configuration port **dis_cache_en_maint** and it is unclear whether the cache memory is still valid, the software must invalidate the cache before enabling it. For more information about invalidate all maintenance, see *3.3.4 Invalidate all maintenance* on page 3-39.

Related references

4.4.2 CTRL, control register on page 4-54

3.1.2 Cache disable

Software can disable the AHB Cache using the APB configuration interface.

For more information, see *5.2 Disable the AHB Cache using software* on page 5-100. If cache disable maintenance is turned on, the AHB Cache starts clean all maintenance. After the maintenance is completed, the AHB Cache is disabled. For more information about cache disable maintenance, see *Cache disable maintenance* on page 3-42.

If cache disable maintenance is turned off by setting the configuration port **dis_cache_dis_maint**, then the software is responsible for cleaning the cache before the cache is disabled to avoid data loss. For more information, see *3.3.2 Clean all maintenance* on page 3-38.

Related references

4.4.2 CTRL, control register on page 4-54

3.1.3 Lookup

The AHB Cache looks up accesses that are marked as cacheable.

The AHB Cache performs a lookup in the four tag RAMs and in the internal buffers. The result of a lookup can be either hit or miss.

Table 3-1 Cache hit and cache miss

Lookup result	Description
Hit	A cache hit means that the data is already in the cache or being filled. The AHB Cache responds to the access without added latency, unless the destination cache line is being filled and the given word has not yet been received.
Miss	A cache miss means that the data is not in the cache: • Allocate miss accesses trigger a linefill. • No-allocate miss accesses are forwarded. Cache miss accesses might affect latency. For more information, see *Table 2-1 Normal access latencies* on page 2-25.

3.1.4 Linefill

A linefill is a read Burst transaction that takes place when a new cache line must be brought into the cache from the external memory.

The linefill results in an entire 32-byte cache line being stored in the internal buffers where it can be looked up. If the target cache set is full, the cache evicts a cache line.

Related concepts

Linefill error and data loss on page 2-27

3.1.5 Eviction

The eviction process creates space for new data in the AHB Cache.

Cacheable accesses, which have addresses with the same index, can be stored in one of the four lines of the corresponding set.

The eviction process takes place when the corresponding set is full and a new line needs to be cached with the same index.

One of the previously stored cache lines is randomly selected and evicted to make room for a new line. After eviction, the cache line is invalidated in the cache memory. If the evicted cache line was dirty, it is also written back to main memory after eviction.

3.2 Performance monitoring

The AHB Cache provides statistical counters to record cache hits and misses. Non-cacheable accesses and debug transactions are not counted.

The statistical counters are available in every configuration. Four counters are available for performance monitoring. Secure and Non-secure hits and misses are counted separately on dedicated counters.

Secure software

By default, the counters are only visible to Secure software.

Non-secure software

Secure software can enable Non-secure software to access the Non-secure counters. For more information, see *4.4.2 CTRL, control register* on page 4-54. However, Non-secure software cannot access the Secure counters.

3.2.1 Snapshotting

The AHB Cache can support snapshots, which are enabled by a hardware configuration, 'SNAPSHOTTING' in the HWPARAMS register. By using the snapshotting functionality, you can capture all four statistical counters into capture registers at the same time.

When enabled, two triggers are provided:

- A hardware trigger as an input port
- A software trigger as a register bit

The AHB Cache uses the **pmsnapshotreq** input port, which serves as a hardware trigger, to ensure that gathered statistics are captured at the same time across multiple components.

Note

If there is power and the clock is running, the **pmsnapshotreq** sample port is sampled and snapshots are taken, even when the cache is in clock and power quiescence.

Table 3-2 Snapshotting registers

Name	Summary	Description
PMSSCR	Software can trigger a snapshot by writing 1 to this register.	*4.4.28 PMSSCR, PMU snapshot capture register* on page 4-81
PMSSSR	The PMSSSR register allows software to check that a capture occurred. The value 1 means no capture has occurred. The value is set to 0 by hardware or software trigger when a snapshot is taken. The value is cleared to 1 by reset only.	*4.4.27 PMSSSR, PMU snapshot status register* on page 4-80
PMSSRR	Setting the PMSSRR register makes each snapshot automatically reset the counters on capture, so that the next snapshot does not contain data from an already captured interval.	*4.4.29 PMSSRR, PMU snapshot reset register* on page 4-82

For more information, see the *Arm® CoreLink™ AHB Cache Configuration and Integration Manual*.

3.3 Maintenance

The AHB Cache receives requests on the APB interface that can either directly or indirectly trigger a maintenance operation. The APB interface responds to those requests using the APB status registers and interrupts.

Quiescence requests received through the Q-Channel interface can also trigger a maintenance operation. The AHB Cache responds to these requests depending on whether there is any maintenance already ongoing. The AHB Cache also has a dedicated output port, **pwr_maintenance**, to track the status of powerdown maintenance.

The AHB Cache initiates automatic maintenance activities, see *3.3.7 Automatic maintenance features* on page 3-39. Software can initiate manual maintenance, see *3.3.8 Manual maintenance* on page 3-43.

All maintenance delays subsequent operations on the cache until the maintenance is complete.

The AHB Cache performs the following maintenance:
- Clean by address
- Clean all
- Invalidate by address
- Invalidate all
- Clean and invalidate by address
- Clean and invalidate all

This section contains the following subsections:

3.3.1 Clean by address maintenance

Clean by address maintenance performs a lookup for a cache line with a selected address.

If the lookup results in a hit and the cache line is dirty, then the AHB Cache writes it back to main memory. The dirty status of the cache line is cleared in the process.

3.3.2 Clean all maintenance

Clean all maintenance walks through all cache lines and writes back all dirty cache lines.

The dirty status of each line is cleared during the clean all maintenance process. After clean all maintenance is completed, the cache status is set to clean. For more information, see *4.4.6 MAINT_STATUS, maintenance status for the cache register* on page 4-59.

Related concepts

Powerdown maintenance on page 3-41

Cache disable maintenance on page 3-42

3.3.3 Invalidate by address maintenance

Invalidate by address maintenance performs a lookup of a cache line with a specific address. If the lookup results in a hit, then the AHB Cache invalidates the selected cache line.

Caution

Only Secure software can perform invalidate by address maintenance. Invalidating a dirty cache line can cause data loss.

3.3.4 Invalidate all maintenance

Invalidate all maintenance walks through all cache lines and invalidates them.

Caution

Only Secure software can perform invalidate all maintenance. Invalidating a dirty cache line can cause data loss.

Invalidate all maintenance usually delays subsequent operations on the cache until the maintenance is complete. However, when the cache is being enabled, automatic invalidate all maintenance does not stall the AHB interface.

Invalidate all is the only type of maintenance that is allowed when the AHB Cache is disabled. If invalidate all takes place when the cache is disabled, it also does not stall the AHB interface.

Related concepts

Cache enable maintenance on page 3-41

3.3.5 Clean and invalidate by address maintenance

The APB interface can initiate clean and invalidate maintenance by address.

Clean and invalidate by address maintenance combines the *3.3.1 Clean by address maintenance* on page 3-38 and *3.3.3 Invalidate by address maintenance* on page 3-38. The address is looked up in the cache. Depending on the result of the lookup, the AHB Cache performs a corresponding action:

- If the lookup results in a hit and the address is dirty, then the address is cleaned, and then invalidated.
- If the lookup results in a hit and the address is clean, the address is invalidated.
- If the lookup results in a miss, the maintenance is complete and no further action is taken.

3.3.6 Clean and invalidate all maintenance

The APB interface can initiate clean and invalidate all maintenance for the entire cache.

Note

Only Secure software can perform clean and invalidate all maintenance. Invalidating a dirty cache line can cause data loss.

Clean and invalidate all maintenance combines *3.3.2 Clean all maintenance* on page 3-38 and *3.3.4 Invalidate all maintenance* on page 3-39. The maintenance cycles through all the lines in the cache. Depending on the cache line status, the AHB Cache performs a corresponding action:

- If the line is valid and dirty, then it is cleaned, and then invalidated.
- If the line is valid and clean, it is invalidated.

3.3.7 Automatic maintenance features

Automatic maintenance operations are triggered by default in some situations.

Automatic maintenance operations do not generate a MAINT_DONE interrupt, but they do generate error interrupts if errors occur.

Configurable automatic maintenance

Configurable automatic maintenance is triggered by default, when one of the following conditions is met:

- The AHB Cache is enabled or disabled.
- There is a quiescence request.

The configurable automatic maintenance processes are described in the following table.

Table 3-3 Configurable automatic maintenance

Trigger	Maintenance	Condition
Quiescence request (while the AHB Cache is enabled and idle)	Clean all	When the **dis_pwr_down_maint** signal is not asserted.
Enabling the cache	Invalidate all	When the **dis_cache_en_maint** signal is not asserted.
Disabling the cache	Clean all	When the **dis_cache_dis_maint** signal is not asserted.

You can change the default automatic maintenance settings using *Maintenance configuration input ports* on page 3-40.

Automatic maintenance to maintain consistency

Automatic maintenance is triggered to maintain consistency in the following situations:

- An AHB locked access hits the cache.
- An AHB Write-Access hits an XOM line in the AHB Cache.
- An AHB exclusive access hits the cache.

The following table describes automatic maintenance triggered for consistency.

Table 3-4 Automatic maintenance to maintain consistency

Trigger	Maintenance	Condition
An AHB exclusive access hits the AHB Cache.	Clean and invalidate by address	-
An AHB Write-Access hits an XOM line in the AHB Cache.	Invalidate by address (an XOM line cannot be dirty)	When XOM support is configured
An AHB locked access hits the AHB Cache.	Clean and invalidate by address	-

Maintenance configuration input ports

To change the default automatic maintenance settings, use the configuration input ports.

Table 3-5 Maintenance configuration input ports

Input port	Maintenance process	Description
dis_pwr_down_maint	*Powerdown maintenance* on page 3-41	Configuration input port to disable automatic clean at a power Q-Channel quiescence request. Sampled when the AHB Cache is preparing for powerdown maintenance. This signal must be stable until the cache has finished preparing for powerdown. The cache indicates it has finished preparing for powerdown by deasserting the **pwr_maintenance** signal.
dis_cache_en_maint	*Cache enable maintenance* on page 3-41	Configuration input port to disable automatic maintenance (invalidate all) at enabling the cache. Sampled while the AHB Cache is preparing for cache enable maintenance. This signal must be stable until the AHB Cache is enabled.
dis_cache_dis_maint	*Cache disable maintenance* on page 3-42	Configuration input port to disable automatic maintenance (clean all) at disabling the cache. Sampled when the AHB Cache is preparing for cache disable maintenance. This signal must be stable until the AHB Cache is disabled.

Note

Software can check whether the automatic maintenance is enabled or disabled by reading the corresponding bits in the HWPARAMS register. For more information, see *4.4.1 HWPARAMS, hardware parameter register* on page 4-52.

Powerdown maintenance

Powerdown maintenance takes place whenever a quiescence request is received through the power Q-Channel interface.

When the **dis_pwr_down_maint** input port is asserted, a power Q-Channel quiescence request does not trigger an automatic clean. You can use this configuration when the cache is not to be powered down or the cache memory contents are preserved through retention.

The **pwr_maintenance** output status port is asserted while the powerdown maintenance is ongoing. While powerdown maintenance is disabled, this status port might still be asserted for a few clock cycles while preparing internally for powerdown.

Powerdown maintenance on

The AHB Cache checks if the cache is clean whenever a quiescence request is received through the power Q-Channel interface. For more information, see *2.5.1 Dirty status indicator* on page 2-31.

If the cache status is clean and otherwise idle, then it accepts the quiescence request.

If the cache status is dirty but otherwise idle, it starts a clean all maintenance process.

The AHB Cache delays the response and only denies the request if a dirty line is actually found in the cache memory during the maintenance operation. If no dirty line was found and the maintenance operation is completed, the AHB Cache accepts the request.

While the maintenance is in progress, the cache reports activity on **pwr_qactive** signal.

Any activity on the AHB Slave interface aborts this type of maintenance.

When the clean is completed, the module stops reporting activity on the **pwr_qactive** signal, so a consecutive quiescence request is accepted. If a Write-Back access makes the cache dirty again, then another clean all maintenance starts at the next quiescence request.

Powerdown maintenance off

The AHB Cache does not check if the cache is clean but accepts the quiescence request if the cache is idle.

Caution

If automatic maintenance is turned off, care must be taken to avoid data loss in Write-Back memory regions, when powering down the AHB Cache.

Related concepts

3.3.2 Clean all maintenance on page 3-38

Cache enable maintenance

When cache enable maintenance is on, cache enable maintenance is triggered automatically when the cache is enabled.

For more information, see *3.1.1 Cache enable* on page 3-35.

Cache enable maintenance on

When an enable command is received, the AHB Cache starts a sequence similar to *3.3.4 Invalidate all maintenance* on page 3-39.

Traffic is not stalled on the AHB Slave interface, since caching is disabled and therefore no lookup would use the RAM interfaces. While the invalidation is in progress, the cache stays disabled and forwards all transactions. This process takes place so that the AHB Cache RAM is initialized to a known empty state. When the invalidation is completed, caching is enabled and the next cacheable nonsequential transaction is looked up.

Note

If the configuration port **power_on_enable** is set, cache enable maintenance can also be triggered by hardware. For more information, see *A.8 Configuration input ports* on page Appx-A-128.

Cache enable maintenance off

When cache enable maintenance is off and the cache is enabled, the AHB Cache immediately looks up the next nonsequential transaction, without regard for the actual content of the cache memory. If the cache memory contents are not valid, the related data is corrupted.

Caution

When enabling the AHB Cache with cache enable maintenance turned off, you must take care to avoid memory corruption.

Related concepts

3.3.4 Invalidate all maintenance on page 3-39

Cache disable maintenance

When cache disable maintenance is on, cache disable maintenance is triggered automatically when the cache is disabled.

For more information, see *3.1.2 Cache disable* on page 3-35.

Cache disable maintenance on

When the AHB Cache is disabled, it starts clean all maintenance when a disable command is received through the APB software programming interface. Before starting the clean maintenance, all traffic to the AHB Slave interface is stalled for the next nonsequential transaction. While the clean is in progress, the AHB Slave interface remains stalled. When the clean is completed, then the cache is disabled, the AHB Slave interface is released and upcoming transactions are forwarded.

Cache disable maintenance off

When the cache disable maintenance is off, the cache disable command immediately disables the cache. The AHB Cache does not look up the next nonsequential transaction.

Caution

When disabling the AHB Cache with cache disable maintenance turned off, you must make sure that there is no dirty data in the cache memory. If the dirty data is not cleaned and the cache is disabled, then the dirty data is no longer visible to the system.

We do not expect the software to send cacheable write transactions while trying to disable the cache with manual maintenance. However, when cache disable automatic maintenance is turned off, we recommend that the cache disable command follows a clean all command. We recommend that the cache disable command is issued without waiting for the maintenance to complete or generate an interrupt. If the cache disable command is received while the clean all maintenance is still running, then the cache executes the disable command before it would service any possible pending transactions.

Related concepts

3.3.2 Clean all maintenance on page 3-38

3.3.8 Manual maintenance

Software can initiate manual maintenance activities including clean and invalidate.

Secure manual maintenance

Secure software can manually start any all-cache or by-address maintenance activities in the AHB Cache. It can also enable Non-secure maintenance.

Non-secure manual maintenance

Secure software can enable Non-secure maintenance.

Non-secure maintenance allows Non-secure software to start maintenance on Non-secure addresses. Non-secure software cannot start maintenance on Secure addresses, and it cannot perform any all-cache maintenance. It also cannot start invalidate-only maintenance.

3.3.9 power_on_enable

You can use the **power_on_enable** input configuration port to allow hardware to enable the AHB Cache.

The **power_on_enable** port is sampled when the cache is directed to running state on its power LPI Q-Channel interface at the first time after reset. When triggered, this feature starts the normal cache enable process. For more information, see *4.4.1 HWPARAMS, hardware parameter register* on page 4-52.

If enable maintenance is disabled through the **dis_cache_en_maint** configuration port, then the related maintenance is not performed and the cache is enabled immediately.

Subsequent interface transitions have no effect on **power_on_enable** until the cache is reset.

Forcing Write-Through can be affected by **power_on_enable**. For more information, see *4.4.2 CTRL, control register* on page 4-54.

Chapter 4
Programmers model

This chapter describes the functionality of the AHB Cache from a programming perspective.

It contains the following sections:

4.1 About the programmers model

This section describes the functions and programmers model of the AHB Cache.

When using the programmers model, adhere to the following guidelines:

- 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.
  - — Unless otherwise specified, all register bits are reset to a logic 0 by a system or power up reset.

The following describes the access type:

RW Read and write.

RO Read-only.

WO Write-only.

RAZ Read as zero.

WI Writes ignored.

4.2 Programming considerations

To avoid data leaks or corruption when programming the AHB Cache, adhere to the following guidance.

Caution

When the memory map needs to be changed (for example, the *Security Attribution Unit* (SAU) is reconfigured), adhere to the following guidance:

- If the cache content is dirty, it should be cleaned before the memory map is changed.
- The cache should stay disabled while the change is in progress.
- After changing the memory map, the cache should be invalidated before being enabled, as it would be during automatic cache enable maintenance.

Note

The AHB Cache requires memory attribute granularity or XOM granularity not to be smaller than a 32-byte cache line. Bursts should not cross between regions of different types.

The AHB Cache allows both the Secure and Non-secure view of an address to be allocated in the cache simultaneously.

Caution

EXEMPT memory regions are not security checked, therefore software security is applied to these regions. If an EXEMPT region is defined as cacheable, Software security for EXEMPT memory regions can cause the following issues:

- Data changes made by Secure code might remain hidden from Non-secure code.
- Data changes made by Non-secure code might remain hidden from the Secure code.

Caution

If the system has a Security Extension and the Secure and Non-secure code use any shared memory regions, the Secure and Non-secure code must be set to use the same memory attributes.

Caution

When the AHB Cache brings in a Non-secure line from external memory, it can evict a Secure line stored in the cache. Likewise, a Secure line brought in from external memory can evict a Non-secure line stored in the cache.

You must evaluate whether Non-secure code or data evicting Secure code or data can cause Secure state leakage in your system. Based on your evaluation you must decide whether you want to mitigate against Secure state leakage: for example, by setting the affected Secure memory region as Non-cacheable.

4.3 Register summary

This section provides a summary of the AHB Cache register map.

The AHB Cache register map is divided into sections.

Table 4-1 AHB Cache register sections

Offset	Section
`0x000`, `0x010-0x014`, `0x020-0x028`	General configuration and status
`0x100-0x110`, `0x140-0x150`	Interrupts
`0x300-0x308`, `0x310-0x318`	Performance counters
`0x600-0x6F4`	Performance monitor snapshotting
`0xFD0-0xFFC`	Product identification registers

Note

The performance monitor snapshotting registers are only present if configured before rendering, otherwise the region is reserved and RAZ/WI.

Note

Locations that are not listed in the table are Reserved.

The following table shows the registers in offset order from the base memory adddress.

Table 4-2 cg095 - APB4_Slave register summary

Offset	Name	Type	Security	Reset	Width	Description
`0x000`	HWPARAMS	RO	Secure	Configuration-dependent	32	*4.4.1 HWPARAMS, hardware parameter register* on page 4-52
`0x010`	CTRL	RW	Secure	`0x0`	32	*4.4.2 CTRL, control register* on page 4-54
`0x014`	NSEC_ACCESS	RO	Non-secure	`0x0`	32	*4.4.3 NSEC_ACCESS, Non-secure access information register* on page 4-56
`0x020`	MAINT_CTRL_ALL	WO	Secure	`0x0`	32	*4.4.4 MAINT_CTRL_ALL, maintenance control for the entire cache register* on page 4-57
`0x024`	MAINT_CTRL_LINES	WO	Software-configurable	`0x0`	32	*4.4.5 MAINT_CTRL_LINES, maintenance control for individual lines register* on page 4-58

Table 4-2 cg095 - APB4_Slave register summary (continued)

Offset	Name	Type	Security	Reset	Width	Description
`0x028`	MAINT_STATUS	RO	Software-configurable	`0x0`	32	*4.4.6 MAINT_STATUS, maintenance status for the cache register* on page 4-59
`0x100`	SECIRQSTAT	RO	Secure	`0x0`	32	*4.4.7 SECIRQSTAT, Secure interrupt request status register* on page 4-60
`0x104`	SECIRQSCLR	WO	Secure	`0x0`	32	*4.4.8 SECIRQSCLR, Secure interrupt status clear register* on page 4-61
`0x108`	SECIRQEN	RW	Secure	`0x0`	32	*4.4.9 SECIRQEN, Secure interrupt enable register* on page 4-62
`0x10C`	SECIRQINFO1	RO	Secure	`0x0`	32	*4.4.10 SECIRQINFO1, Secure transfer error information register 1* on page 4-63
`0x110`	SECIRQINFO2	RO	Secure	`0x0`	32	*4.4.11 SECIRQINFO2, Secure transfer error information register 2* on page 4-64
`0x140`	NSECIRQSTAT	RO	Non-secure	`0x0`	32	*4.4.12 NSECIRQSTAT, Non-secure interrupt request status register* on page 4-65
`0x144`	NSECIRQSCLR	WO	Non-secure	`0x0`	32	*4.4.13 NSECIRQSCLR, Non-secure interrupt status clear register* on page 4-66
`0x148`	NSECIRQEN	RW	Non-secure	`0x0`	32	*4.4.14 NSECIRQEN, Non-secure interrupt enable register* on page 4-67
`0x14C`	NSECIRQINFO1	RO	Non-secure	`0x0`	32	*4.4.15 NSECIRQINFO1, Non-secure transfer error information register 1* on page 4-68
`0x150`	NSECIRQINFO2	RO	Non-secure	`0x0`	32	*4.4.16 NSECIRQINFO2, Non-secure transfer error information register 2* on page 4-69
`0x300`	SECHIT	RO	Secure	`0x0`	32	*4.4.17 SECHIT, Secure transfers hit register* on page 4-70
`0x304`	SECMISS	RO	Secure	`0x0`	32	*4.4.18 SECMISS, Secure transfers miss register* on page 4-71
`0x308`	SECSTATCTRL	RW	Secure	`0x0`	32	*4.4.19 SECSTATCTRL, Secure transfers statistic counters control* on page 4-72
`0x310`	NSECHIT	RO	Software-configurable	`0x0`	32	*4.4.20 NSECHIT, Non-secure transfers hit register* on page 4-73
`0x314`	NSECMISS	RO	Software-configurable	`0x0`	32	*4.4.21 NSECMISS, Non-secure transfers miss register* on page 4-74
`0x318`	NSECSTATCTRL	RW	Software-configurable	`0x0`	32	*4.4.22 NSECSTATCTRL, Non-secure transfers statistic counters control register* on page 4-75

Table 4-2 cg095 - APB4_Slave register summary (continued)

Offset	Name	Type	Security	Reset	Width	Description
0x600	PMSVR0	RO	Secure	0x0	32	*4.4.23 PMSVR0, saved value register 0 - Secure hit* on page 4-76
0x604	PMSVR1	RO	Secure	0x0	32	*4.4.24 PMSVR1, saved value register 1 - Secure miss* on page 4-77
0x608	PMSVR2	RO	Software-configurable	0x0	32	*4.4.25 PMSVR2, saved value register 2 - Non-secure hit* on page 4-78
0x60C	PMSVR3	RO	Software-configurable	0x0	32	*4.4.26 PMSVR3, saved value register 3 - Non-secure miss* on page 4-79
0x680	PMSSSR	RO	Software-configurable	0x1	32	*4.4.27 PMSSSR, PMU snapshot status register* on page 4-80
0x6F0	PMSSCR	WO	Software-configurable	0x0	32	*4.4.28 PMSSCR, PMU snapshot capture register* on page 4-81
0x6F4	PMSSRR	RW	Software-Configurable	0x0	32	*4.4.29 PMSSRR, PMU snapshot reset register* on page 4-82
0xFD0	PIDR4	RO	Non-secure	0x4	32	*4.4.30 PIDR4, peripheral ID register 4* on page 4-83
0xFD4	PIDR5	RO	Non-secure	0x0	32	*4.4.31 PIDR5, peripheral ID register 5* on page 4-84
0xFD8	PIDR6	RO	Non-secure	0x0	32	*4.4.32 PIDR6, peripheral ID register 6* on page 4-85
0xFDC	PIDR7	RO	Non-secure	0x0	32	*4.4.33 PIDR7, peripheral ID register 7* on page 4-86
0xFE0	PIDR0	RO	Non-secure	0x31	32	*4.4.34 PIDR0, peripheral ID register 0* on page 4-87
0xFE4	PIDR1	RO	Non-secure	0xB8	32	*4.4.35 PIDR1, peripheral ID register 1* on page 4-88
0xFE8	PIDR2	RO	Non-secure	0xB	32	*4.4.36 PIDR2, peripheral ID register 2* on page 4-89
0xFEC	PIDR3	RO	Non-secure	0x0	32	*4.4.37 PIDR3, peripheral ID register 3* on page 4-90
0xFF0	CIDR0	RO	Non-secure	0xD	32	*4.4.38 CIDR0, component ID register 0* on page 4-91
0xFF4	CIDR1	RO	Non-secure	0xF0	32	*4.4.39 CIDR1, component ID register 1* on page 4-92

Table 4-2 cg095 - APB4_Slave register summary (continued)

Offset	Name	Type	Security	Reset	Width	Description
`0xFF8`	CIDR2	RO	Non-secure	`0x5`	32	*4.4.40 CIDR2, component ID register 2* on page 4-93
`0xFFC`	CIDR3	RO	Non-secure	`0xB1`	32	*4.4.41 CIDR3, component ID register 3* on page 4-94

4.4 Register descriptions

This section describes the AHB Cache registers.

4.3 Register summary on page 4-47 provides cross references to individual registers.

4.4.1 HWPARAMS, hardware parameter register

The HWPARAMS register allows the software to check the implementation options of the AHB Cache.

The HWPARAMS register characteristics are:

Attributes

Offset `0x0000`

Type Read-only

Reset The reset value is configuration-dependent.

Width 32

The following figure shows the bit assignments.

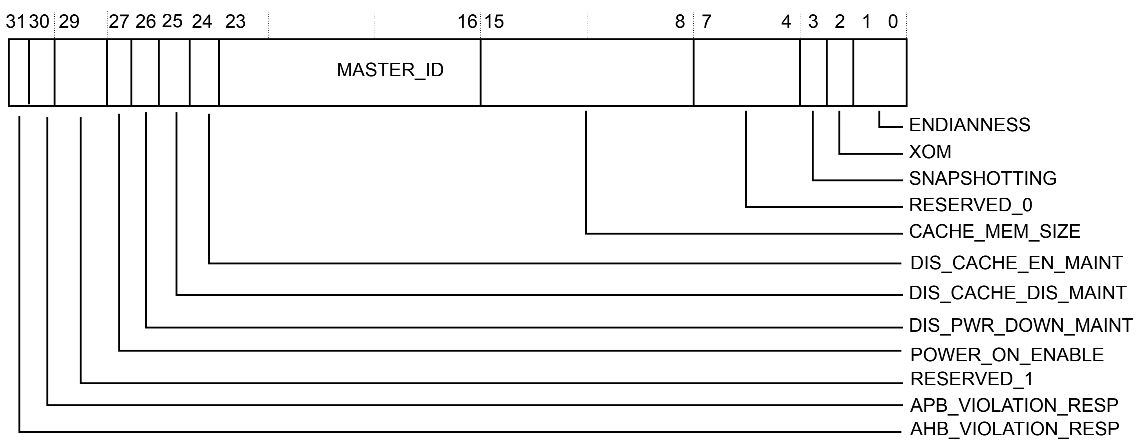

Figure 4-1 HWPARAMS register bit assignments

The following table shows the bit assignments.

Table 4-3 HWPARAMS register bit assignments

Bits	Name	Security	Function
[31]	AHB_VIOLATION_RESP	Secure	Respond with error (1) or RAZ/WI (0) to illegal AHB operations on XOM. Fixed to 0 if the XOM render parameter is OFF.
[30]	APB_VIOLATION_RESP	Secure	Respond with error (1) or RAZ/WI (0) to illegal APB operations. Illegal operations include transfers that are any of: non-privileged, instruction, unaligned or incomplete write strobes. The error response takes place only for a cacheable transfer.
[29:28]	RESERVED_1	Non-secure	Read-As-Zero, Writes Ignored.
[27]	POWER_ON_ENABLE	Secure	Value of the input configuration port that controls the function that enables the cache automatically after powerup.
[26]	DIS_PWR_DOWN_MAINT	Secure	Value of the input configuration port that controls the function to turn off powerdown maintenance.
[25]	DIS_CACHE_DIS_MAINT	Secure	Value of the input configuration port that controls the function to turn off cache disable automatic maintenance.
[24]	DIS_CACHE_EN_MAINT	Secure	Value of the input configuration port that controls the function to turn off cache enable automatic maintenance.
[23:16]	MASTER_ID	Secure	The cache generates transactions with this ID. For more information, see *A.4 AHB Master interface signals* on page Appx-A-122.

Table 4-3 HWPARAMS register bit assignments (continued)

Bits	Name	Security	Function
[15:8]	CACHE_MEM_SIZE	Secure	Cache memory size in address bits. The actual size is this value to the power of 2. 11 = 2KB 12 = 4KB 13 = 8KB 14 = 16KB 15 = 32KB 16 = 64KB
[7:4]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[3]	SNAPSHOTTING	Secure	SNAPSHOTTING support. 1 = ON 0 = OFF
[2]	XOM	Secure	XOM support. 1 = ON 0 = OFF
[1:0]	ENDIANNESS	Secure	The endianness of the module. 0 = LE 1 = BE8 2 = BE32

Related concepts

Maintenance configuration input ports on page 3-40
Powerdown maintenance on page 3-41
Cache enable maintenance on page 3-41
Cache disable maintenance on page 3-42
3.3.9 power_on_enable on page 3-43

4.4.2 CTRL, control register

The CTRL register allows the software to turn the cache off or on, and to configure it.

The CTRL register characteristics are:

Attributes

Offset `0x0010`

Type Read-write

Reset `0x0`

Width 32

The following figure shows the bit assignments.

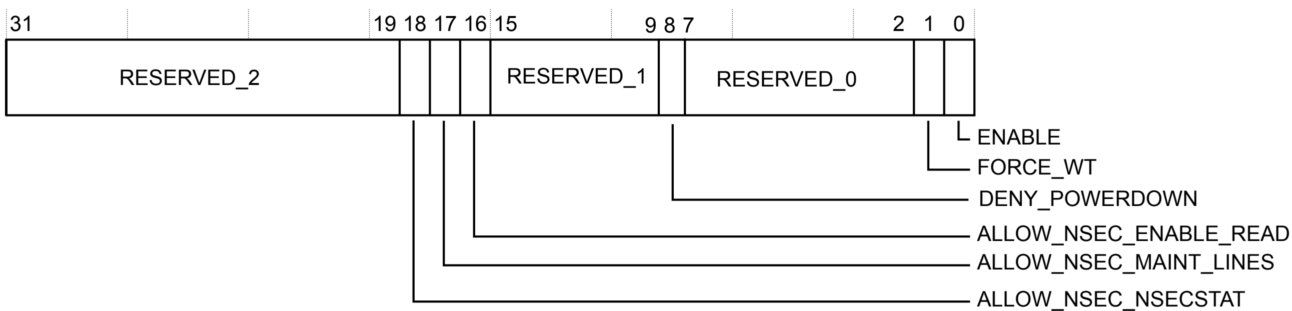

Figure 4-2 CTRL register bit assignments

The following table shows the bit assignments.

Table 4-4 CTRL register bit assignments

Bits	Name	Security	Function
[31:19]	RESERVED_2	Non-secure	Read-As-Zero, Writes Ignored.
[18]	ALLOW_NSEC_NSECSTAT	Secure	Allow Non-secure software to read and control Non-secure statistics counter registers and receive saturation interrupt.
[17]	ALLOW_NSEC_MAINT_LINES	Secure	Allow Non-secure software to trigger maintenance (only for lines and only Non-secure views of cache lines).
[16]	ALLOW_NSEC_ENABLE_READ	Secure	Allow Non-secure software to see if the cache is enabled.
[15:9]	RESERVED_1	Non-secure	Read-As-Zero, Writes Ignored.
[8]	DENY_POWERDOWN	Secure	When set, powerdown LPI requests are denied. Does not affect clock LPI requests.
[7:2]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.

Table 4-4 CTRL register bit assignments (continued)

Bits	Name	Security	Function
[1]	FORCE_WT	Secure	Forces Write-Through policy. **Note** Enabling FORCE_WT can be blocked when POWER_ON_ENABLE is set, as caching is already enabled by the time Force Write-Through would be activated from software. If you are using POWER_ON_ENABLE, you can force Write-Through by using software to complete the following steps. 1. Disable the cache. 2. Enable FORCE_WT. 3. Re-enable the cache.
[0]	ENABLE	Secure	Request to enable or disable cache. 1 = Enabled 0 = Disable. Enabling causes the AHB Cache to invalidate the cache memory unless DIS_CACHE_EN_MAINT is set. Disabling causes a clean of all cache lines, unless DIS_CACHE_DIS_MAINT is set. If another maintenance or enable or disable is in progress, the read value of ENABLE shows the requested state of the cache, not the current effective internal state which is shown in the MAINT_STATUS register. If the POWER_ON_ENABLE port is asserted, the enable request triggers automatically when the AHB Cache leaves Power down mode.

Related concepts

3.1.1 Cache enable on page 3-35
3.1.2 Cache disable on page 3-35

4.4.3 NSEC_ACCESS, Non-secure access information register

The NSEC_ACCESS register allows Non-secure software to check its access level and to see if the AHB Cache is enabled.

The NSEC_ACCESS register characteristics are:

Attributes

Offset `0x0014`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

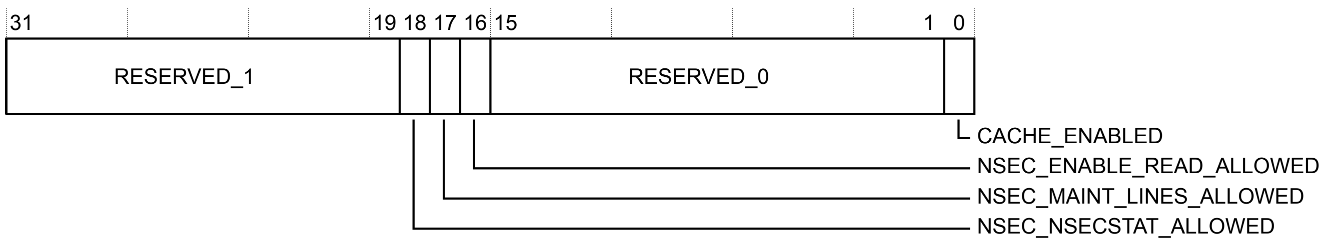

Figure 4-3 NSEC_ACCESS register bit assignments

The following table shows the bit assignments.

Table 4-5 NSEC_ACCESS register bit assignments

Bits	Name	Security	Function
[31:19]	RESERVED_1	Non-secure	Read-As-Zero, Writes Ignored.
[18]	NSEC_NSECSTAT_ALLOWED	Non-secure	Non-secure software is allowed to read and control Non-secure statistics counters and receives saturation interrupt.
[17]	NSEC_MAINT_LINES_ALLOWED	Non-secure	Non-secure software is allowed to trigger maintenance (only for lines).
[16]	NSEC_ENABLE_READ_ALLOWED	Non-secure	Non-secure software is allowed to see the cache enabled state.
[15:1]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[0]	CACHE_ENABLED	Software-configurable	Shows if the cache is enabled or disabled. 1 = Enabled 0 = Disabled If the NSEC_ENABLE_READ_ALLOWED bit is not set, then CACHE_ENABLED is masked for Non-secure reads.

4.4.4 MAINT_CTRL_ALL, maintenance control for the entire cache register

The MAINT_CTRL_ALL register is used to trigger maintenance operations on the entire cache.

For more information, see *3.3 Maintenance* on page 3-38. The MAINT_CTRL_ALL register characteristics are:

Attributes

Offset `0x0020`

Type Write-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

Figure 4-4 MAINT_CTRL_ALL register bit assignments

The following table shows the bit assignments.

Table 4-6 MAINT_CTRL_ALL register bit assignments

Bits	Name	Security	Function
[31:2]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[1]	TRIG_INVALIDATE_ALL	Secure	Trigger invalidate all maintenance. It can be used together with TRIG_CLEAN_ALL. TRIG_INVALIDATE_ALL can be used even if the cache is not enabled, but only if used without clean.
[0]	TRIG_CLEAN_ALL	Secure	Trigger clean all maintenance. This can be used together with TRIG_INVALIDATE_ALL.

4.4.5 MAINT_CTRL_LINES, maintenance control for individual lines register

The MAINT_CTRL_LINES register is used to trigger maintenance operations for a specific address.

For more information, see *3.3 Maintenance* on page 3-38. The MAINT_CTRL_LINES register characteristics are:

Attributes

Offset `0x0024`

Type Write-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

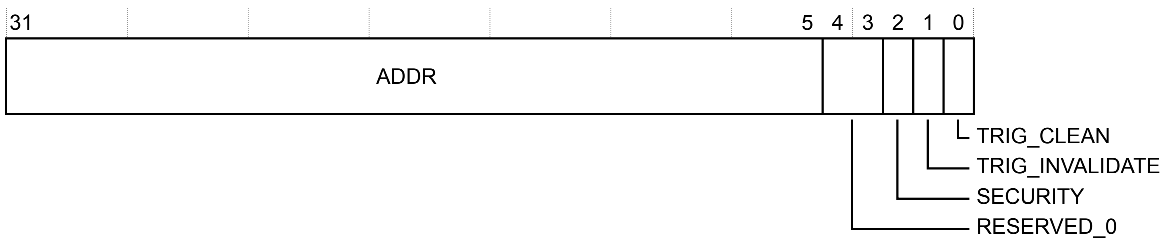

Figure 4-5 MAINT_CTRL_LINES register bit assignments

The following table shows the bit assignments.

Table 4-7 MAINT_CTRL_LINES register bit assignments

Bits	Name	Security	Function
[31:5]	ADDR	Software-configurable	Address to look up in the cache and perform invalidate or cleaning on matching cache line. Use bits [31:5] of the address.
[4:3]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[2]	SECURITY	Secure	Cache maintenance is performed on the Secure or Non-secure view of the address. It is possible to have both the Secure and Non-secure views of the same address allocated in the cache. 0 = Secure 1 = Non-secure For Non-secure accesses, the Secure view is ignored, and the Non-secure view is always selected.
[1]	TRIG_INVALIDATE	Software-configurable	Trigger invalidate by address on the addressed cache line. It can be used together with TRIG_CLEAN.
[0]	TRIG_CLEAN	Software-configurable	Trigger clean by address on the addressed cache line.

4.4.6 MAINT_STATUS, maintenance status for the cache register

Reading the MAINT_STATUS register returns if any maintenance is already in progress on the AHB Cache. The software can check the value of the register before attempting to issue a maintenance operation. Otherwise the write operation is stalled until new maintenance can be started.

The MAINT_STATUS register characteristics are:

Attributes

Offset `0x0028`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

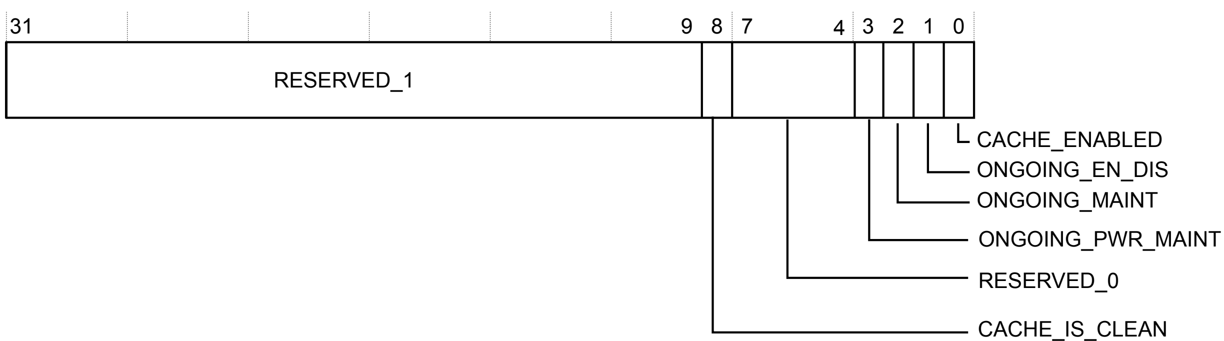

Figure 4-6 MAINT_STATUS register bit assignments

The following table shows the bit assignments.

Table 4-8 MAINT_STATUS register bit assignments

Bits	Name	Security	Function
[31:9]	RESERVED_1	Non-secure	Read-As-Zero, Writes Ignored.
[8]	CACHE_IS_CLEAN	Software-configurable	Reading 1 means that the cache has no dirty data. The AHB Cache uses a simplified model to check for dirty data. For more information, see *2.5.1 Dirty status indicator* on page 2-31.
[7:4]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[3]	ONGOING_PWR_MAINT	Software-configurable	Reading 1 means low-power request automatic maintenance is in progress.
[2]	ONGOING_MAINT	Software-configurable	Reading 1 means that a cache maintenance operation is in progress (clean, invalidate, or cache enable or cache disable).
[1]	ONGOING_EN_DIS	Software-configurable	Ongoing Enable or Disable. Reading 1 means that the cache is in progress of being disabled or enabled. The CACHE_ENABLED bit changes when done.
[0]	CACHE_ENABLED	Software-configurable	Cache enable status. 1 = The cache is enabled 0 = The cache is disabled.

4.4.7 SECIRQSTAT, Secure interrupt request status register

The SECIRQSTAT register is used to check the source of a Secure interrupt.

The SECIRQSTAT register characteristics are:

Attributes

Offset `0x0100`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

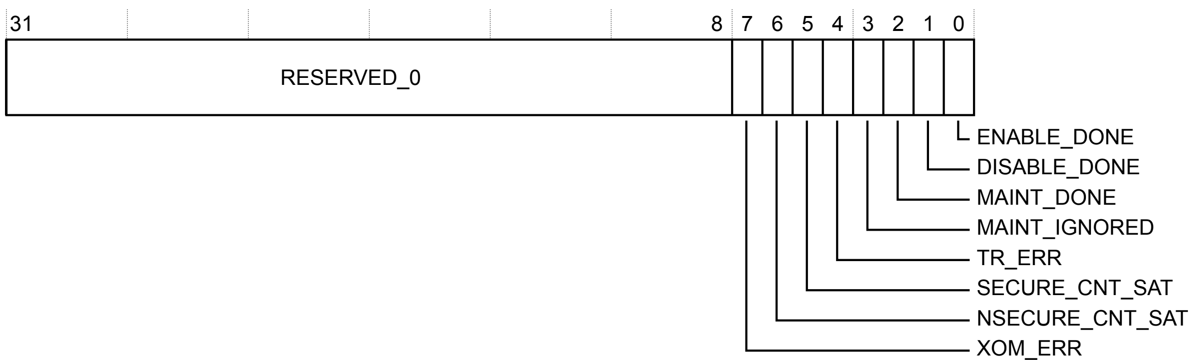

Figure 4-7 SECIRQSTAT register bit assignments

The following table shows the bit assignments.

Table 4-9 SECIRQSTAT register bit assignments

Bits	Name	Security	Function
[31:8]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[7]	XOM_ERR	Secure	A data, write, locked, or exclusive access was attempted to an XOM by a Secure transfer. **Note** If a bus master connected to the cache generates Speculative data transfers, these Speculative data transfers can cause a data access to XOM. In such a situation, a data access to XOM might not be an indication of a security threat.
[6]	NSECURE_CNT_SAT	Secure	Non-secure statistics counters are saturated and stopped (when ALLOW_NSEC_NSECSTAT is not set).
[5]	SECURE_CNT_SAT	Secure	Secure statistics counters are saturated and stopped.
[4]	TR_ERR	Secure	Secure transaction error on master side (any bus error, data type access to XOM).
[3]	MAINT_IGNORED	Secure	Secure software attempted maintenance or enable or disable of the cache. One of those operations was already in progress and the new request was ignored.
[2]	MAINT_DONE	Secure	Manual maintenance operation (either or both of clean or invalidate) started by Secure software finished.
[1]	DISABLE_DONE	Secure	The disable operation is complete. The AHB Cache is bypassed.
[0]	ENABLE_DONE	Secure	The enable operation is complete. The AHB Cache is operational.

4.4.8 SECIRQSCLR, Secure interrupt status clear register

The SECIRQSCLR register allows clearing sources for Secure interrupt.

The SECIRQSCLR register characteristics are:

Attributes

Offset `0x0104`

Type Write-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

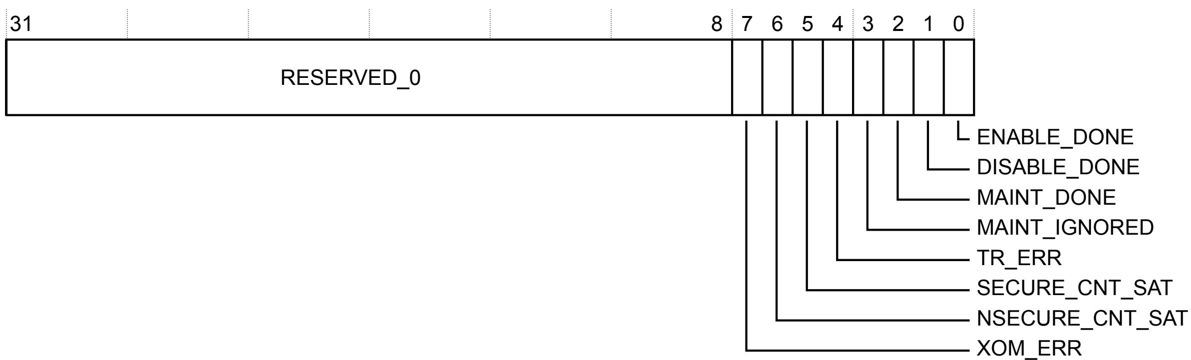

Figure 4-8 SECIRQSCLR register bit assignments

The following table shows the bit assignments.

Table 4-10 SECIRQSCLR register bit assignments

Bits	Name	Security	Function
[31:8]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[7]	XOM_ERR	Secure	Clear XOM_ERR interrupt
[6]	NSECURE_CNT_SAT	Secure	Clear NSECURE_CNT_SAT interrupt
[5]	SECURE_CNT_SAT	Secure	Clear SECURE_CNT_SAT interrupt
[4]	TR_ERR	Secure	Clear TR_ERR Interrupt
[3]	MAINT_IGNORED	Secure	Clear Secure MAINT_IGNORED interrupt
[2]	MAINT_DONE	Secure	Clear Secure MAINT_DONE interrupt
[1]	DISABLE_DONE	Secure	Clear DISABLE_DONE interrupt
[0]	ENABLE_DONE	Secure	Clear ENABLE_DONE interrupt

4.4.9 SECIRQEN, Secure interrupt enable register

The SECIRQEN register allows enabling sources for Secure interrupt. If a bit is set to zero, that source does not trigger an interrupt.

The SECIRQEN register characteristics are:

Attributes

Offset `0x0108`

Type Read-write

Reset `0x0`

Width 32

The following figure shows the bit assignments.

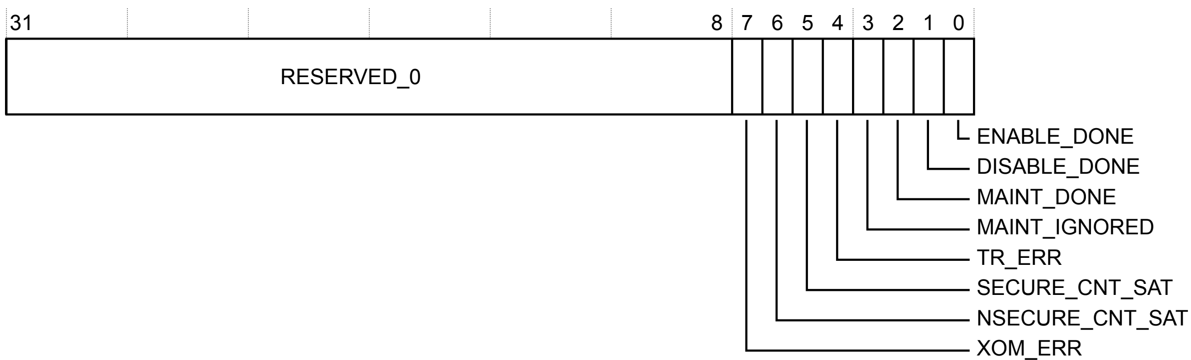

Figure 4-9 SECIRQEN register bit assignments

The following table shows the bit assignments.

Table 4-11 SECIRQEN register bit assignments

Bits	Name	Security	Function
[31:8]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[7]	XOM_ERR	Secure	Enable XOM_ERR interrupt.
[6]	NSECURE_CNT_SAT	Secure	Enable NSECURE_CNT_SAT interrupt.
[5]	SECURE_CNT_SAT	Secure	Enable SECURE_CNT_SAT interrupt.
[4]	TR_ERR	Secure	Enable TR_ERR interrupt.
[3]	MAINT_IGNORED	Secure	Enable Secure MAINT_IGNORED interrupt.
[2]	MAINT_DONE	Secure	Enable Secure MAINT_DONE interrupt.
[1]	DISABLE_DONE	Secure	Enable DISABLE_DONE interrupt.
[0]	ENABLE_DONE	Secure	Enable ENABLE_DONE interrupt.

4.4.10 SECIRQINFO1, Secure transfer error information register 1

The SECIRQINFO1 register contains the address of the operation which caused the error that triggered the Secure TR_ERR interrupt.

The SECIRQINFO1 register characteristics are:

Attributes

Offset `0x010c`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

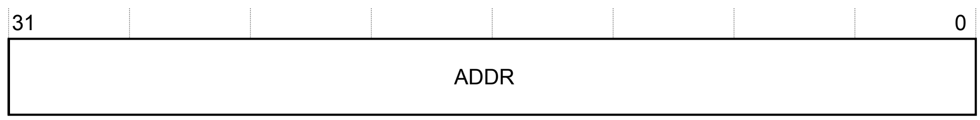

Figure 4-10 SECIRQINFO1 register bit assignments

The following table shows the bit assignments.

Table 4-12 SECIRQINFO1 register bit assignments

Bits	Name	Security	Function
[31:0]	ADDR	Secure	Address used by the Secure transfer that caused the Secure TR_ERR.

4.4.11 SECIRQINFO2, Secure transfer error information register 2

The SECIRQINFO2 register contains the master ID of the operation which caused the error that triggered the Secure TR_ERR interrupt. It also identifies the source of the error.

The SECIRQINFO2 register characteristics are:

Attributes

Offset `0x0110`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

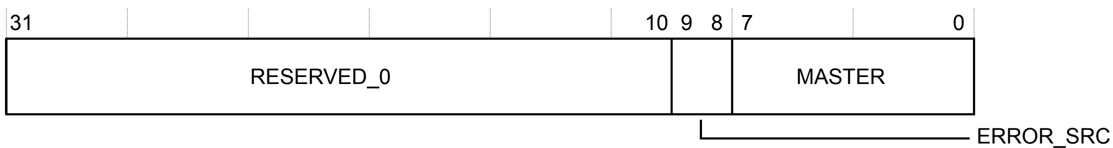

Figure 4-11 SECIRQINFO2 register bit assignments

The following table shows the bit assignments.

Table 4-13 SECIRQINFO2 register bit assignments

Bits	Name	Security	Function
[31:10]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[9:8]	ERROR_SRC	Secure	Origin of the Secure transfer that received the bus error: 0 = Early write response 1 = Eviction Write-Back 2 = Linefill read error not propagated to master. The fetched line is invalidated and any writes to it are lost.
[7:0]	MASTER	Secure	The HMASTER ID of the Secure transfer that caused the error.

4.4.12 NSECIRQSTAT, Non-secure interrupt request status register

The NSECIRQSTAT register is used to check what source caused a Non-secure interrupt.

The NSECIRQSTAT register characteristics are:

Attributes

Offset `0x0140`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

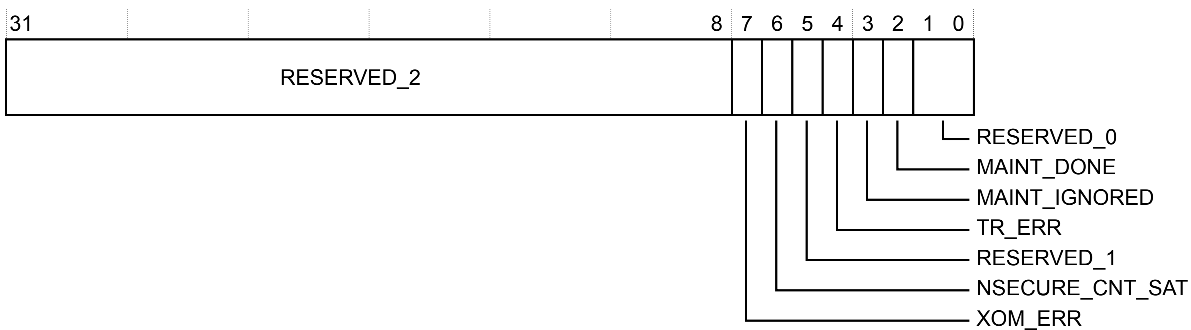

Figure 4-12 NSECIRQSTAT register bit assignments

The following table shows the bit assignments.

Table 4-14 NSECIRQSTAT register bit assignments

Bits	Name	Security	Function
[31:8]	RESERVED_2	Non-secure	Read-As-Zero, Writes Ignored.
[7]	XOM_ERR	Non-secure	A data, write, locked, or exclusive access was attempted to an XOM by a Non-secure transfer. ——— **Note** ——— If a bus master connected to the cache generates Speculative data transfers, these Speculative data transfers can cause a data access to XOM. In such a situation, a data access to XOM might not be an indication of a security threat. ———————
[6]	NSECURE_CNT_SAT	Non-secure	Non-secure statistics counters are saturated and stopped (when ALLOW_NSEC_NSECSTAT is not set).
[5]	RESERVED_1	Non-secure	Read-As-Zero, Writes Ignored.
[4]	TR_ERR	Non-secure	Non-secure transaction error on master side. The details of the transaction are saved in the NSECIRQINFOx registers.
[3]	MAINT_IGNORED	Non-secure	Non-secure software attempted maintenance or enabling or disabling of the cache while such an operation was already in progress and the new request was ignored.
[2]	MAINT_DONE	Non-secure	Manual maintenance operations (either or both of clean or invalidate) started by Non-secure software have finished.
[1:0]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.

4.4.13 NSECIRQSCLR, Non-secure interrupt status clear register

The NSECIRQCLR register allows clearing sources for Non-secure interrupt.

The NSECIRQSCLR register characteristics are:

Attributes

Offset `0x0144`

Type Write-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

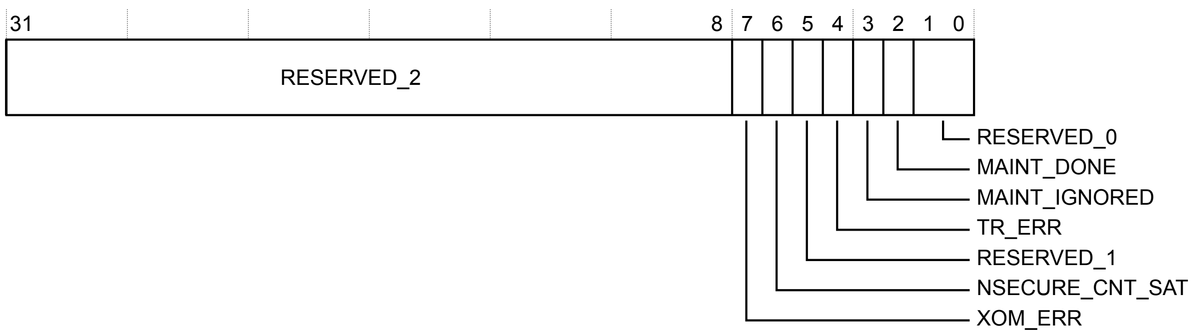

Figure 4-13 NSECIRQSCLR register bit assignments

The following table shows the bit assignments.

Table 4-15 NSECIRQSCLR register bit assignments

Bits	Name	Security	Function
[31:8]	RESERVED_2	Non-secure	Read-As-Zero, Writes Ignored.
[7]	XOM_ERR	Non-secure	Clear Non-secure XOM_ERR interrupt
[6]	NSECURE_CNT_SAT	Non-secure	Clear NSECURE_CNT_SAT interrupt
[5]	RESERVED_1	Non-secure	Read-As-Zero, Writes Ignored.
[4]	TR_ERR	Non-secure	Clear Non-secure TR_ERR interrupt
[3]	MAINT_IGNORED	Non-secure	Clear Non-secure MAINT_IGNORED interrupt
[2]	MAINT_DONE	Non-secure	Clear Non-secure MAINT_DONE interrupt
[1:0]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.

4.4.14 NSECIRQEN, Non-secure interrupt enable register

The NSECIRQEN register allows enabling sources for Non-secure interrupt. If a bit is set to zero, the source does not trigger an interrupt.

The NSECIRQEN register characteristics are:

Attributes

Offset `0x0148`

Type Read-write

Reset `0x0`

Width 32

The following figure shows the bit assignments.

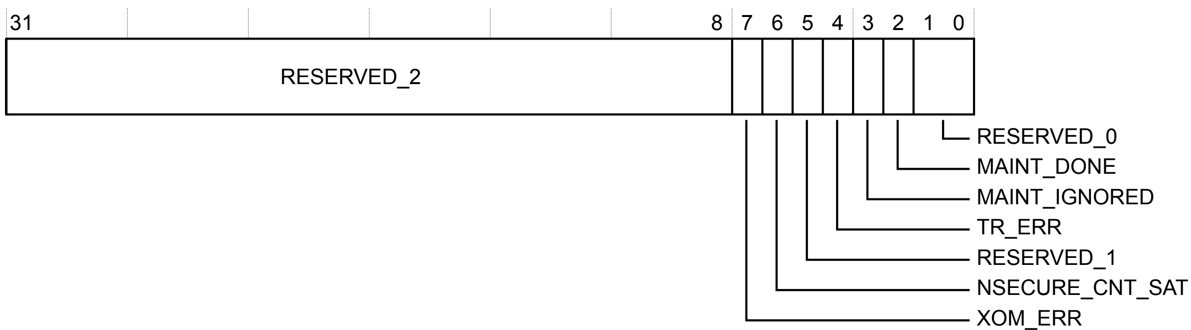

Figure 4-14 NSECIRQEN register bit assignments

The following table shows the bit assignments.

Table 4-16 NSECIRQEN register bit assignments

Bits	Name	Security	Function
[31:8]	RESERVED_2	Non-secure	Read-As-Zero, Writes Ignored.
[7]	XOM_ERR	Non-secure	Enable Non-secure XOM_ERR interrupt.
[6]	NSECURE_CNT_SAT	Non-secure	Enable NSECURE_CNT_SAT interrupt.
[5]	RESERVED_1	Non-secure	Read-As-Zero, Writes Ignored.
[4]	TR_ERR	Non-secure	Enable Non-secure TR_ERR interrupt.
[3]	MAINT_IGNORED	Non-secure	Enable Non-secure MAINT_IGNORED interrupt.
[2]	MAINT_DONE	Non-secure	Enable Non-secure MAINT_DONE interrupt.
[1:0]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.

4.4.15 NSECIRQINFO1, Non-secure transfer error information register 1

The NSECIRQINFO1 register contains the address of the operation which caused the error that triggered Non-secure TR_ERR interrupt.

The NSECIRQINFO1 register characteristics are:

Attributes

Offset `0x014c`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

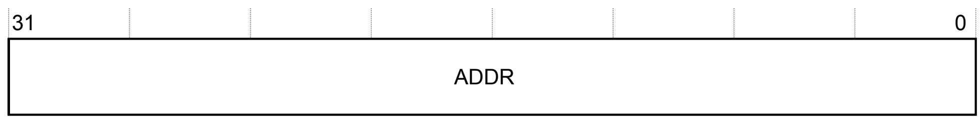

Figure 4-15 NSECIRQINFO1 register bit assignments

The following table shows the bit assignments.

Table 4-17 NSECIRQINFO1 register bit assignments

Bits	Name	Security	Function
[31:0]	ADDR	Non-secure	Address used by the Non-secure transfer that caused Non-secure TR_ERR.

4.4.16 NSECIRQINFO2, Non-secure transfer error information register 2

The NSECIRQINFO2 register contains the master ID of the operation which caused the error that triggered Non-secure TR_ERR interrupt. It also identifies the source of the error.

The NSECIRQINFO2 register characteristics are:

Attributes

Offset `0x0150`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

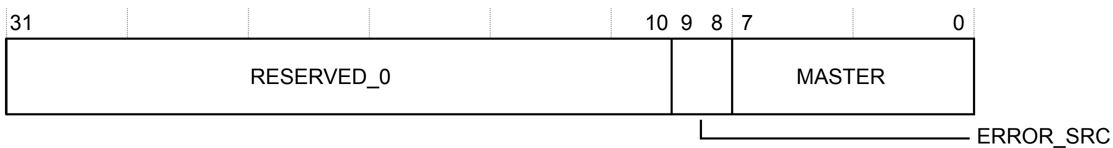

Figure 4-16 NSECIRQINFO2 register bit assignments

The following table shows the bit assignments.

Table 4-18 NSECIRQINFO2 register bit assignments

Bits	Name	Security	Function
[31:10]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[9:8]	ERROR_SRC	Non-secure	The origin of the Non-secure transfer that received the bus error. 0 = Early write response 1 = Eviction Write-Back 2 = Linefill read error not propagated to master. The fetched line is invalidated and any writes to it are lost.
[7:0]	MASTER	Non-secure	The HMASTER ID of the Non-secure transfer that caused the error.

4.4.17 SECHIT, Secure transfers hit register

The SECHIT register displays the value of the Secure hit counter.

The SECHIT register characteristics are:

Attributes

Offset `0x0300`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

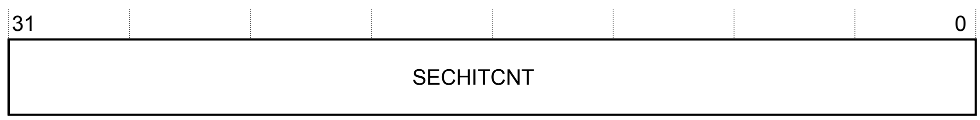

Figure 4-17 SECHIT register bit assignments

The following table shows the bit assignments.

Table 4-19 SECHIT register bit assignments

Bits	Name	Security	Function
[31:0]	SECHITCNT	Secure	The number of Secure transfers that have hit the cache.

4.4.18 SECMISS, Secure transfers miss register

The SECMISS register displays the value of the Secure miss counter.

The SECMISS register characteristics are:

Attributes

Offset `0x0304`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

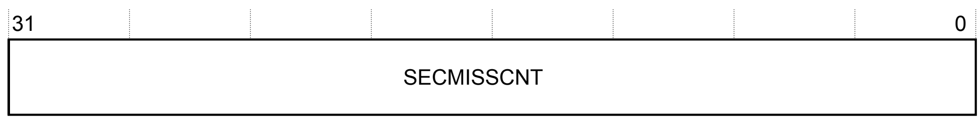

Figure 4-18 SECMISS register bit assignments

The following table shows the bit assignments.

Table 4-20 SECMISS register bit assignments

Bits	Name	Security	Function
[31:0]	SECMISSCNT	Secure	The number of Secure transfers that have missed the cache.

4.4.19 SECSTATCTRL, Secure transfers statistic counters control

The SECSTATCTRL register provides control over the Secure counters.

The SECSTATCTRL register characteristics are:

Attributes

Offset `0x0308`

Type Read-write

Reset `0x0`

Width 32

The following figure shows the bit assignments.

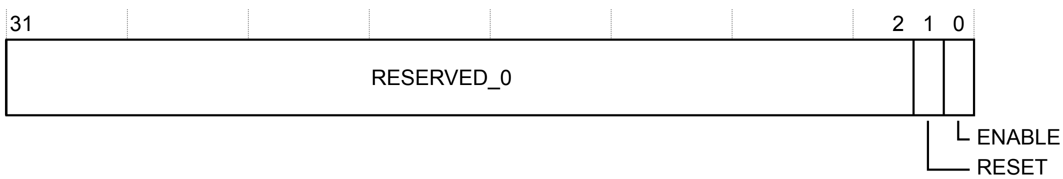

Figure 4-19 SECSTATCTRL register bit assignments

The following table shows the bit assignments.

Table 4-21 SECSTATCTRL register bit assignments

Bits	Name	Security	Function
[31:2]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[1]	RESET	Secure	Reset statistics counters for Secure transactions.
[0]	ENABLE	Secure	Enable statistics counters for Secure transactions.

4.4.20 NSECHIT, Non-secure transfers hit register

The NSECHIT register displays the value of the Non-secure hit counter.

The NSECHIT register characteristics are:

Attributes

Offset 0x0310

Type Read-only

Reset 0x0

Width 32

The following figure shows the bit assignments.

Figure 4-20 NSECHIT register bit assignments

The following table shows the bit assignments.

Table 4-22 NSECHIT register bit assignments

Bits	Name	Security	Function
[31:0]	NSECHITCNT	Software-configurable	The number of Non-secure transfers that have hit the cache.

4.4.21 NSECMISS, Non-secure transfers miss register

The NSECMISS register displays the value of the Non-secure miss counter.

The NSECMISS register characteristics are:

Attributes

Offset `0x0314`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

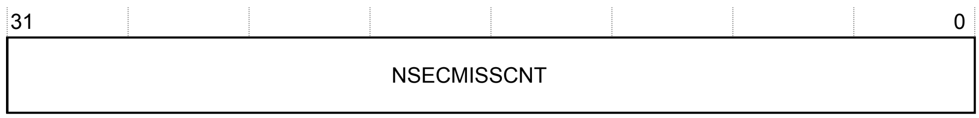

Figure 4-21 NSECMISS register bit assignments

The following table shows the bit assignments.

Table 4-23 NSECMISS register bit assignments

Bits	Name	Security	Function
[31:0]	NSECMISSCNT	Software-configurable	The number of Non-secure transfers that have missed the cache.

4.4.22 NSECSTATCTRL, Non-secure transfers statistic counters control register

The NSECSTATCTRL register provides control over the Non-secure counters.

The NSECSTATCTRL register characteristics are:

Attributes

Offset `0x0318`

Type Read-write

Reset `0x0`

Width 32

The following figure shows the bit assignments.

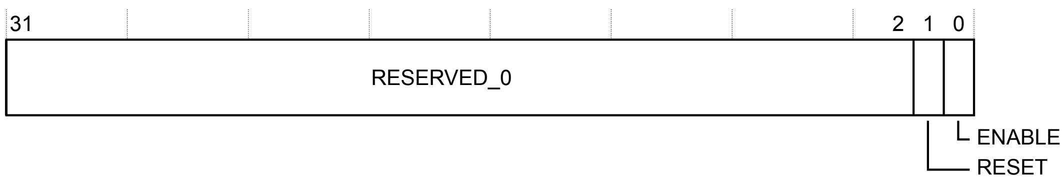

Figure 4-22 NSECSTATCTRL register bit assignments

The following table shows the bit assignments.

Table 4-24 NSECSTATCTRL register bit assignments

Bits	Name	Security	Function
[31:2]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[1]	RESET	Software-configurable	Reset statistics counters for Non-secure transactions
[0]	ENABLE	Software-configurable	Enable statistics counters for Non-secure transactions.

4.4.23 PMSVR0, saved value register 0 - Secure hit

Secure hit counter snapshot register.

The PMSVR0 register characteristics are:

Attributes

Offset `0x0600`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

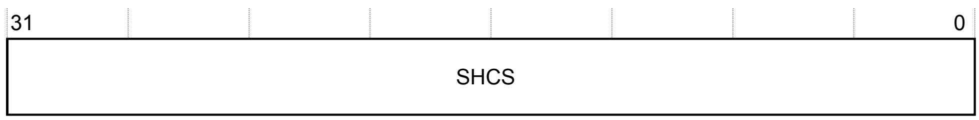

Figure 4-23 PMSVR0 register bit assignments

The following table shows the bit assignments.

Table 4-25 PMSVR0 register bit assignments

Bits	Name	Security	Function
[31:0]	SHCS	Secure	Secure hit counter snapshot

4.4.24 PMSVR1, saved value register 1 - Secure miss

Secure miss counter snapshot register.

The PMSVR1 register characteristics are:

Attributes

Offset `0x0604`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

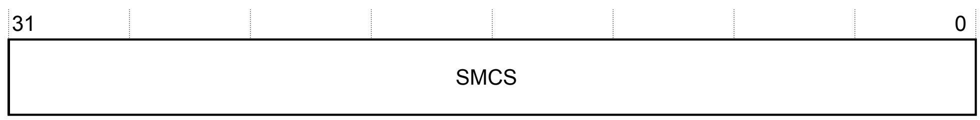

Figure 4-24 PMSVR1 register bit assignments

The following table shows the bit assignments.

Table 4-26 PMSVR1 register bit assignments

Bits	Name	Security	Function
[31:0]	SMCS	Secure	Secure miss counter snapshot

4.4.25 PMSVR2, saved value register 2 - Non-secure hit

Non-secure hit counter snapshot register.

The PMSVR2 register characteristics are:

Attributes

Offset `0x0608`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

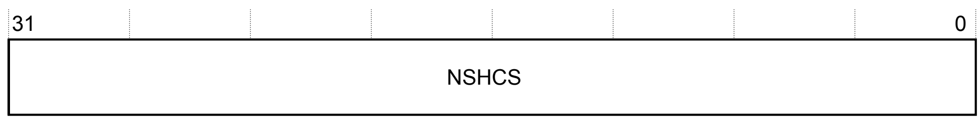

Figure 4-25 PMSVR2 register bit assignments

The following table shows the bit assignments.

Table 4-27 PMSVR2 register bit assignments

Bits	Name	Security	Function
[31:0]	NSHCS	Software-configurable	Non-secure hit counter snapshot

4.4.26 PMSVR3, saved value register 3 - Non-secure miss

Non-secure miss counter snapshot register.

The PMSVR3 register characteristics are:

Attributes

Offset `0x060C`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

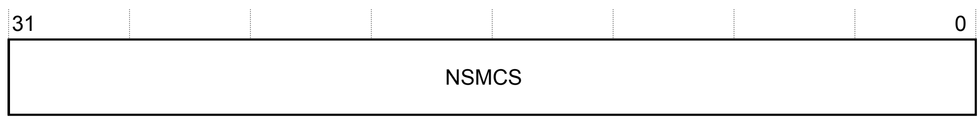

Figure 4-26 PMSVR3 register bit assignments

The following table shows the bit assignments.

Table 4-28 PMSVR3 register bit assignments

Bits	Name	Security	Function
[31:0]	NSMCS	Software-configurable	Non-secure miss counter snapshot

4.4.27 PMSSSR, PMU snapshot status register

PMU Snapshot status register.

The PMSSSR register characteristics are:

Attributes

Offset `0x0680`

Type Read-only

Reset `0x1`

Width 32

The following figure shows the bit assignments.

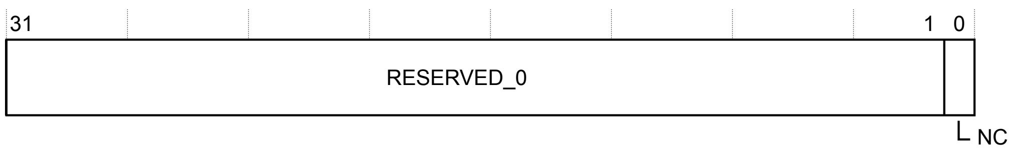

Figure 4-27 PMSSSR register bit assignments

The following table shows the bit assignments.

Table 4-29 PMSSSR register bit assignments

Bits	Name	Security	Function
[31:1]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[0]	NC	Software-configurable	No capture. Indicates whether the PMU counters have been captured.

4.4.28 PMSSCR, PMU snapshot capture register

PMU snapshot capture register

The PMSSCR register characteristics are:

Attributes

Offset `0x06F0`

Type Write-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

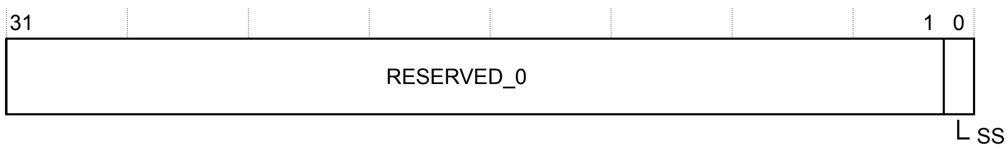

Figure 4-28 PMSSCR register bit assignments

The following table shows the bit assignments.

Table 4-30 PMSSCR register bit assignments

Bits	Name	Security	Function
[31:1]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[0]	SS	Software-configurable	Provides a mechanism for software to initiate a snapshot. Writing: 1 = Initiates a capture immediately. 0 = Ignored.

4.4.29 PMSSRR, PMU snapshot reset register

PMU snapshot reset register

The PMSSRR register characteristics are:

Attributes

Offset `0x06F4`

Type Read-write

Reset `0x0`

Width 32

The following figure shows the bit assignments.

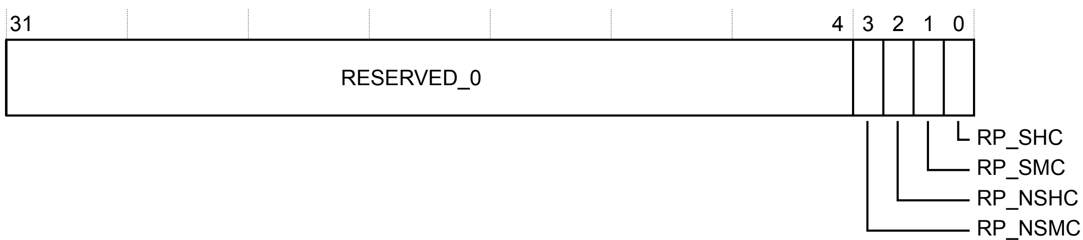

Figure 4-29 PMSSRR register bit assignments

The following table shows the bit assignments.

Table 4-31 PMSSRR register bit assignments

Bits	Name	Security	Function
[31:4]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[3]	RP_NSMC	Software-configurable	Reset Non-secure miss counter when making snapshot. Mirrors RP_NSHC as the two counters are grouped and should not be reset separately. This field is read-only.
[2]	RP_NSHC	Software-configurable	Reset Non-secure hit counter when making snapshot. The miss counter copies this value.
[1]	RP_SMC	Secure	Reset Secure miss counter when making snapshot. Mirrors RP_SHC as the two counters are grouped and should not be reset separately. This field is read-only.
[0]	RP_SHC	Secure	Reset Secure hit counter when making snapshot. The miss counter copies this value.

4.4.30 PIDR4, peripheral ID register 4

Peripheral ID 4.

The PIDR4 register characteristics are:

Attributes

Offset `0x0FD0`

Type Read-only

Reset `0x4`

Width 32

The following figure shows the bit assignments.

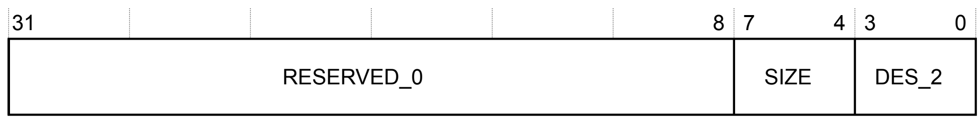

Figure 4-30 PIDR4 register bit assignments

The following table shows the bit assignments.

Table 4-32 PIDR4 register bit assignments

Bits	Name	Security	Function
[31:8]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[7:4]	SIZE	Non-secure	4KB Count.
[3:0]	DES_2	Non-secure	JEP106 Continuation Code.

4.4.31 PIDR5, peripheral ID register 5

Peripheral ID 5.

The PIDR5 register characteristics are:

Attributes

Offset `0x0FD4`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

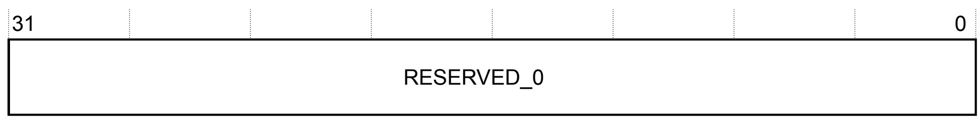

Figure 4-31 PIDR5 register bit assignments

The following table shows the bit assignments.

Table 4-33 PIDR5 register bit assignments

Bits	Name	Security	Function
[31:0]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.

4.4.32 PIDR6, peripheral ID register 6

Peripheral ID 6.

The PIDR6 register characteristics are:

Attributes

Offset `0x0FD8`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

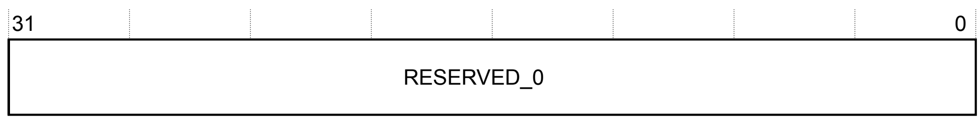

Figure 4-32 PIDR6 register bit assignments

The following table shows the bit assignments.

Table 4-34 PIDR6 register bit assignments

Bits	Name	Security	Function
[31:0]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.

4.4.33 PIDR7, peripheral ID register 7

Peripheral ID 7.

The PIDR7 register characteristics are:

Attributes

Offset `0x0FDC`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

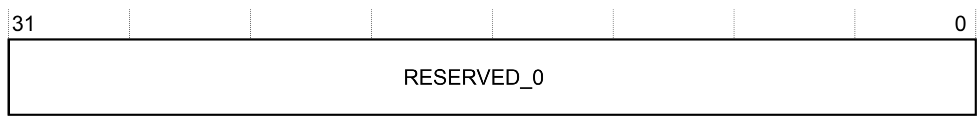

Figure 4-33 PIDR7 register bit assignments

The following table shows the bit assignments.

Table 4-35 PIDR7 register bit assignments

Bits	Name	Security	Function
[31:0]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.

4.4.34 PIDR0, peripheral ID register 0

Peripheral ID 0.

The PIDR0 register characteristics are:

Attributes

Offset `0x0FE0`

Type Read-only

Reset `0x31`

Width 32

The following figure shows the bit assignments.

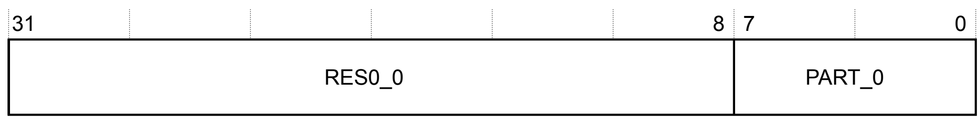

Figure 4-34 PIDR0 register bit assignments

The following table shows the bit assignments.

Table 4-36 PIDR0 register bit assignments

Bits	Name	Security	Function
[31:8]	RES0_0	Non-secure	Read-As-Zero, Writes Ignored
[7:0]	PART_0	Non-secure	Part Number [7:0].

4.4.35 PIDR1, peripheral ID register 1

Peripheral ID 1.

The PIDR1 register characteristics are:

Attributes

Offset `0x0fe4`

Type Read-only

Reset `0xb8`

Width 32

The following figure shows the bit assignments.

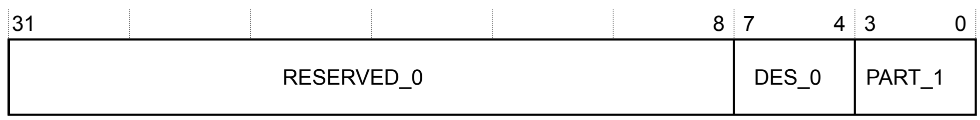

Figure 4-35 PIDR1 register bit assignments

The following table shows the bit assignments.

Table 4-37 PIDR1 register bit assignments

Bits	Name	Security	Function
[31:8]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[7:4]	DES_0	Non-secure	JEP106 Identity Code [3:0].
[3:0]	PART_1	Non-secure	Part Number [11:8].

4.4.36 PIDR2, peripheral ID register 2

Peripheral ID 2.

The PIDR2 register characteristics are:

Attributes

Offset `0x0FE8`

Type Read-only

Reset `0xB`

Width 32

The following figure shows the bit assignments.

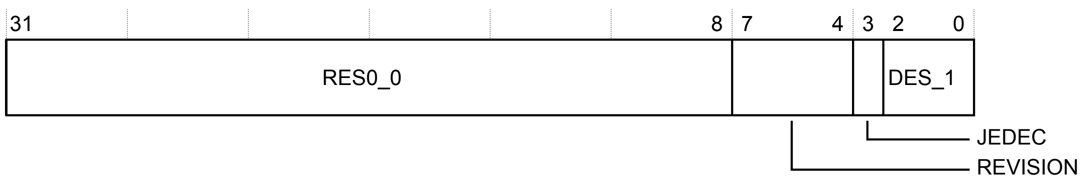

Figure 4-36 PIDR2 register bit assignments

The following table shows the bit assignments.

Table 4-38 PIDR2 register bit assignments

Bits	Name	Security	Function
[31:8]	RES0_0	Non-secure	Read-As-Zero, Writes Ignored
[7:4]	REVISION	Non-secure	Revision Code.
[3]	JEDEC	Non-secure	JEDEC.
[2:0]	DES_1	Non-secure	JEP106 Identity Code [6:4].

4.4.37 PIDR3, peripheral ID register 3

Peripheral ID 3.

The PIDR3 register characteristics are:

Attributes

Offset `0x0FEC`

Type Read-only

Reset `0x0`

Width 32

The following figure shows the bit assignments.

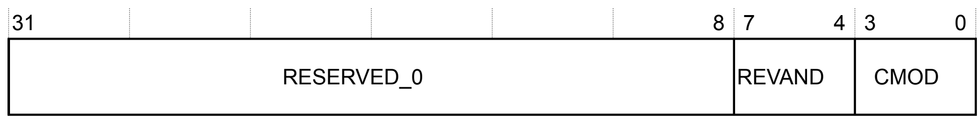

Figure 4-37 PIDR3 register bit assignments

The following table shows the bit assignments.

Table 4-39 PIDR3 register bit assignments

Bits	Name	Security	Function
[31:8]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[7:4]	REVAND	Non-secure	Manufacturer revision number.
[3:0]	CMOD	Non-secure	Customer Modified.

4.4.38 CIDR0, component ID register 0

Component ID 0.

The CIDR0 register characteristics are:

Attributes

Offset `0x0FF0`

Type Read-only

Reset `0xD`

Width 32

The following figure shows the bit assignments.

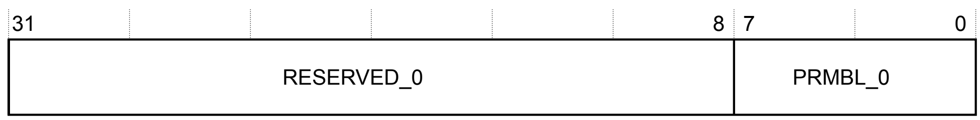

Figure 4-38 CIDR0 register bit assignments

The following table shows the bit assignments.

Table 4-40 CIDR0 register bit assignments

Bits	Name	Security	Function
[31:8]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[7:0]	PRMBL_0	Non-secure	Preamble.

4.4.39 CIDR1, component ID register 1

Component ID 1.

The CIDR1 register characteristics are:

Attributes

Offset `0x0FF4`

Type Read-only

Reset `0xF0`

Width 32

The following figure shows the bit assignments.

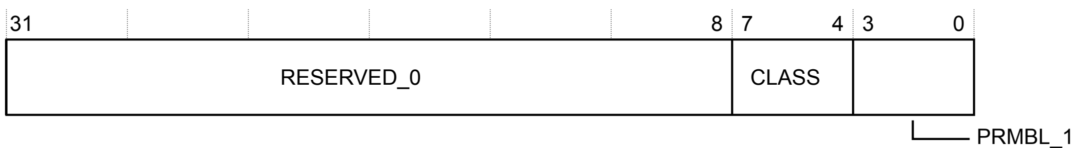

Figure 4-39 CIDR1 register bit assignments

The following table shows the bit assignments.

Table 4-41 CIDR1 register bit assignments

Bits	Name	Security	Function
[31:8]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[7:4]	CLASS	Non-secure	Component class.
[3:0]	PRMBL_1	Non-secure	Preamble.

4.4.40 CIDR2, component ID register 2

Component ID 2.

The CIDR2 register characteristics are:

Attributes

Offset `0x0FF8`

Type Read-only

Reset `0x5`

Width 32

The following figure shows the bit assignments.

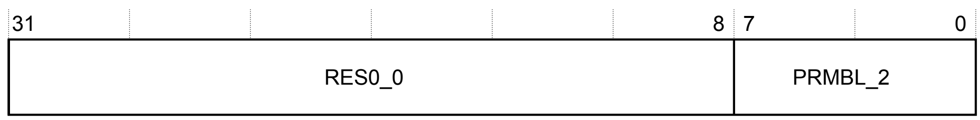

Figure 4-40 CIDR2 register bit assignments

The following table shows the bit assignments.

Table 4-42 CIDR2 register bit assignments

Bits	Name	Security	Function
[31:8]	RES0_0	Non-secure	Read-As-Zero, Writes Ignored
[7:0]	PRMBL_2	Non-secure	Preamble.

4.4.41 CIDR3, component ID register 3

Component ID 3.

The CIDR3 register characteristics are:

Attributes

Offset `0x0FFC`

Type Read-only

Reset `0xB1`

Width 32

The following figure shows the bit assignments.

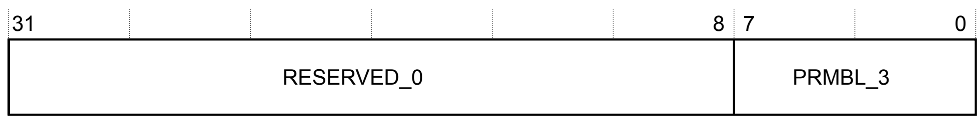

Figure 4-41 CIDR3 register bit assignments

The following table shows the bit assignments.

Table 4-43 CIDR3 register bit assignments

Bits	Name	Security	Function
[31:8]	RESERVED_0	Non-secure	Read-As-Zero, Writes Ignored.
[7:0]	PRMBL_3	Non-secure	Preamble.

Chapter 5
Using software to program the AHB Cache

This chapter provides details on programming the AHB Cache by exploring typical scenarios.

It contains the following sections:

5.1 Enable the AHB Cache by using software

Non-secure software cannot enable the cache. Secure software can enable the cache by setting the ENABLE field of the CTRL register using the APB configuration interface.

This section contains the following subsections:

5.1.1 About the CACHE_ENABLED bit

The CACHE_ENABLED bit is mirrored in two registers: in NSEC_ACCESS register and the MAINT_STATUS register.

This duplication is to ensure that the CACHE_ENABLED bit is sampled at the same time as other relevant information.

The CACHE_ENABLED bit in the NSEC_ACCESS register

In the NSEC_ACCESS register, the CACHE_ENABLED bit only holds a valid value when NSEC_ENABLE_READ_ALLOWED is set.

If NSEC_ENABLE_READ_ALLOWED is not set, the CACHE_ENABLED bit in the NSEC_ACCESS register is masked for Non-secure reads.

The CACHE_ENABLED bit in the MAINT_STATUS register

When checking the CACHE_ENABLED_bit in the MAINT_STATUS register, you should also check if the ONGOING_EN_DIS bit is set. The ONGOING_EN_DIS bit indicates that cache enable or cache disable is ongoing. If this bit is set, the CACHE_ENABLED bit only changes state once the process of enabling or disabling the cache is complete.

For example: the software reads that the cache is enabled, but the ONGOING_EN_DIS bit is also set. This means that the cache is transitioning to disabled state.

5.1.2 Enable the AHB Cache by using software with automatic maintenance on

You can use software to enable the AHB Cache when automatic maintenance is turned on.

The AHB Cache receives the enable command on the APB interface. If the **dis_cache_en_maint** is set to 0, the software makes the AHB Cache start automatic cache enable maintenance. After the AHB Cache completes the automatic cache enable maintenance, it applies the enable command.

Procedure

1. If you do not want to use interrupt handling, skip to step 2. If you want the software to use interrupt handling:
   a. Ensure that there are no pending interrupts in the Secure interrupt status register. SECIRQSTAT must read as `0x0`. If there are pending interrupts, the software must serve and clear them. The software can clear the pending interrupts by writing the active bits in the SECIRQSCLR register.
   b. Enable the ENABLE_DONE interrupt in the SECIRQEN register by writing `0x1` to the ENABLE_DONE field.
2. Enable the cache by writing `0x1` to the ENABLE field of the CTRL register.
3. Wait until the enable takes effect. Do one of the following:
   - If you are not using interrupt handling, poll the MAINT_STATUS register and wait until the CACHE_ENABLED field is set to `0x1`.
   - If you are using interrupt handling, then wait for the interrupt to occur. Check in the SECIRQSTAT register that the ENABLE_DONE field is set to `0x1`. Then clear the interrupt in the SECIRQSCLR register by writing `0x1` to the ENABLE_DONE field.

The cache is now enabled and ready to perform caching.

Example 5-1 Enable the AHB Cache by using software with automatic maintenance on

```
global bool cache_enabled_interrupt_occured = false;

EnableAHBCacheAutoMaintenanceOn()

    if(InterruptHandling) then
        // Ensure that there are no pending interrupts in
        // the Secure Interrupt Status Register
        bool no_pending_interrupts = false;
        repeat
            service_and_clear_interrupts();
            no_pending_interrupts = (ReadRegister(SECIRQSTAT) == 0x0);
        until no_pending_interrupts;

        // clear global interrupt occured flag
        cache_enabled_interrupt_occured = false;
        // Enable the ENABLE_DONE interrupt
        WriteRegisterField(SECIRQEN_ENABLE_DONE,0x1);

    // Enable the cache
    WriteRegisterField(CTRL_ENABLE,0x1);

    if(InterruptHandling) then
        // wait for interrupt to arrive (InterruptHandler)
        repeat
            Wait();
        until cache_enabled_interrupt_occured;
    else
        bool cache_enabled = false;
        repeat
            cache_enabled = (ReadRegisterField(MAINT_STATUS_CACHE_ENABLED) == 0x1);
        until cache_enabled;



SecureInterruptHandler()
    if(ReadRegisterField(SECIRQSTAT_ENABLE_DONE) == 0x1) then
        // The cache is now enabled, continue
        cache_enabled_interrupt_occured = true;
        WriteRegisterField(SECIRQCLR_ENABLE_DONE,0x1);
```

5.1.3 Enable the AHB Cache by using software with automatic maintenance off

Software can enable the AHB Cache when automatic maintenance is turned off. The AHB Cache receives the enable command on the APB interface. If the **dis_cache_en_maint** is set to 1, the cache is enabled immediately.

Note

When the cache is disabled, it can only run invalidate all maintenance.

Procedure

1. If you do not want to use interrupt handling, skip to step 2. If you want the software to use interrupt handling:
    a. Ensure that there are no pending interrupts in the Secure interrupt status register. SECIRQSTAT must read as `0x0`. If there are pending interrupts, the software must serve and clear them. The software can clear the pending interrupts by writing the active bits in the SECIRQSCLR register.
    b. Enable the MAINT_DONE and ENABLE_DONE interrupts in the SECIRQEN register by writing `0x1` to the MAINT_DONE and ENABLE_DONE fields.
2. Trigger a manual invalidate all operation by writing `0x1` to the TRIG_INVALIDATE_ALL field in the MAINT_CTRL_ALL register.

3. Wait until the maintenance is finished. Do one of the following:
   - If you are not using interrupt handling, poll the MAINT_STATUS register and wait until the ONGOING_MAINT field is set to `0x0`.
   - If you are using interrupt handling, wait for the interrupt to occur. Check that the MAINT_DONE field is set to `0x1` in the SECIRQSTAT register. Then clear the interrupt by writing the MAINT_DONE field with `0x1` in the SECIRQSCLR register.
4. Enable the cache by writing `0x1` to the ENABLE field in the CTRL register.
5. Wait until the enable takes effect. Do one of the following:
   - If you are not using interrupt handling, poll the MAINT_STATUS register and wait until the CACHE_ENABLED field is set to `0x1`.
   - If you are using interrupt handling, wait for the interrupt to occur. Check that the ENABLE_DONE field is set to `0x1` in the SECIRQSTAT register. Clear the interrupt by writing the ENABLE_DONE field with `0x1`. in the SECIRQSCLR register.

The cache is now enabled and ready to perform caching.

Example 5-2 Enable the AHB Cache by using software with automatic maintenance off

```
global bool cache_enabled_interrupt_occured = false;
global bool cache_invalidate_all_interrupt_occured = false;

EnableAHBCacheAutoMaintenanceOff()

    if(InterruptHandling) then
        // Ensure that there are no pending interrupts in
        // the Secure Interrupt Status Register
        bool no_pending_interrupts = false;
        repeat
            service_and_clear_interrupts();
            no_pending_interrupts = (ReadRegister(SECIRQSTAT) == 0x0);
        until no_pending_interrupts;

        // clear all global interrupt occured flags
        cache_enabled_interrupt_occured = false;
        cache_invalidate_all_interrupt_occured = false;

        // Enable the ENABLE_DONE interrupt
        WriteRegisterField(SECIRQEN_ENABLE_DONE,0x1);
        // Enable the MAINT_DONE interrupt
        WriteRegisterField(SECIRQEN_MAINT_DONE,0x1);

    // Trigger a manual invalidate all operation
    WriteRegisterField(MAINT_CTRL_ALL_TRIG_INVALIDATE_ALL,0x1);

    if(InterruptHandling) then
        // wait for maint_done interrupt to arrive (InterruptHandler)
        repeat
            Wait();
        until cache_invalidate_all_interrupt_occured;
    else
        bool maint_finished = false;
        repeat
            maint_finished = (ReadRegisterField(MAINT_STATUS_ONGOING_MAINT) == 0x0);
        until maint_finished;

    // Enable the cache
    WriteRegisterField(CTRL_ENABLE,0x1);

    if(InterruptHandling) then
        // wait for enable_done interrupt to arrive (InterruptHandler)
        repeat
            Wait();
        until cache_enabled_interrupt_occured;
    else
        bool cache_enabled = false;
        repeat
            cache_enabled = (ReadRegisterField(MAINT_STATUS_CACHE_ENABLED) == 0x1);
        until cache_enabled;



SecureInterruptHandler()
     if(ReadRegisterField(SECIRQSTAT_ENABLE_DONE) == 0x1) then
```

```
        // The cache is now enabled, continue
        cache_enabled_interrupt_occured = true;
        WriteRegisterField(SECIRQCLR_ENABLE_DONE,0x1);
    if (ReadRegisterField(SECIRQSTAT_MAINT_DONE) == 0x1) then
        // The cache is invalidated, continue
        cache_invalidate_all_interrupt_occured = true;
        WriteRegisterField(SECIRQCLR_MAINT_DONE,0x1);
```

5.2 Disable the AHB Cache using software

Non-secure software cannot disable the cache. Secure software can disable the AHB Cache using the APB configuration interface.

This section contains the following subsections:

- *5.2.1 Disable the AHB Cache by using software with automatic maintenance on* on page 5-100.
- *5.2.2 Disable the AHB Cache by using software with automatic maintenance off* on page 5-101.

5.2.1 Disable the AHB Cache by using software with automatic maintenance on

Software can disable the AHB Cache when automatic maintenance is turned on.

The AHB Cache receives the disable command on the APB interface. If the **dis_cache_dis_maint** is set to 1'b0, the software makes the AHB Cache start automatic cache disable maintenance. During cache disable maintenance, traffic on the AHB interface is stalled so that new accesses cannot make the cache dirty. After the AHB Cache completes the automatic cache disable maintenance, it applies the disable command.

Procedure

1. If you do not want to use interrupt handling, skip to step 2. If you want the software to use interrupt handling:
   a. Ensure that there are no pending interrupts in the Secure interrupt status register. SECIRQSTAT register must read as `0x0`. If there are pending interrupts, the software must serve and clear them. The software can clear the pending interrupts by writing the active bits in the SECIRQSCLR register.
   b. Enable the DISABLE_DONE interrupt by writing `0x1` to the DISABLE_DONE field in the SECIRQEN register.
2. Disable the cache by writing `0x0` to the ENABLE field in the CTRL register.
3. Wait until the disable takes effect. Do one of the following:
   - If you are not using interrupt handling, poll the MAINT_STATUS register and wait until the CACHE_ENABLED field is set to `0x0`.
   - If you are using interrupt handling, wait for the interrupt to occur. Check in the SECIRQSTAT register that the DISABLE_DONE field is set to `0x1`. Then clear the interrupt by writing `0x1` to the DISABLE_DONE field in the SECIRQSCLR register.

The cache is now disabled.

Example 5-3 Disable the AHB Cache by using software with automatic maintenance on

```
global bool cache_disabled_interrupt_occured = false;

DisableAHBCacheAutoMaintenanceOn()

    if(InterruptHandling) then
        // Ensure that there are no pending interrupts in
        // the Secure Interrupt Status Register
        bool no_pending_interrupts = false;
        repeat
            service_and_clear_interrupts();
            no_pending_interrupts = (ReadRegister(SECIRQSTAT) == 0x0);
        until no_pending_interrupts;

        // clear global interrupt occured flag
        cache_disabled_interrupt_occured = false;
        // Enable the DISABLE_DONE interrupt
        WriteRegisterField(SECIRQEN_DISABLE_DONE,0x1);

    // Disable the cache
    WriteRegisterField(CTRL_ENABLE,0x0);

    if(InterruptHandling) then
```

```
        // wait for interrupt to arrive (InterruptHandler)
        repeat
            Wait();
        until cache_disabled_interrupt_occured;
    else
        bool cache_disabled = false;
        repeat
            cache_disabled = (ReadRegisterField(MAINT_STATUS_CACHE_ENABLED) == 0x0);
        until cache_disabled;



SecureInterruptHandler()
     if(ReadRegisterField(SECIRQSTAT_DISABLE_DONE) == 0x1) then
         // The cache is now disabled, continue
         cache_disabled_interrupt_occured = true;
         WriteRegisterField(SECIRQCLR_DISABLE_DONE,0x1);
```

5.2.2 Disable the AHB Cache by using software with automatic maintenance off

Software can disable the AHB Cache when automatic maintenance is turned off. The following sequence describes how the software can disable the cache, assuming that the **dis_cache_dis_maint** is set to 1.

If the **dis_cache_dis_maint** is set to 1, the cache does not start clean all maintenance before it applies the cache disable command that it has received on the APB interface. To prevent data loss, software must perform clean all maintenance before disabling the cache. The software must also make sure that no Cacheable transaction is generated after the clean all maintenance and before the cache disable commands.

Note

We recommend you use a data memory barrier (DMB or DSB).

Procedure

1. If you want to use interrupts, then complete the following steps, otherwise skip to step 3:
   a. Ensure that there are no pending interrupts in the Secure interrupt status register. SECIRQSTAT register must read as `0x0`. If there are pending interrupts, the software must serve and clear them. The software can clear the pending interrupts by writing the active bits in the SECIRQSCLR register.
   b. Enable the DISABLE_DONE interrupt by writing `0x1` to the DISABLE_DONE field in the SECIRQEN register.

      The MAINT_DONE interrupt is not required, as the commands are launched back-to-back and the cache performs the maintenance and disable operations in the right order if initiated properly.
2. Trigger a manual clean all operation by writing `0x1` to the TRIG_CLEAN_ALL register field in the MAINT_CTRL_ALL register.
3. Disable global interrupts.
4. Disable the cache by writing `0x0` to the ENABLE field in the CTRL register.
5. Enable the global interrupt.
6. Wait until the disable takes effect. Do one of the following:
   - If you are not using interrupt handling, poll the MAINT_STATUS register. Wait until both CACHE_ENABLED and ONGOING_MAINT fields are set to `0x0`.
   - If you are using interrupt handling, then wait for the interrupt to occur, check that both DISABLE_DONE and MAINT_DONE fields are set to `0x1` in the SECIRQSTAT register. Then clear the interrupt by writing the DISABLE_DONE and MAINT_DONE fields with `0x1` in the SECIRQSCLR register.

The cache is now disabled.

Example 5-4 Disable the AHB Cache by using software with automatic maintenance off

```
global bool cache_disabled_interrupt_occured = false;
global bool cache_invalidate_all_interrupt_occured = false;

DisableAHBCacheAutoMaintenanceOff()

    if(InterruptHandling) then
        // Ensure that there are no pending interrupts in
        // the Secure Interrupt Status Register
        bool no_pending_interrupts = false;
        repeat
            service_and_clear_interrupts();
            no_pending_interrupts = (ReadRegister(SECIRQSTAT) == 0x0);
        until no_pending_interrupts;

        // clear global interrupt occured flags
        cache_disabled_interrupt_occured = false;
        cache_invalidate_all_interrupt_occured = false;

        // Enable the DISABLE_DONE interrupt
        WriteRegisterField(SECIRQEN_DISABLE_DONE,0x1);

    // Disable global interrupts
    DisableGlogalInterrupts();

    // Trigger a manual clean all operation
    WriteRegisterField(MAINT_CTRL_ALL_TRIG_CLEAN_ALL,0x1);

    // Disable the cache
    WriteRegisterField(CTRL_ENABLE,0x0);

    // Enable global interrupts
    EnableGlogalInterrupts();

    if(InterruptHandling) then
        // wait for the DISABLE_DONE interrupt to arrive (InterruptHandler)
        repeat
            Wait();
        until (cache_disabled_interrupt_occured && cache_invalidate_all_interrupt_occured);
    else
        bool cache_disabled = false;
        repeat
            cache_disabled = ((ReadRegisterField(MAINT_STATUS_CACHE_ENABLED) == 0x0) &&
                              (ReadRegisterField(MAINT_STATUS_ONGOING_MAINT) == 0x0));
        until cache_disabled;



SecureInterruptHandler()
     if(ReadRegisterField(SECIRQSTAT_DISABLE_DONE) == 0x1) then
        // The cache is now disabled, continue
        cache_disabled_interrupt_occured = true;
        WriteRegisterField(SECIRQCLR_DISABLE_DONE,0x1);
     if (ReadRegisterField(SECIRQSTAT_MAINT_DONE) == 0x1) then
        // The cache is invalidated, continue
        cache_invalidate_all_interrupt_occured = true;
        WriteRegisterField(SECIRQCLR_MAINT_DONE,0x1);
```

5.3 Use Non-secure software to check cache enable status

Non-secure software can check the cache enable status.

To allow Non-secure software to check the cache enable state, Secure software writes `0x1` to the ALLOW_NSEC_ENABLE_READ field in the CTRL register. Non-secure software can then perform the following steps:

Procedure

1. Read the NSEC_ENABLE_READ_ALLOWED field in the NSEC_ACCESS register to check that it can read the cache enable state. The field should be set to `0x1`.
2. Check the CACHE_ENABLED register field value matches with expectations.

5.4 Configurable cache diagnostics available for Non-secure software

Non-secure software cannot enable or disable the cache. It has limited access to the cache enable status.

Secure software can grant rights to Non-secure software by writing to the specific fields in the CTRL register. The following table describes the relevant fields.

Table 5-1 CTRL register fields for configuring Non-secure software permissions

Field name	Description
ALLOW_NSEC_NSECSTAT	Non-secure software can access and manage the Non-secure statistics counters.
ALLOW_NSEC_MAINT_LINES	Non-secure software can request maintenance by cache lines operations for cached Non-secure data. Non-secure software can access ongoing maintenance status information in the MAIN_STATUS register. ——— **Note** ——— Non-secure software cannot destroy data inside the cache. Non-secure software cannot request an invalidate without requesting a clean at the same time. ———————
ALLOW_NSEC_ENABLE_READ	Non-secure software can read the cache enable status in the MAIN_STATUS register.

Non-secure software can check the granted rights by reading the corresponding register field values in the NSEC_ACCESS status register.

——— **Note** ———

When NSEC_ACCESS.NSEC_MAINT_LINES_ALLOWED=0, Non-secure software is not allowed to initiate any type of manual maintenance.

———————

The following table shows which maintenance type is available to Secure and Non-secure software.

Table 5-2 Manual maintenance available for software

Maintenance type	Available to Secure software	Available to Non-secure software when NSEC_ACCESS.NSEC_MAINT_LINES_ALLOWED=1
Manual invalidate all	Yes	No
Manual clean all	Yes	No
Manual clean and invalidate all	Yes	No
Manual invalidate by address	Yes	No
Manual clean by address	Yes	Yes
Manual clean and invalidate by address	Yes	Yes

5.5 Use software for manual maintenance on the AHB Cache

Secure and Non-secure software can perform manual maintenance on the AHB Cache.

This section contains the following subsections:

5.5.1 Use Secure software to perform manual clean all or invalidate all maintenance

If the cache is enabled, Secure software can trigger manual maintenance operations. Software can manually trigger clean all, invalidate all, or clean and invalidate all maintenance. Secure software acknowledges when the maintenance is completed, if interrupt handling is enabled.

Note

Invalidate all maintenance is the only type of maintenance that can be started with a disabled cache.

Procedure

1. Ensure that the cache is able to accept a new maintenance request.

   To check that no maintenance is running, read the MAINT_STATUS register. The ONGOING_EN_DIS, ONGOING_MAINT, and ONGOING_PWR_MAINT fields must read as `0x0`. If maintenance is running, you must wait until it completes.
2. Ensure that there are no pending interrupts in the interrupt status register.

   The SECIRQSTAT register must read as `0x0`. If there are pending interrupts, the software has to serve and clear them. The software can clear the pending interrupts by writing the active bits in the SECIRQSCLR register.
3. If you are not using interrupt handling, skip this step.

   If you are using interrupt handling: in the SECIRQEN register, enable the MAINT_DONE and the MAINT_IGNORED interrupt by writing `0x1` to the MAINT_DONE and the MAINT_IGNORED fields.
4. Write the relevant value to one of the following fields in the MAINT_CTRL_ALL register to trigger manual invalidate or clean all maintenance.

Table 5-3 Triggering maintenance operations using the MAINT_CTRL_ALL register

Maintenance operation	Register field	Value
Clean all	TRIG_CLEAN_ALL	`0x1`
Invalidate all	TRIG_INVALIDATE_ALL	`0x2`
Clean and invalidate all	TRIG_CLEAN_ALL and TRIG_INVALIDATE_ALL	`0x3`

5. Wait until the maintenance is completed. Do one of the following:
   - If you are not using interrupt handling, poll the SECIRQSTAT register and wait until the MAINT_DONE or the MAINT_IGNORED field is set to `0x1`.
   - If you are using interrupt handling, wait for the interrupt to occur. In the SECIRQSTAT register, either the MAINT_DONE or the MAINT_IGNORED field must be set to `0x1`.
6. Check the cache maintenance status. If in the SECIRQSTAT register the MAINT_DONE field is set to `0x1` and MAINT_IGNORED is `0x0`, the cache maintenance is complete. Otherwise the cache has not been not cleaned or invalidated properly. The maintenance request could have been ignored.
7. In the SECIRQSCLR register, write `0x1` to the corresponding field to clear the SECIRQSTAT register.

Example 5-5 Use Secure software to perform manual clean all or invalidate all maintenance

```
global bool cache_maint_ignored_interrupt_occured = false;
global bool cache_maint_done_interrupt_occured = false;

// The function returns true when the maintenance was successful or
// false when the maintenance was ignored.
// maintenance_type can be CLEAN_ALL INVALIDATE_ALL or CLEAN_AND_INVALIDATE_ALL
bool AHBCacheMaintenance(maintenance_type)

    // Ensure the cache is able to accept a new maintenance request
    bool no_ongoing_maintenance = false;
    repeat
        no_ongoing_maintenance = ( !ReadRegisterField(MAINT_STATUS_ONGOING_EN_DIS) &&
                                   !ReadRegisterField(MAINT_STATUS_ONGOING_MAINT) &&
                                   !ReadRegisterField(MAINT_STATUS_ONGOING_PWR_MAINT);
    until no_ongoing_maintenance;

    // Ensure that there are no pending interrupts in the Secure Interrupt Status Register
    bool no_pending_interrupts = false;
    repeat
        service_and_clear_interrupts();
        no_pending_interrupts = (ReadRegister(SECIRQSTAT) == 0x0);
    until no_pending_interrupts;

    if(InterruptHandling) then
        // clear global interrupt occured flags
        cache_maint_ignored_interrupt_occured = false;
        cache_maint_done_interrupt_occured = false;

        // Enable the MAINT_DONE and MAINT_IGNORED interrupts
        WriteRegisterField(SECIRQEN_MAINT_DONE,0x1);
        WriteRegisterField(SECIRQEN_MAINT_IGNORED,0x1);

    // Trigger the maintenance operation
    case maintenance_type of
        when CLEAN_ALL
            WriteRegisterField(MAINT_CTRL_ALL_TRIG_CLEAN_ALL,0x1);
        when INVALIDATE_ALL
            WriteRegisterField(MAINT_CTRL_ALL_TRIG_INVALIDATE_ALL,0x1);
        when CLEAN_AND_INVALIDATE_ALL
            WriteRegister(MAINT_CTRL_ALL,0x3);

    if(InterruptHandling) then
        // wait for MAINT_DONE or MAINT_IGNORED interrupt to arrive (InterruptHandler)
        repeat
            Wait();
        until (cache_maint_ignored_interrupt_occured || cache_maint_done_interrupt_occured);
        // evaluate result
        if(cache_maint_ignored_interrupt_occured) then
            return false;
        else
            return true;
    else
        while true do
            if(ReadRegisterField(SECIRQSTAT_MAINT_IGNORED) == 0x1) then
                WriteRegisterField(SECIRQCLR_MAINT_IGNORED,0x1);
                return false;
            elsif(ReadRegisterField(SECIRQSTAT_MAINT_DONE) == 0x1) then
                WriteRegisterField(SECIRQCLR_MAINT_DONE,0x1);
                return true;


SecureInterruptHandler()
    if(ReadRegisterField(SECIRQSTAT_MAINT_IGNORED) == 0x1) then
        // The cache maintenance is ignored, continue
        cache_maint_ignored_interrupt_occured = true;
        WriteRegisterField(SECIRQCLR_MAINT_IGNORED,0x1);
    if(ReadRegisterField(SECIRQSTAT_MAINT_DONE) == 0x1) then
        // The cache maintenance is completed, continue
        cache_maint_done_interrupt_occured = true;
        WriteRegisterField(SECIRQCLR_MAINT_DONE,0x1);
```

5.5.2 Use Secure software to perform manual maintenance by address

If the cache is enabled, Secure software can trigger manual maintenance operations by address.

Note

To perform manual by address maintenance, you must provide an address range which aligns with cache line boundaries.

Procedure

1. Ensure that the cache is able to accept a new maintenance request.

   To check that no enable or disable maintenance is running, read the MAINT_STATUS register. The ONGOING_EN_DIS, ONGOING_MAINT, and ONGOING_PWR_MAINT fields must read as `0x0`. If maintenance is running, you must wait until it has completed.
2. Ensure that there are no pending interrupts in the interrupt status register.

   The SECIRQSTAT register must read as `0x0`. If there are pending interrupts, the Secure software must serve and clear them. The software can clear the pending interrupts by writing the active bits in the SECIRQSCLR register.
3. If you are not using interrupt handling, skip this step.

   If you are using interrupt handling, enable the MAINT_DONE and the MAINT_IGNORED interrupt. Write `0x1` to the MAINT_DONE and the MAINT_IGNORED fields in the SECIRQEN register.
4. You must use a single Write-Access to perform a, b, and c:
   a. Write the address which you want to invalidate or clean to the ADDR field of the MAIN_CTRL_LINES register. The address bits [4:0] are ignored.
   b. Write to the SECURITY field of the MAINT_CTRL_LINES register to set the security for the memory range which is going to be invalidated or cleaned
   c. Write the relevant value to one of the following fields in the MAINT_CTRL_LINES register to trigger manual invalidate or clean all maintenance for a memory range with the parameters set in step 4.

Table 5-4 Triggering maintenance operations using the MAINT_CTRL_LINES register

Maintenance operation	Register field	Value
Clean	TRIG_CLEAN	`0x1`
Invalidate	TRIG_INVALIDATE	`0x2`
Clean and invalidate	TRIG_CLEAN and TRIG_INVALIDATE	`0x3`

5. Wait until the maintenance is completed. Do one of the following:
   - If you are not using interrupt handling, poll the SECIRQSTAT register, and wait until the MAINT_DONE or the MAINT_IGNORED field is set to `0x1`.
   - If you are using interrupt handling, wait for the interrupt to occur. In the SECIRQSTAT register, the MAINT_DONE or the MAINT_IGNORED field must be set to `0x1`.
6. Check the cache maintenance status. If in the SECIRQSTAT register the MAINT_DONE field is set to `0x1` and the MAINT IGNORED field is set to `0x0`, the cache maintenance is complete. Otherwise, the cache line written in the ADDR field of the MAINT_CTRL_LINES has not been cleaned or invalidated properly. The maintenance request could have been ignored.
7. In the SECIRQSCLR register, write `0x1` to the corresponding field to clear the SECIRQSTAT register.

Note

Perform steps 4-5 for the remaining addresses in the memory region which you want to be invalidated or cleaned.

Example 5-6 Use Secure software to perform manual maintenance by address

```
global bool cache_maint_ignored_interrupt_occured = false;
global bool cache_maint_done_interrupt_occured = false;

// The function returns true when the maintenance was successful or
// false when the maintenance was ignored.
// address is the 32 bit address you want to invalidate or clean, security is the memory
// type of that address - NON_SECURE or SECURE, maintenance_type can be CLEAN, INVALIDATE
// or CLEAN_AND_INVALIDATE
bool AHBCacheMaintenancebyAddressSecure(address, security, maintenance_type)

    // Ensure the cache is able to accept a new maintenance request
    bool no_ongoing_maintenance = false;
    repeat
        no_ongoing_maintenance = ( !ReadRegisterField(MAINT_STATUS_ONGOING_EN_DIS) &&
                                   !ReadRegisterField(MAINT_STATUS_ONGOING_MAINT) &&
                                   !ReadRegisterField(MAINT_STATUS_ONGOING_PWR_MAINT);
    until no_ongoing_maintenance;

    // Ensure that there are no pending interrupts in the Secure Interrupt Status Register
    bool no_pending_interrupts = false;
    repeat
        service_and_clear_interrupts();
        no_pending_interrupts = (ReadRegister(SECIRQSTAT) == 0x0);
    until no_pending_interrupts;

    if(InterruptHandling) then
        // clear global interrupt occured flags
        cache_maint_ignored_interrupt_occured = false;
        cache_maint_done_interrupt_occured = false;

        // Enable the MAINT_DONE and MAINT_IGNORED interrupts
        WriteRegisterField(SECIRQEN_MAINT_DONE,0x1);
        WriteRegisterField(SECIRQEN_MAINT_IGNORED,0x1);

    // Write the address you want to invalidate and/or clean
    LineMaintenanceSecure(address, security, maintenance_type);

    if(InterruptHandling) then
        // wait for MAINT_DONE or MAINT_IGNORED interrupt to arrive (InterruptHandler)
        repeat
            Wait();
        until (cache_maint_ignored_interrupt_occured || cache_maint_done_interrupt_occured);
        // evaluate result
        if(cache_maint_ignored_interrupt_occured) then
            return false;
        else
            return true;
    else
        while true do
            if(ReadRegisterField(SECIRQSTAT_MAINT_IGNORED) == 0x1) then
                WriteRegisterField(SECIRQCLR_MAINT_IGNORED,0x1);
                return false;
            elsif(ReadRegisterField(SECIRQSTAT_MAINT_DONE) == 0x1) then
                WriteRegisterField(SECIRQCLR_MAINT_DONE,0x1);
                return true;


LineMaintenanceSecure(address, security, maintenance_type)
    integer secure;
    integer maintenance;

     // Resolve the security
    case security of
    when NON_SECURE
        secure = 0x0;
    when SECURE
        secure = 0x1;

    // Resolve the maintenance_type
    case maintenance_type of
    when CLEAN
```

```
         maintenance = 0x1;
    when INVALIDATE
         maintenance = 0x2;
    when CLEAN_AND_INVALIDATE
         maintenance = 0x3;

    // Set the value to be written to the MAINT_CTRL_LINES register
    // -- address [31:5], reserved [4:3], security [2], invalidate [1], clean [0]
    maintenance_register_value = (address & 0xffffffe0) + (secure << 2) + maintenance;
    WriteRegister(MAINT_CTRL_LINES,maintenance_register_value);


SecureInterruptHandler()
    if(ReadRegisterField(SECIRQSTAT_MAINT_IGNORED) == 0x1) then
        // The cache maintenance is ignored, continue
        cache_maint_ignored_interrupt_occured = true;
        WriteRegisterField(SECIRQCLR_MAINT_IGNORED,0x1);
    if(ReadRegisterField(SECIRQSTAT_MAINT_DONE) == 0x1) then
        // The cache maintenance is completed, continue
        cache_maint_done_interrupt_occured = true;
        WriteRegisterField(SECIRQCLR_MAINT_DONE,0x1);
```

5.5.3 Use Non-secure software to perform manual maintenance by address

If the cache is enabled and Secure software sets the correct register field, Non-secure software can trigger manual maintenance operations by address.

To allow Non-secure software to trigger manual maintenance, Secure software must set the NSEC_MAINT_LINES_ALLOWED register field to `0x1` in the CTRL register.

Note

To perform manual by address maintenance, you must provide an address range which aligns with cache line boundaries.

Procedure

1. Read the NSEC_ACCESS register and ensure that the NSEC_MAINT_LINES_ALLOWED register field is set to `0x1`.
2. Ensure that the cache is able to accept a new maintenance request.

   To check that no enable or disable maintenance is running, read the MAINT_STATUS register. The ONGOING_EN_DIS, ONGOING_MAINT, and ONGOING_PWR_MAINT fields must read as `0x0`. If maintenance is running, you must wait until it has completed.
3. Ensure that there are no pending interrupts in the interrupt status register.

   NSECIRQSTAT register must read as `0x0`. If there are pending interrupts, the Non-secure software must serve and clear them. The Non-secure software can clear the interrupts by writing the corresponding bits in the NSECIRQSCLR register.
4. If you are not using interrupt handling, skip this step.

   If you are using interrupt handling: write `0x1` to the MAINT_DONE and the MAINT_IGNORED fields in the NSECIRQEN register, to enable the MAINT_DONE and the MAINT_IGNORED interrupt.
5. Use the same Write-Access to perform steps a and b.
   a. Write the address which needs to be invalidated or cleaned to the ADDR field of the MAIN_CTRL_LINES register. The address bits [4:0] are ignored. This address is automatically considered as Non-secure.
   b. To trigger manual clean or clean and invalidate maintenance for a memory range with the parameters set in step a, write the relevant value to one of the following fields in the MAINT_CTRL_LINES register.

Note

Non-secure software cannot perform invalidate by address maintenance without also performing clean by address maintenance.

Table 5-5 Triggering maintenance operations using the MAINT_CTRL_LINES register

Maintenance operation	Register field	Value
Clean	TRIG_CLEAN	`0x1`
Clean and invalidate	TRIG_INVALIDATE and TRIG_CLEAN	`0x3`

6. Wait until the maintenance is complete. Do one of the following:
   - If you are not using interrupt handling, poll the SECIRQSTAT register, and wait until the MAINT_DONE or the MAINT_IGNORED field is set to `0x1`.
   - If you are using interrupt handling, wait for the interrupt to occur. In the NSECIRQSTAT register, the MAINT_DONE or the MAINT_IGNORED field must be set to `0x1`.
7. Check the cache maintenace status. If the MAINT_IGNORED field is not set to `0x1` in the NSECIRQSTAT register, the memory range written in the ADDR field of the MAINT_CTRL_LINES is now cleaned or cleaned and invalidated. Otherwise it means that the cache line is not cleaned or invalidated properly.

   Note

   Perform steps 5-6 for the remaining addresses in the memory region which needs to be invalidated or cleaned.

8. Clear the interrupt by writing 1 to the corresponding field in the NSECIRQSCLR register.

Example 5-7 Use Non-secure software to perform manual maintenance by address

```
global bool cache_maint_ignored_interrupt_occured = false;
global bool cache_maint_done_interrupt_occured = false;

// The function returns true when the maintenance is succesful or
// false when the maintenance is ignored.
// address is the 32 bit address you want to invalidate or clean, maintenance_type can be
// CLEAN or CLEAN_AND_INVALIDATE (Non secure software cannot perform invalidate by
// address without clean)
bool AHBCacheMaintenancebyAddressNonSecure(address, maintenance_type)

    // Make sure the Non secure software maintenance is enabled
    if (ReadRegisterField(NSEC_ACCESS_NSEC_MAINT_LINES_ALLOWED) == 0 ) then
        Error();

    // Make sure the cache is able to accept a new maintenance request
    bool no_ongoing_maintenance = false;
    repeat
        no_ongoing_maintenance = ( !ReadRegisterField(MAINT_STATUS_ONGOING_EN_DIS) &&
                                   !ReadRegisterField(MAINT_STATUS_ONGOING_MAINT) &&
                                   !ReadRegisterField(MAINT_STATUS_ONGOING_PWR_MAINT);
    until no_ongoing_maintenance;

    // Make sure that there are no pending interrupts in the Non secure Interrupt
    // Status Register
    bool no_pending_interrupts = false;
    repeat
        service_and_clear_interrupts();
        no_pending_interrupts = (ReadRegister(NSECIRQSTAT) == 0x0);
    until no_pending_interrupts;

    if(InterruptHandling) then
        // clear global interrupt occured flags
        cache_maint_ignored_interrupt_occured = false;
        cache_maint_done_interrupt_occured = false;

        // Enable the MAINT_DONE and MAINT_IGNORED interrupts
```

```
        WriteRegisterField(NSECIRQEN_MAINT_DONE,0x1);
        WriteRegisterField(NSECIRQEN_MAINT_IGNORED,0x1);

    // Write the address you want to invalidate and/or clean
    LineMaintenanceNonSecure(address, maintenance_type);

    if(InterruptHandling) then
        // wait for MAINT_DONE or MAINT_IGNORED interrupt to arrive (InterruptHandler)
        repeat
            Wait();
        until (cache_maint_ignored_interrupt_occured || cache_maint_done_interrupt_occured);
        // evaluate result
        if(cache_maint_ignored_interrupt_occured) then
            return false;
        else
            return true;
    else
        while true do
            if(ReadRegisterField(NSECIRQSTAT_MAINT_IGNORED) == 0x1) then
                WriteRegisterField(NSECIRQCLR_MAINT_IGNORED,0x1);
                return false;
            elsif(ReadRegisterField(NSECIRQSTAT_MAINT_DONE) == 0x1) then
                WriteRegisterField(NSECIRQCLR_MAINT_DONE,0x1);
                return true;


LineMaintenanceNonSecure(address, maintenance_type)
     integer maintenance;

     // Resolve the maintenance_type
     case maintenance_type of
     when CLEAN
         maintenance = 0x1;
     when CLEAN_AND_INVALIDATE
         maintenance = 0x3;

     // Set the value to be written to the MAINT_CTRL_LINES register
     // -- address [31:5], reserved [4:3], security = 0 [2], invalidate [1], clean [0]
     maintenance_register_value = (address & 0xffffffe0) + maintenance;
     WriteRegister(MAINT_CTRL_LINES,maintenance_register_value);


NonSecureInterruptHandler()
    if(ReadRegisterField(NSECIRQSTAT_MAINT_IGNORED) == 0x1) then
        // The cache maintenance is ignored, continue
        cache_maint_ignored_interrupt_occured = true;
        WriteRegisterField(NSECIRQCLR_MAINT_IGNORED,0x1);
    if(ReadRegisterField(NSECIRQSTAT_MAINT_DONE) == 0x1) then
        // The cache maintenance is completed, continue
        cache_maint_done_interrupt_occured = true;
        WriteRegisterField(NSECIRQCLR_MAINT_DONE,0x1);
```

5.6 Use software to access the statistics counters in the AHB Cache

The AHB Cache provides four statistics counters: Secure hit and miss, and Non-secure hit and miss counters.

This section contains the following subsections:

5.6.1 Access the Secure statistics counters

Only Secure software can use the Secure statistics counters in the design.

Procedure

1. Ensure that there are no pending interrupts in the interrupt status register. The SECIRQSTAT register must read as `0x0`.

   If there are pending interrupts, the Secure software must serve and clear them. The software can clear the pending interrupts by writing the active bits in the SECIRQSCLR register.
2. Enable the statistics counters for Secure transactions by writing `0x1` to the ENABLE field in the SECSTATCTRL register.
3. Reset the statistics counters for Secure transactions by writing `0x1` to the RESET field in the SECSTATCTRL register.
4. Execute the code on which the profiling is needed.
5. Disable the statistics counters for Secure transactions by writing `0x0` to the ENABLE field in the SECSTATCTRL register.
6. Check that the SECURE_CNT_SAT field is not set in the SECIRQSTAT register to ensure that the measurements are valid.
7. Read the hit statistics from the SECHIT register.
8. Read the miss statistics from the SECMISS register.

Example 5-8 The Secure statistics counters

```
// Call this function when there are no pending interrupts in the interrupt status register.
// (The SECIRQSTAT register must read as 0x0.)
AHBCacheSecureStatisticCounter()

    // Clear SECIRQSTAT_SECURE_CNT_SAT if it is set
    WriteRegisterField(SECIRQSCLR_SECURE_CNT_SAT, 0x1);

    // Enable the statistics counters for Secure transactions
    WriteRegisterField(SECSTATCTRL_ENABLE,0x1);

    // Reset the statistics counters for Secure transactions
    WriteRegisterField(SECSTATCTRL_RESET,0x1);

    // Execute the code you need to profile

    ...

    // Disable the statistics counters for Secure transactions
    WriteRegisterField(SECSTATCTRL_ENABLE,0x0);

    integer hit_count;
    integer miss_count;

    // If the counters were saturated then their values are invalid
    if (ReadRegisterField(SECIRQSTAT_SECURE_CNT_SAT) == 0x1) then
        Error();
    else
        hit_count = ReadRegister(SECHIT);
        miss_count = ReadRegister(SECMISS);
```

```
    print("Hit count is " + hit_count);
    print("Miss count is " + miss_count);
```

5.6.2 Use Secure software to access the Non-secure statistics counters

Secure software can use the Non-secure statistics counters in the design.

If ALLOW_NSEC_NSECSTAT is set and the Secure software starts the Non-secure counters, then saturation triggers a Secure interrupt.

Procedure

1. Ensure that there are no pending interrupts in the interrupt status register. The SECIRQSTAT register must read as `0x0`.

   If there are pending interrupts, the Secure software must serve and clear them. The software can clear the pending interrupts by writing the active bits in the SECIRQSCLR register.
2. Enable the statistics counters for Non-secure transactions by writing `0x1` to the ENABLE field in the NSECSTATCTRL register.
3. Reset the statistics counters for Non-secure transactions by writing `0x1` to the RESET field in the NSECSTATCTRL register.
4. Execute the code on which the profiling is needed.
5. Disable the statistics counters for Non-secure transactions by writing `0x0` to the ENABLE field in the NSECSTATCTRL register.
6. Check in the SECIRQSTAT register that the NSECURE_CNT_SAT field is not set to ensure that the measurements are valid.
7. Read the hit statistics from the NSECHIT register.
8. Read the miss statistics from the NSECMISS register.

Example 5-9 Use Secure software to access the Non-secure statistics counters

```
// Call this function when there are no pending interrupts in the interrupt status register.
// (The SECIRQSTAT register must read as 0x0.)
AHBCacheSecureUsingNonSecureStatisticCounter()

    // Clear SECIRQSTAT_NSECURE_CNT_SAT if it is set
    WriteRegisterField(SECIRQSCLR_NSECURE_CNT_SAT, 0x1);

    // Enable the statistics counters for Secure transactions
    WriteRegisterField(NSECSTATCTRL_ENABLE,0x1);

    // Reset the statistics counters for Secure transactions
    WriteRegisterField(NSECSTATCTRL_RESET,0x1);

    // Execute the code you need to profile

    ...

    // Disable the statistics counters for Secure transactions
    WriteRegisterField(NSECSTATCTRL_ENABLE,0x0);

    integer hit_count;
    integer miss_count;

    // If the counters were saturated then their values are invalid
    if (ReadRegisterField(SECIRQSTAT_NSECURE_CNT_SAT) == 0x1) then
        Error();
    else
        hit_count = ReadRegister(NSECHIT);
        miss_count = ReadRegister(NSECMISS);

    print("Hit count is " + hit_count);
    print("Miss count is " + miss_count);
```

5.6.3 Use Non-secure software to access the Non-secure statistics counters

Non-secure software can use the Non-secure statistics counters.

To allow the Non-secure software to use the Non-secure counters, Secure software must set the NSEC_NSECSTAT_ALLOWED register field to `0x1` in the CTRL register.

Note

Non-secure software cannot access the Secure statistics.

Procedure

1. Read the NSEC_ACCESS register and ensure that the NSEC_NSECSTAT_ALLOWED register field is set to `0x1`.
2. Ensure that there are no pending interrupts in the interrupt status register. NSECIRQSTAT register must read as `0x0`.

   If there are pending interrupts, the Non-secure software must serve and clear them. The Non-secure software can clear the interrupts by writing the corresponding bits in the NSECIRQSCLR register.
3. Enable the statistics counters for Non-secure transactions by writing `0x1` to the ENABLE field in the NSECSTATCTRL register.
4. Reset the statistics counters for Non-secure transactions by writing `0x1` to the RESET field in the NSECSTATCTRL register.
5. Execute the Non-secure code on which the profiling is needed.
6. Disable the statistics counters for Non-secure transactions by writing `0x0` to the ENABLE field in the NSECSTATCTRL register.
7. Check in the SECIRQSTAT register that the NSECURE_CNT_SAT field is not set to ensure that the measurements are valid.
8. Read the hit statistics from the NSECHIT register.
9. Read the miss statistics from the NSECMISS register.

Example 5-10 Use Non-secure software to access the Non-secure statistics counters

```
// Call this function when there are no pending interrupts in the interrupt status register.
// (The SECIRQSTAT register must read as 0x0.)
AHBCacheNonSecureUsingNonSecureStatisticCounter()

    // Ensure the non secure statistics is enabled
    if (ReadRegisterField(NSEC_ACCESS_NSEC_NSECSTAT) == 0 ) then
        Error();

    // Clear NSECIRQSTAT_NSECURE_CNT_SAT if it is set
    WriteRegisterField(NSECIRQSCLR_NSECURE_CNT_SAT, 0x1);

    // Enable the statistics counters for non secure transactions
    WriteRegisterField(NSECSTATCTRL_ENABLE,0x1);

    // Reset the statistics counters for non secure transactions
    WriteRegisterField(NSECSTATCTRL_RESET,0x1);

    // Execute the code you need to profile

    ...

    // Disable the statistics counters for non secure transactions
    WriteRegisterField(NSECSTATCTRL_ENABLE,0x0);

    integer hit_count;
    integer miss_count;
```

```
    // If the counters were saturated then their values are invalid
    if (ReadRegisterField(NSECIRQSTAT_NSECURE_CNT_SAT) == 0x1) then
        Error();
    else
        hit_count = ReadRegister(NSECHIT);
        miss_count = ReadRegister(NSECMISS);

    print("Hit count is: " + hit_count);
    print("Miss count is: " + miss_count);
```

5.7 Power control

The AHB Cache supports various low-power features such as clock or power gating.

This section contains the following subsections:

5.7.1 Sleep mode and clock gating

The design supports architectural clock gating.

You do not need to use software to perform architectural clock gating.

When the QSTOPPED state is requested on the LPI Clock Q-Channel interface, a hardware handshake is performed through the LPI.

If no clock is required, the AHB Cache automatically accepts the quiescence state. The cache requests a clock when it is needed for its internal operation.

5.7.2 Sleep mode with cache RAMs in retention

The AHB Cache supports retention or RAM retention through the QSTOPPED state of LPI Power Q-Channel interface.

You do not have to use software to trigger sleep mode with cache RAMs in retention.

When the QSTOPPED state is requested, the AHB Cache starts clean all automatic maintenance.

Note

You can use software to check that the configuration port **dis_pwr_down_maint** is set to 0 by reading the DIS_PWR_DOWN_MAINT field in the HWPARAMS register.

When exiting from retention state, the cache returns to its previous state. For example, if the AHB Cache was enabled before entering retention state, it is enabled again after exiting the retention state and it remains enabled. The cache data is preserved.

5.7.3 Sleep mode with cache RAMs in power down state

As the LPI Q-Channel supports only a single power domain, the individual management of logic and RAM domains is not supported.

5.7.4 Handle Warm reset by using software

The cache supports Warm reset through the QSTOPPED state of LPI Power Q-Channel interface.

When the QSTOPPED state is requested, the AHB Cache starts clean all automatic maintenance.

To ensure reliably quick acceptance of the low-power request, you can use software to disable clean all automatic maintenance by setting the configuration port **dis_pwr_down_maint** to 1. Otherwise, no software interaction is needed.

The software can check that the configuration port **dis_pwr_down_maint** is set to 1 by reading the DIS_PWR_DOWN_MAINT field in the HWPARAMS register.

When exiting from Warm reset state without being reset, the cache returns to its previous state. If the cache was enabled in its previous state, it remains enabled. The cached data is preserved.

Appendix A
Signal descriptions

This appendix describes the AHB Cache interface signals.

It contains the following sections:

A.1 Clock and reset signals

To reduce power consumption when not in active use, the AHB Cache allows clock gating of the module.

Table A-1 Clock and reset signals

Signal name	Direction	Connection	Description
clk	Input	Clock generator	Clock signal. All signal timings are related to the rising edge.
resetn	Input	Reset controller	Active-LOW reset. This signal resets all flops asynchronously when asserted. ——— **Note** ——— **resetn** must be deasserted synchronously with **clk**. ———————

Related concepts

2.1 Clocking and reset on page 2-24

A.2 LPI signals

The AHB Cache uses LPI Q-Channels for power and clock management.

AHB Cache clock and power management is based on the standard modes and handshake behavior as specified in the *AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces, Issue C*.

Table A-2 Power control LPI-Q signals

Signal name	Direction	Connection	Description
pwr_qreqn	Input	PPU or power controller	Active-LOW quiescence request signal driven by the power controller.
pwr_qacceptn	Output	PPU or power controller	When LOW, this signal indicates that the AHB Cache accepts the quiescence request from the power controller.
pwr_qdeny	Output	PPU or power controller	When HIGH, this signal indicates that the AHB Cache denies the quiescence request from the power controller.
pwr_qactive	Output	PPU or power controller	This signal, when HIGH, indicates to the controller that the AHB Cache needs power. When the signal is driven LOW, the AHB Cache might accept a quiescence request.

Table A-3 Clock control LPI-Q signals

Signal name	Direction	Connection	Description
clk_qreqn	Input	Clock controller	Active-LOW quiescence request signal driven by the clock controller.
clk_qacceptn	Output	Clock controller	When LOW, this signal indicates that the AHB Cache accepts the quiescence request from the clock controller.
clk_qdeny	Output	Clock controller	When HIGH, this signal indicates that the AHB Cache denies the quiescence request from the clock controller.
clk_qactive	Output	Clock controller	This signal, when HIGH, indicates to the controller that the AHB Cache requires the clock. When the signal is driven LOW, the AHB Cache might accept a quiescence request.

Related concepts

2.5 Low-Power Interface on page 2-31
2.5.2 Clock LPI on page 2-31
2.5.3 Power LPI on page 2-31

A.3 AHB Slave interface signals

The AHB Slave interface receives AHB transfers and forwards them to the AHB master or performs a cache lookup.

Note

The parameterized indexes are derived from configuration parameters. For more information, see the *Arm® CoreLink™ AHB Cache Configuration and Integration Manual*.

Table A-4 AHB slave interface signals

Signal name	Direction	Connection	Description
hsel_s	Input	CPU or system interconnect	Slave select. This signal qualifies a valid transfer.
hnonsec_s	Input	CPU or system interconnect	Non-secure transfer indicator
haddr_s[31:0]	Input	CPU or system interconnect	This is a 32-bit bus address.
htrans_s[1:0]	Input	CPU or system interconnect	Transfer type
hsize_s[2:0]	Input	CPU or system interconnect	Indicates the size of the transfer.
hwrite_s	Input	CPU or system interconnect	Indicates the direction of a transfer.
hready_s	Input	CPU or system interconnect	Indicates the completion of a transfer.
hprot_s[6:0]	Input	CPU or system interconnect	This signal indicates whether the transfer is: • A privileged or normal access • A data or an instruction access It also indicates the memory attributes of a transfer.
hburst_s[2:0]	Input	CPU or system interconnect	Indicates the Burst type.
hmastlock_s	Input	CPU or system interconnect	Indicates a locked sequence.
hwdata_s[31:0]	Input	CPU or system interconnect	Write data. The AHB Cache can forward this signal through a direct path or a buffer path.
hexcl_s	Input	CPU or system interconnect	Indicates an exclusive transfer.
hmaster_s[HMASTER_WIDTH-1:0]	Input	CPU or system interconnect	Master identifier, configurable width. The AHB Cache forwards this input when forwarding a transfer to the AHB Master interface.
hrdata_s[31:0]	Output	CPU or system interconnect	Read data

Table A-4 AHB slave interface signals (continued)

Signal name	Direction	Connection	Description
hreadyout_s	Output	CPU or system interconnect	Completion indicator for transfers targeting the AHB Cache. The AHB Cache drives this signal LOW to extend the transfer.
hresp_s	Output	CPU or system interconnect	Transfer response. The slave uses this signal to indicate errors.
hexokay_s	Output	CPU or system interconnect	Exclusive okay
hruser_s[HRUSER_WIDTH-1:0]	Output	CPU or system interconnect	Read channel User signals, configurable bus width. If the AHB Cache cannot provide valid read channel user signals for cacheable read transfers, it sets the value to 0. The input for **hruser_s** is `0x0` for a cacheable transfer. **Note** Only present when `HRUSER_WIDTH > 0`.
hauser_s[HAUSER_WIDTH-1:0]	Input	CPU or system interconnect	Address channel User signals, configurable bus width. **Note** Only present when `HAUSER_WIDTH > 0`.
hwuser_s[HWUSER_WIDTH-1:0]	Input	CPU or system interconnect	Write channel User signals, configurable bus width. **Note** Only present when `HWUSER_WIDTH > 0`.

Table A-5 AHB Slave sideband signals

Signal name	Direction	Connection	Description
hrxom_s	Output	CPU or system interconnect	The AHB Cache drives this signal in response to a read request. The timing for **hrxom_s** must follow the timing for **hruser_s[HRUSER_WIDTH-1:0]**. **Note** Only present when `XOM=ON`.
hdebug_s	Input	CPU, debugger, or system interconnect	AHB address phase sideband signal. This signal indicates that the AHB access is a debug access. **hdebug_s** can be decoded from HMASTER, if the debugger has a specific identifier.

A.4 AHB Master interface signals

The AHB master interface has many functions, including generating the Linefill and Write-Back AHB transactions and forwarding Non-cacheable transactions.

Note

The parameterized indexes are derived from configuration parameters. For more information, see the *Arm® CoreLink™ AHB Cache Configuration and Integration Manual*.

Table A-6 AHB Master signals

Signal name	Direction	Connection	Description
hnonsec_m	Output	System interconnect or memory controller	Non-secure transfer indicator
haddr_m[31:0]	Output	System interconnect or memory controller	Address
htrans_m[1:0]	Output	System interconnect or memory controller	Transfer type
hsize_m[2:0]	Output	System interconnect or memory controller	Indicates the size of the transfer.
hwrite_m	Output	System interconnect or memory controller	Indicates the direction of a transfer.
hready_m	Input	System interconnect or memory controller	Indicates the completion of a transfer. The slave drives this ready signal LOW to extend the transfer. The inactive state is HIGH.
hprot_m[6:0]	Output	System interconnect or memory controller	Protection control
hburst_m[2:0]	Output	System interconnect or memory controller	Indicates the burst type.
hmastlock_m	Output	System interconnect or memory controller	Indicates a locked sequence.
hwdata_m[31:0]	Output	System interconnect or memory controller	Write data
hexcl_m	Output	System interconnect or memory controller	Indicates an exclusive transfer.
hmaster_m[HMASTER_WIDTH-1:0]	Output	System interconnect or memory controller	Master identifier, configurable width.
hrdata_m[31:0]	Input	System interconnect or memory controller	Read data
hresp_m	Input	System interconnect or memory controller	Transfer response. The slave uses this signal to indicate errors.
hexokay_m	Input	System interconnect or memory controller	Exclusive okay

Table A-6 AHB Master signals (continued)

Signal name	Direction	Connection	Description
hruser_m[HRUSER_WIDTH-1:0]	Input	System interconnect or memory controller	Read channel User signals, configurable width. **Note** Only present when `HRUSER_WIDTH > 0`.
hauser_m[HAUSER_WIDTH-1:0]	Output	System interconnect or memory controller	Address channel User signals, configurable width. Set to 0 when the transfer is a linefill or a Write-Back generated by the cache. All other transfers, including buffered write transfers, forward this user signal. **Note** Only present when `HAUSER_WIDTH > 0`.
hwuser_m[HWUSER_WIDTH-1:0]	Output	System interconnect or memory controller	Write channel User signals, configurable width. Set to 0 when the transfer is a linefill or a Write-Back generated by the cache. All other transfers, including buffered write transfers, forward this user signal. **Note** Only present when `HWUSER_WIDTH > 0`.

Table A-7 AHB Master sideband signals

Signal name	Direction	Connection	Description
hrxom_m	Input	System interconnect or memory controller	The slave asserts this signal responding to a read request, when a transaction hits an XOM region. The XOM attribute must be consistent throughout each cache line. The signal must follow the same timing as **hruser_m**. **Note** Only present when `XOM=ON`.

A.5 APB interface signals

The APB interface provides a software control interface to the AHB Cache.

Table A-8 APB interface signals

Signal name	Direction	Connection	Description
paddr[11:0]	Input	APB bridge or interconnect	12-bit address bus
pwrite	Input	APB bridge or interconnect	When HIGH, this signal indicates a write. When LOW, it indicates a read.
psel	Input	APB bridge or interconnect	When HIGH, the APB slave is selected.
penable	Input	APB bridge or interconnect	Starts the APB transfer one cycle after **psel**.
pwdata [31:0]	Input	APB bridge or interconnect	The data input of the APB slave when **pwrite** is HIGH.
prdata [31:0]	Output	APB bridge or interconnect	The read data output of the APB slave when **pwrite** is LOW.
pready	Output	APB bridge or interconnect	The slave drives this ready signal LOW to extend the transfer.
pslverr	Output	APB bridge or interconnect	The slave uses this error signal to indicate that an error has occurred during the transfer and that the transfer was aborted.
pprot[2:0]	Input	APB bridge or interconnect	APB Protection type. This signal indicates the privilege or security level of the transaction and whether the transaction is a data access or an instruction access.
pstrb[3:0]	Input	APB bridge or interconnect	Write strobes. This signal indicates which byte lanes to update during a write transfer. There is one write strobe for every 8 bits of the write data bus. Write strobes must not be active during a read transfer.

Table A-9 APB interface sideband signals

Signal name	Direction	Connection	Description
pclken	Input	Clock generator	The clock enable signal. This signal allows the APB to run on a divided frequency. —— **Note** —— This signal must be periodical and synchronous to **clk**. ————
pwakeup	Input	APB bridge or interconnect	Wake up signal. This signal is used to indicate that there is ongoing activity that is associated with the APB interface.

A.6 System interface signals

The AHB Cache has separate interrupt lines for signaling Secure or Non-secure transaction-related events. It also has a hardware status signal for powerdown maintenance. For performance monitoring, the AHB Cache provides a hardware trigger signal for the snapshotting feature.

Table A-10 System signals

Signal name	Direction	Connection	Description
sec_irq	Output	CPU or NVIC	Secure interrupt request
nsec_irq	Output	CPU or NVIC	Non-secure interrupt request
pwr_maintenance	Output	CPU or power controller	Status signal for an ongoing powerdown maintenance operation.
pmsnapshotreq	Input	Performance Monitoring Unit (PMU)	A trigger signal which initiates the capture of the current value of the statistics counters. Must be a synchronous pulse. **Note** Only present when `SNAPSHOTTING=ON`.

Related concepts

2.4 Interrupts on page 2-30

A.7 Memory interface signals

Each type of memory interface has a dedicated set of signals.

This section contains the following subsections:

- *A.7.1 Data RAM interface signals* on page Appx-A-126.
- *A.7.2 Tag RAM interface signals* on page Appx-A-126.
- *A.7.3 Dirty RAM interface signals* on page Appx-A-127.

A.7.1 Data RAM interface signals

The Data RAM interface executes Data-RAM-related read or write requests.

For information about the *Index Width* (IW), *Data RAM Address Width* (DAW), and *Tag RAM Data Width* (TDW), see section 6.3.1. *RAM bus widths* in the *Arm® CoreLink™ AHB Cache Configuration and Integration Manual*.

Table A-11 Cache Data RAM interface signals

Signal name	Direction	Connection	Description
data_ram_cs[3:0]	Output	Data RAM	Data RAM chip select for the respective data RAM block.
data_ram_bytewe[3:0]	Output	Data RAM	Data RAM byte-write enable. This bus is connected to each RAM block.
data_ram_addr0[DAW-1:0]	Output	Data RAM	Address for Data RAM block0. The upper index is cache-size dependent.
data_ram_addr1[DAW-1:0]	Output	Data RAM	Address for Data RAM block1. The upper index is cache-size dependent.
data_ram_addr2[DAW-1:0]	Output	Data RAM	Address for Data RAM block2. The upper index is cache-size dependent.
data_ram_addr3[DAW-1:0]	Output	Data RAM	Address for Data RAM block3. The upper index is cache-size dependent.
data_ram_wdata0[31:0]	Output	Data RAM	Write data to Data RAM block0
data_ram_wdata1[31:0]	Output	Data RAM	Write data to Data RAM block1
data_ram_wdata2[31:0]	Output	Data RAM	Write data to Data RAM block2
data_ram_wdata3[31:0]	Output	Data RAM	Write data to Data RAM block3
data_ram_rdata0[31:0]	Input	Data RAM	Read data from Data RAM block0
data_ram_rdata1[31:0]	Input	Data RAM	Read data from Data RAM block1
data_ram_rdata2[31:0]	Input	Data RAM	Read data from Data RAM block2
data_ram_rdata3[31:0]	Input	Data RAM	Read data from Data RAM block3

A.7.2 Tag RAM interface signals

The Tag RAM interface executes Tag RAM-related read or write requests.

For information about the *Index Width* (IW), *Data RAM Address Width* (DAW), and *Tag RAM Data Width* (TDW), see section 6.3.1. *RAM bus widths* in the *Arm® CoreLink™ AHB Cache Configuration and Integration Manual*.

Table A-12 Cache Tag RAM interface signals

Signal name	Direction	Connection	Description
tag_ram_cs[3:0]	Output	Tag RAM	Data RAM chip select for the respective Tag RAM block.
tag_ram_we[3:0]	Output	Tag RAM	Tag RAM write enable for the respective Tag RAM block.

Table A-12 Cache Tag RAM interface signals (continued)

Signal name	Direction	Connection	Description
tag_ram_addr[IW-1:0]	Output	Tag RAM	Address for Tag RAMs. This bus is connected to each RAM block. The upper index is cache-size dependent.
tag_ram_wdata[TDW-1:0]	Output	Tag RAM	Write data to the Tag RAMs. This bus is connected to each RAM block. The upper index is cache-size dependent.
tag_ram_rdata0[TDW-1:0]	Input	Tag RAM	Read data from Tag RAM block0. The upper index is cache-size dependent.
tag_ram_rdata1[TDW-1:0]	Input	Tag RAM	Read data from Tag RAM block1. The upper index is cache-size dependent.
tag_ram_rdata2[TDW-1:0]	Input	Tag RAM	Read data from Tag RAM block2. The upper index is cache-size dependent.
tag_ram_rdata3[TDW-1:0]	Input	Tag RAM	Read data from Tag RAM block3. The upper index is cache-size dependent.

A.7.3 Dirty RAM interface signals

The Cache Dirty RAM interface executes Dirty-RAM-related read or write requests.

For information about the *Index Width* (IW), *Data RAM Address Width* (DAW), and *Tag RAM Data Width* (TDW), see section 6.3.1. *RAM bus widths* in the *Arm® CoreLink™ AHB Cache Configuration and Integration Manual*.

Table A-13 Cache Dirty RAM interface

Signal name	Direction	Connection	Description
dirty_ram_cs	Output	Dirty RAM	Dirty RAM chip select
dirty_ram_bitwe[3:0]	Output	Dirty RAM	Dirty RAM bit-write enable
dirty_ram_addr[IW-1:0]	Output	Dirty RAM	Address to the Dirty RAM. The upper index is cache-size independent.
dirty_ram_wdata[3:0]	Output	Dirty RAM	Write data to the Dirty RAM.
dirty_ram_rdata[3:0]	Input	Dirty RAM	Read data from the Dirty RAM.

A.8 Configuration input ports

There are several configuration signals that you must use to configure AHB Cache behavior.

For more information about the configuration input ports, see the *Arm® CoreLink™ AHB Cache Configuration and Integration Manual*.

Table A-14 Configuration input ports

Name	Direction	Connection	Description
apb_violation_resp	Input	CPU or tie-off	The AHB Cache uses this signal to respond with an error to illegal operations on the APB interface. Illegal operations on the APB interface include: • Writing with no full write strobe • Accessing non-word aligned addresses • Instruction accesses • Non-privileged accesses
ahb_violation_resp	Input	CPU or tie-off	The AHB Cache uses this signal to control whether illegal access attempts to the cached data are responded with a bus error or silently ignored. Illegal accesses currently only include data type reads into XOM cached memory. The signal has no effect on responses that are returned from the downstream interface. **Note** Only present when `XOM=ON`.
power_on_enable	Input	CPU or tie-off	This signal enables the cache automatically after powerup.
dis_pwr_down_maint	Input	CPU, power controller, or tie-off	This signal turns off powerdown maintenance.
dis_cache_en_maint	Input	CPU, power controller, or tie-off	This signal turns off cache enable maintenance.
dis_cache_dis_maint	Input	CPU, power controller, or tie-off	This signal turns off cache disable maintenance.

Appendix B
Revisions

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

It contains the following section:

B.1 Revisions

This appendix describes changes between released issues of this book.

Table B-1 Issue 0000-01

Change	Location
First release.	-

Table B-2 Differences between issue 0000-01 and issue 0000-02

Change	Location
Added information about the AHB interface	• *2.2.2 Write-Through and Write-Back support* on page 2-26 • *2.2.3 Exclusive access sequences* on page 2-26
Added information about cache enable and cache disable maintenance	• *Cache enable maintenance* on page 3-41 • *Cache disable maintenance* on page 3-42
Added information about **power_on_enable**	*3.3.9 power_on_enable* on page 3-43
Added POWER_ON_ENABLE, DIS_PWR_DOWN_MAINT, DIS_CACHE_DIS_MAINT, and DIS_CACHE_EN_MAINT to HWPARAMS bit description	*4.4.1 HWPARAMS, hardware parameter register* on page 4-52
Added DENY_POWERDOWN to CTRL bit description	*4.4.2 CTRL, control register* on page 4-54
Added PIDR7, PIDR6, and PIDR5 registers	• *4.4.33 PIDR7, peripheral ID register 7* on page 4-86 • *4.4.32 PIDR6, peripheral ID register 6* on page 4-85 • *4.4.31 PIDR5, peripheral ID register 5* on page 4-84

Table B-3 Differences between issue 0000-02 and 0000-03

Change	Location
Added note about snapshotting during clock and power quiescence	*3.2.1 Snapshotting* on page 3-37
Added detail on cache disable maintenance	*Cache disable maintenance* on page 3-42
Added Chapter 5	*Chapter 5 Using software to program the AHB Cache* on page 5-95
Changed signal name from **pwakeup_s** to **pwakeup**	*A.5 APB interface signals* on page Appx-A-124