ARM® CoreLink™ CCI-500 Cache Coherent Interconnect

Revision: r0p1

Technical Reference Manual

ARM

ARM 100023_0001_00_en

ARM® CoreLink™ CCI-500 Cache Coherent Interconnect

Technical Reference Manual

Release Information

Document History

Issue	Date	Confidentiality	Change
0000-00	27 November 2014	Non-Confidential	First release for r0p0.
0000-01	19 December 2014	Confidential	Second release for r0p0.
0001-00	19 March 2015	Non-Confidential	First release for r0p1.

Non-Confidential Proprietary Notice

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

http://www.arm.com

Contents

ARM® CoreLink™ CCI-500 Cache Coherent Interconnect Technical Reference Manual

Preface

About this book 7
Feedback 10

Chapter 1 Introduction

1.1 About the CoreLink CCI-500 Cache Coherent Interconnect 1-12
1.2 Compliance 1-13
1.3 Features 1-14
1.4 Interfaces 1-15
1.5 CoreLink CCI-500 operation 1-16
1.6 Configurable options 1-17
1.7 Test features 1-18
1.8 Product design flow and documentation 1-19
1.9 Product revisions 1-21

Chapter 2 Functional Description

2.1 About the functions 2-23
2.2 Interfaces 2-24
2.3 Clocking and reset 2-27
2.4 Operation 2-28

Chapter 3 **Programmers Model**
3.1 About this programmers model 3-43
3.2 Register summary 3-44
3.3 Register descriptions 3-50
3.4 Address map 3-70

Appendix A **Signal Descriptions**
A.1 Clock and reset signals Appx-A-74
A.2 Power and clock control signals Appx-A-75
A.3 Configuration signals Appx-A-77
A.4 Debug signals Appx-A-79
A.5 DFT signals Appx-A-80
A.6 APB4 signals Appx-A-81
A.7 ACE and ACE-Lite slave interface signals Appx-A-82
A.8 AXI Master interface signals Appx-A-87
A.9 Miscellaneous signals Appx-A-90

Appendix B **Revisions**
B.1 Revisions Appx-B-92

Preface

This preface introduces the *ARM® CoreLink™ CCI-500 Cache Coherent Interconnect Technical Reference Manual*.

It contains the following:

- *About this book* on page 7.
- *Feedback* on page 10.

About this book

ARM CoreLink CCI-500 Cache Coherent Interconnect Technical Reference Manual (TRM). This documentation provides technical information for designers, integrators, and programmers who want to design or program an SoC that uses the CCI-500. It describes functionality, interfaces, registers, and signals.

Product revision status

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

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

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

Intended audience

This book is written for system designers, system integrators, and programmers who are designing or programming a System-on-Chip (SoC) that uses the CoreLink CCI-500 Cache Coherent Interconnect.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction
This chapter provides an overview of the CoreLink CCI-500 Cache Coherent Interconnect.

Chapter 2 Functional Description
This chapter describes the functionality of the CoreLink CCI-500 Cache Coherent Interconnect.

Chapter 3 Programmers Model
This chapter describes the programmers model of the CoreLink CCI-500 Cache Coherent Interconnect

Appendix A Signal Descriptions
This appendix describes the external signals of the CoreLink CCI-500 Cache Coherent Interconnect.

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.

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

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

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

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

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

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

Timing diagrams

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

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

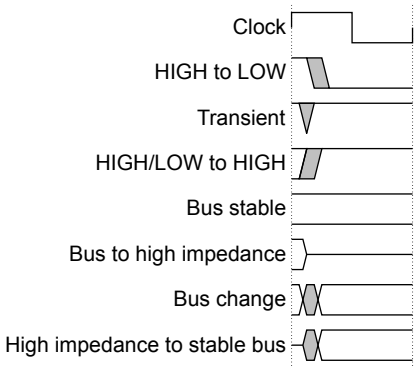

Figure 1 Key to timing diagram conventions

Signals

The signal conventions are:

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

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

Additional reading

This section lists publications by ARM and by third parties.

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

ARM publications

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

- *ARM® AMBA® AXI and ACE Protocol Specification* (ARM IHI 0022).
- *ARM® AMBA® APB Protocol Specification* (ARM IHI 0024).
- *Low Power Interface Specification, ARM® Q-Channel and P-Channel Interfaces* (ARM IHI 0068).
- *ARM® CoreSight™ Architecture Specification* (ARM IHI 0029).
- *Principles of ARM® Memory Maps* (ARM DEN 0001).

The following confidential books are only available to licensees:

- *ARM® CoreLink™ CCI-500 Cache Coherent Interconnect Configuration and Sign-off Guide* (ARM 100024).
- *ARM® CoreLink™ CCI-500 Cache Coherent Interconnect Integration Manual* (ARM 100025).

Other publications

This section lists relevant documents published by third parties:

- *JEDEC Standard Manufacturer's Identification Code, JEP106* http://www.jedec.org.

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.
- The number ARM 100023_0001_00_en.
- The page number(s) to which your comments refer.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

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

Chapter 1
Introduction

This chapter provides an overview of the CoreLink CCI-500 Cache Coherent Interconnect.

It contains the following sections:

- *1.1 About the CoreLink CCI-500 Cache Coherent Interconnect* on page 1-12.
- *1.2 Compliance* on page 1-13.
- *1.3 Features* on page 1-14.
- *1.4 Interfaces* on page 1-15.
- *1.5 CoreLink CCI-500 operation* on page 1-16.
- *1.6 Configurable options* on page 1-17.
- *1.7 Test features* on page 1-18.
- *1.8 Product design flow and documentation* on page 1-19.
- *1.9 Product revisions* on page 1-21.

1.1 About the CoreLink CCI-500 Cache Coherent Interconnect

The CCI-500 is a programmable high bandwidth interconnect that enables hardware-coherent systems.

Hardware-managed coherency can improve system performance and reduce system power by sharing on-chip data. Managing coherency in hardware has the following benefits:

- Reduces external memory accesses.
- Reduces the software overhead and complexity.
- Enables use of ARM big.LITTLE™ processing with multiple processor clusters.

The CCI-500 is a configurable interconnect that supports connectivity of:

- Up to four AMBA 4 ACE masters, such as the ARM Cortex®-A57 or Cortex-A53 processors.
- Up to six AMBA 4 ACE-Lite masters, such as the ARM Mali™-T760.
- Up to six AMBA 4 AXI4 slaves, such as memory and system peripherals.

Note

The CCI-500 permits combinations of ACE and ACE-Lite masters, up to a maximum total of seven masters.

The CCI-500 AXI4 master interfaces provide connection to memory and peripheral address space.

1.2 Compliance

The CCI-500 complies with the following specifications:

- *ARM® AMBA® AXI and ACE Protocol Specification*.
- *Low Power Interface Specification, ARM® Q-Channel and P-Channel Interfaces*.

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

1.3 Features

The CCI-500 features combine to provide a programmable high-bandwidth interconnect that enables hardware coherent systems.

The CCI-500 provides:

- Data coherency between ACE masters.
- *Input and Output* (IO) coherency with ACE-Lite masters.
- Crossbar interconnect functionality between the masters and up to six slaves.
- A snoop filter to reduce snoop power and improve performance for snoop misses.
- DVM message transport between masters for communication between MMUs.
- *Quality of Service* (QoS) features for traffic management.
- A *Performance Monitoring Unit* (PMU) to count performance-related events.
- Support for ARM TrustZone® to provide Secure, Non-secure, and protected states.
- A *Programmers View* (PV) to control coherency and interconnect functionality.

1.4 Interfaces

The CCI-500 has several interfaces to connect it to a wider system.

The CCI-500 is highly-configurable. You can select how many master and slave components to include in your system. The following figure shows an example CCI-500-based system.

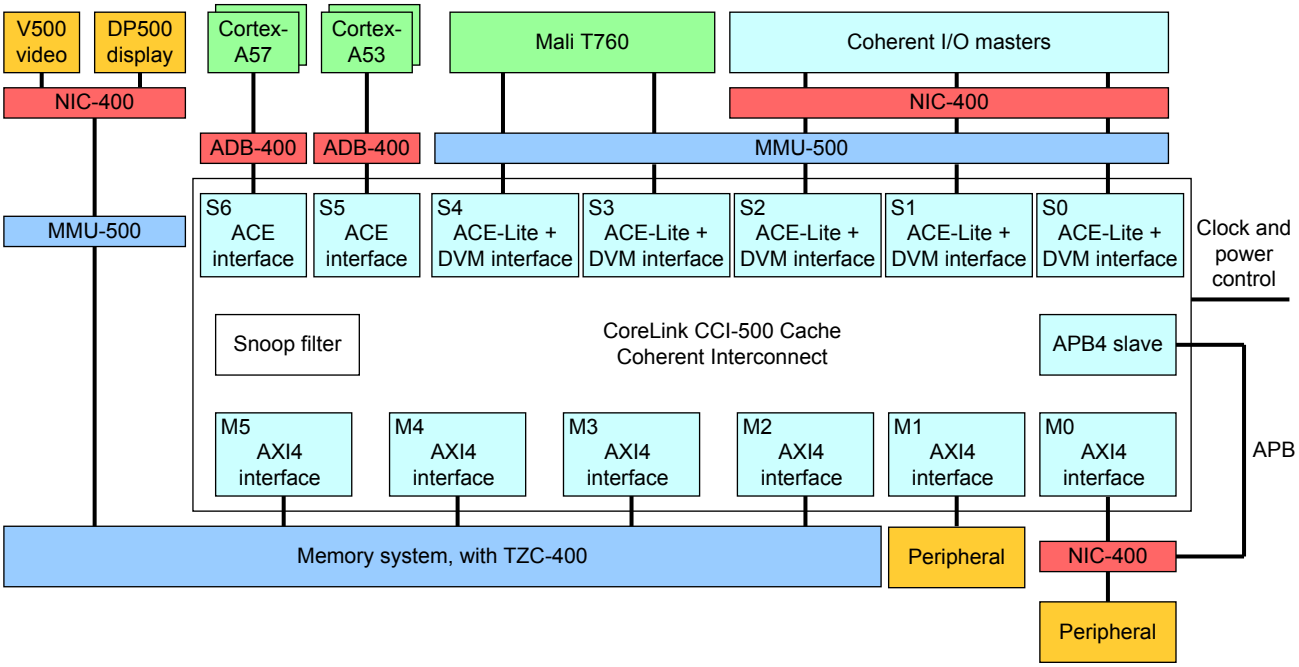

Figure 1-1 Example system with a CCI-500

In this example, slave interfaces S5 to S6 support the ACE protocol for connecting masters such as the Cortex-A53 or Cortex-A57 processors. The CCI-500 manages full coherency and data sharing between L1 and L2 caches of all connected processor clusters. Optionally, you can use the ADB-400 asynchronous bridge between components to integrate multiple power domains or clock domains.

Slave interfaces S0 to S4 support ACE-Lite and DVM signaling for connecting I/O coherent devices such as the Mali-T760 graphics unit. You can use DVM signaling for MMUs such as the MMU-500.

You can use the APB4 slave programming interface to program the CCI-500 registers.

Typically, up to four AXI4 master interfaces are connected to compatible memory controllers for LPDDR4 and LPDDR3 memory. Interfaces M5-M2 in the figure show these connections.

Typically, up to two AXI4 master interfaces are connected to system components, as shown by interfaces M1 and M0.

The clock and power control is achieved using the Q-Channel and the P-Channel.

1.5 CoreLink CCI-500 operation

The CCI-500 has dual-layer request channels, with a central *Transaction Tracker* (TT) that handles coherency and ordering. This is non-blocking and can re-order requests based on QoS requirements.

The following figure shows the CCI-500 high-level operation.

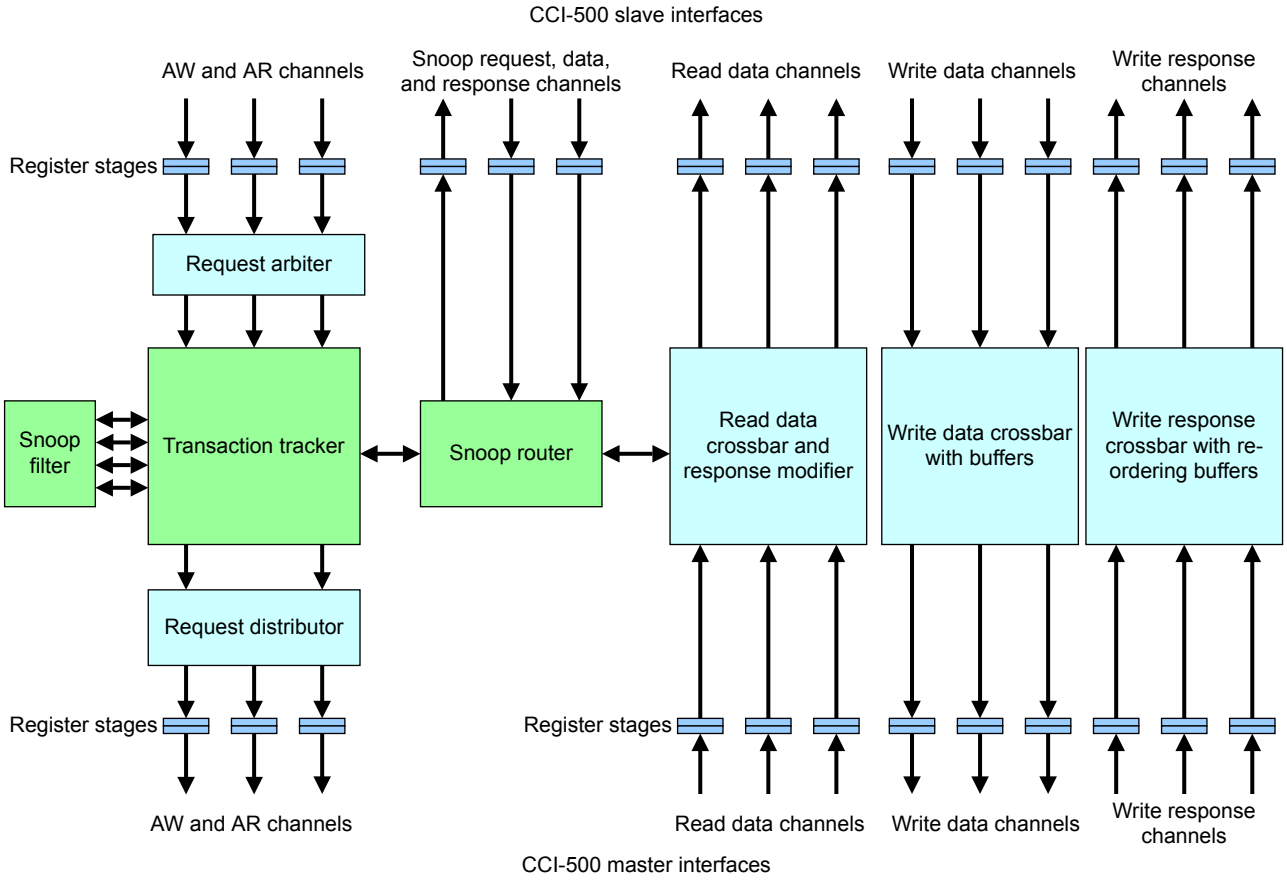

Figure 1-2 CCI-500 high-level operation

The TT uses a snoop filter to determine where to send snoop requests. To maximize throughput:

- The snoop filter has four partitions.
- The read data and write data interconnects are fully-connected crossbars.

Write responses also use a crossbar interconnect and the re-order buffer helps the CCI-500 to meet ordering requirements without stalling requests.

Each interface has a configurable number of register stages, with a minimum of one stage for each interface.

1.6 Configurable options

The CCI-500 is highly-configurable. Design-time configuration options enable you to meet your functional requirements with the smallest possible area and power.

Reset-time configuration options enable you to change the functionality of the interconnect for different applications.

When using the CCI-500 in your system, you can configure:

- The number of slave and master interfaces.
- The number of pipeline stages on interfaces to aid timing closure for large designs.
- Write buffering to achieve trade-off between area and write bandwidth.
- The size of the TT, to achieve trade-off between performance and area.
- QoS threshold, to define the transactions that are treated as high priority within the interconnect.
- Address widths.
- ID widths.
- Burst splitting option for slave interfaces.
- User-defined signal widths.
- The snoop filter RAM capacity to match connected processor cache sizes.
- The transport of data checksums, for example, parity or ECC.

The CCI-500 contains an address map that you can configure at reset-time. It includes options that you can use to interleave memory channels. Optionally, you can implement your own address decoder that defines any arbitrary addressing scheme. The CCI-500 includes a set of assertions that you can use with formal tools or in simulation to verify that your address decoder adheres to CCI-500 requirements.

1.7 Test features

The CCI-500 supports both scan cell insertion and MBIST methodologies for your SoC *Design For Test* (DFT) strategy. There are several additional signals to provide high coverage for your test strategy of the CCI-500 design and associated internal RAM cells.

The DFT control signals provide the following capabilities:

- Disabling of internal resets.
- Controlling architectural clock gating.
- Controlling of internal RAM chip-select control to preserve state.
- Controlling of internal RAM MBIST signals.
- Limiting of multi-cycle paths to enable delay testing.

See *A.5 DFT signals* on page Appx-A-80 for more information.

1.8 Product design flow and documentation

You must complete several processes to use the CCI-500. To obtain the best performance from the CCI-500, ARM recommends that you perform some of the implementation stages, including RAM integration, before integrating it into your wider SoC.

The processes you must perform are as follows:

Implementation
The implementer configures and synthesizes the RTL.

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

Final SoC implementation
The process of implementing the final, fully integrated SoC in silicon. ARM can provide only guidance relevant to its own products for this process. If ARM provides guidance on this process for your product, then a separate document is included in the implementation bundle for that product.

Programming
This is the last process. The system programmer develops the software required to configure and initialize the CCI-500, and tests the required application software.

For information on the CCI-500 documents that provide information on these processes, see *1.8.1 Documentation* on page 1-19.

Each process:

- Is separate, and a different person can complete it.
- Can include implementation and integration choices that affect the behavior and features of the CCI-500, and therefore the other tasks in the flow.

The operation of the final device depends on:

Build configuration
The implementer chooses the options that affect the preprocessing of the RTL source files. These options usually include or exclude the logic that affects one or more of the features, the area, or the maximum frequency and performance of the resulting macrocell. For example, the number of outstanding transactions that each master and slave interface supports.

Configuration inputs
The integrator configures some features of the CCI-500 by tying inputs to specific values. These configurations affect the start-up behavior before you specify the software configuration. They can also limit the options available to the software. For example, the **ACCHANNELENSx** signal inputs prevent AC coherency requests from being emitted from an unconnected slave interface.

Software configuration
The programmer configures the CCI-500 by programming particular values into registers. This affects the behavior of the CCI-500, for example, by enabling QoS features.

1.8.1 Documentation

Each CCI-500 document has an intended audience and is associated with specific tasks in the design flow. These documents do not reproduce ARM architecture and protocol information.

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

The CoreLink CCI-500 Cache Coherent Interconnect documentation is as follows:

Technical Reference Manual

The *Technical Reference Manual* (TRM) describes the functionality and the effects of functional options on the behavior of the CCI-500. It is required at all stages of the design flow. The choices made in the design flow can mean that some behaviors described in the TRM are not relevant. If you are programming the CCI-500, then contact:

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

Configuration and Sign-off Guide

The *Configuration and Sign-off Guide* (CSG) describes:

- A list of the design-time configuration options.
- The available build configuration options and related issues in selecting them.
- How to configure the *Register Transfer Level* (RTL) with the build configuration options.
- How to integrate RAM arrays.
- How to run test vectors.
- The processes to sign off the configured design.

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

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

Integration Manual

The *Integration Manual* (IM) describes how to integrate the CCI-500 into a SoC. It includes:

- A description of the CCI-500 features.
- A list of the reset-time configuration options.
- Considerations when integrating the CCI-500 into your system.

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

1.9 Product revisions

There can be differences in functionality between different product revisions. ARM records these differences in this section.

r0p0 First release.

r0p0-r0p1

The following changes apply to this release:

- Usage constraints and access permissions for PMCR is changed. See *3.3.5 Performance Monitor Control Register (PMCR)* on page 3-57.
- Usage constraints for Interface Monitor Control Register is changed. See *3.3.6 Interface Monitor Control Register* on page 3-58.
- Peripheral ID2 register value is changed to reflect the product status. See *3.3.7 Component and Peripheral ID Registers* on page 3-58.
- Support for connecting PCIe root complex components to the CCI-500 is added, through the following changes:
  - Added `SIx_W_MIN` parameter.
  - Added **MI_DEPENDENT_ON_SI_Mx** signal. See *A.3 Configuration signals* on page Appx-A-77.
  - Amended **ORDERED_WRITE_OBSERVATION[n:0]** functionality to remove the option to set Ordered Write Observations on ACE slave interfaces. See *A.3 Configuration signals* on page Appx-A-77.

Chapter 2
Functional Description

This chapter describes the functionality of the CoreLink CCI-500 Cache Coherent Interconnect.

It contains the following sections:

- *2.1 About the functions* on page 2-23.
- *2.2 Interfaces* on page 2-24.
- *2.3 Clocking and reset* on page 2-27.
- *2.4 Operation* on page 2-28.

2.1 About the functions

The CCI-500 is a coherent interconnect that enables hardware coherency. In hardware coherent systems, an operating system can run over multiple processor clusters without complicated cache maintenance software. This is a fundamental requirement for advanced ARM big.LITTLE processing models such as *Global Task Scheduling* (GTS).

In addition to the AXI and ACE interfaces, the CCI-500 provides interfaces that you can use for various system operations, such as:

- Programming the CCI-500 internal registers, debugging, and performance monitoring using an APB4 interface.
- Controlling clock and power states with P-Channel and Q-Channel to minimize power at low bandwidth.
- Logic and RAM testing, for manufacture test.

The CCI-500 includes snoop functionality that permits snooping of the ACE interfaces. A snoop filter provides efficient snoop transaction management by keeping a record of the addresses stored in the caches of the attached ACE masters. This means that the snoop filter can often resolve coherency messaging instead of broadcasting to all ACE interfaces. This can offer system power savings and reduce the latency in the case where data is not held in any of the upstream caches.

A *Performance Monitoring Unit* (PMU) provides events and counters that indicate CCI-500 run-time performance. Additional registers provide information on the current status of the interconnect and you can use these to help debug system deadlock. In addition, the CCI-500 provides a set of QoS regulation and control mechanisms.

The CCI-500 supports Secure and Non-secure operations that can be used within a system that uses ARM TrustZone to provide Secure, Non-secure, and protected states.

The CCI-500 also supports cache maintenance operations and exclusive accesses.

2.2 Interfaces

The CCI-500 has several interfaces to connect it to a wider system.

The following figure shows the CCI-500 interfaces.

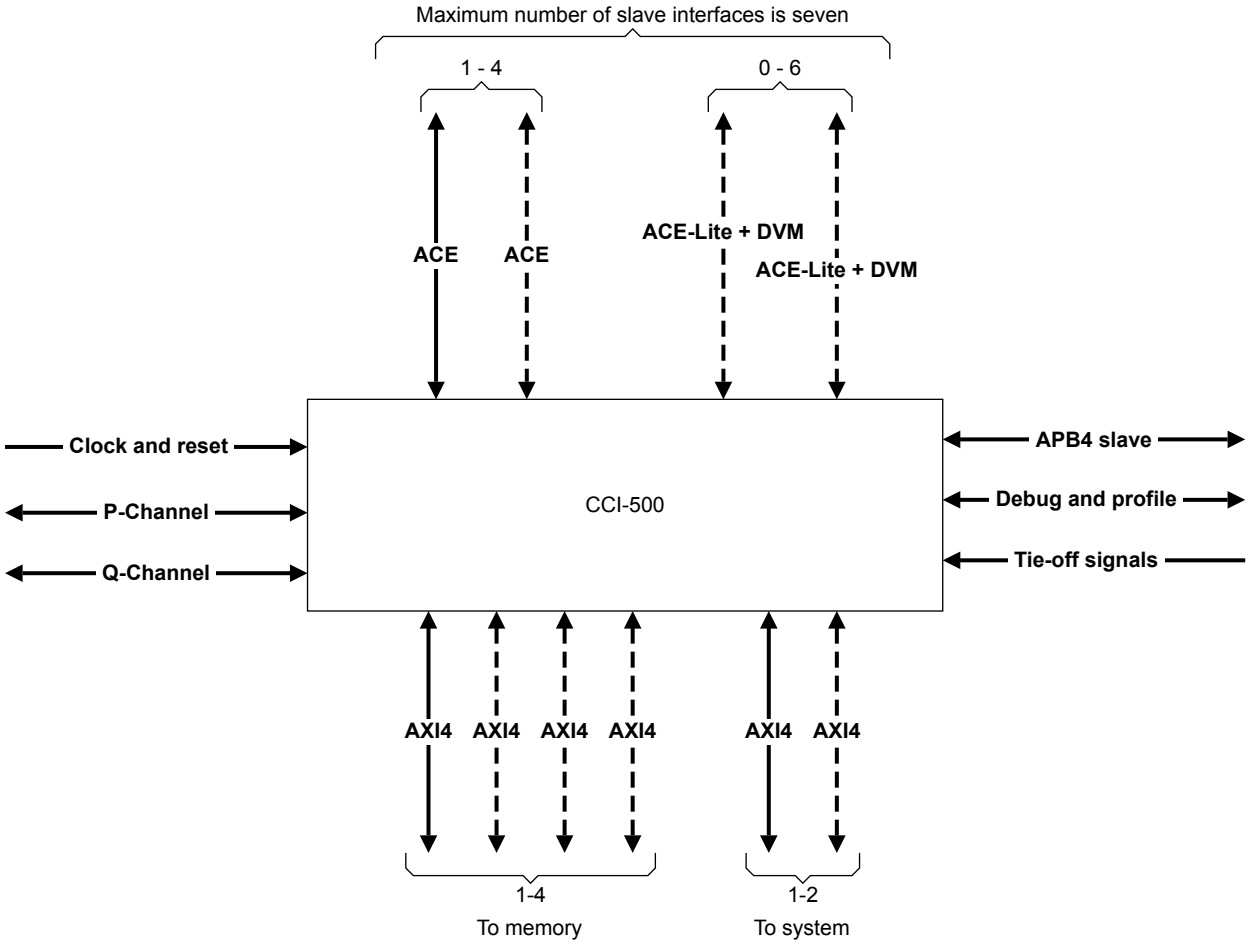

Figure 2-1 CCI-500 interfaces

All master and slave interfaces are numbered from 0, and the following table shows how many interfaces the CCI-500 can have.

Note

You can have a minimum of two and a maximum of seven slave interfaces.

Table 2-1 Permitted CCI-500 interfaces

Interface type	Number of interfaces permitted by the CCI-500	
	Minimum	**Maximum**
ACE slave.	1	4
ACE-Lite + DVM slave.	0	6
AXI4 master, to memory.	1	4
AXI4 master, to system.	1	2

This section contains the following subsections:

- *2.2.1 ACE interfaces* on page 2-25.
- *2.2.2 ACE-Lite slave interfaces* on page 2-25.
- *2.2.3 AXI4 master interfaces* on page 2-25.
- *2.2.4 APB slave interface* on page 2-25.
- *2.2.5 Clock and power control interfaces* on page 2-25.
- *2.2.6 Debug and performance monitoring interface* on page 2-26.
- *2.2.7 DFT interface* on page 2-26.

2.2.1 ACE interfaces

You can configure the CCI-500 to have up to four standard *AXI Coherency Extensions* (ACE) slave interfaces.

The CCI-500 supports the full ACE protocol, with a coherency granule of 64-bytes. For more information about how the CCI-500 handles snoop transactions, see *2.4.3 Snoop connectivity and control* on page 2-29.

See the *ARM® AMBA® AXI and ACE Protocol Specification* for more information.

2.2.2 ACE-Lite slave interfaces

The ACE-Lite interfaces are a defined subset of the full ACE interfaces.

You can configure the CCI-500 to have up to six ACE-Lite slave interfaces. In addition to standard ACE-Lite functionality, these interfaces also support DVM messages that are passed to upstream system MMU components.

See the *ARM® AMBA® AXI and ACE Protocol Specification*.

2.2.3 AXI4 master interfaces

The AMBA 4 protocol defines the AXI4 protocol that supports high-performance, high-frequency system designs.

Depending on configuration, the CCI-500 includes:

- Between one and four AXI4 master interfaces for connecting to memory.
- One or two AXI4 master interfaces for connecting to the rest of the system.

The primary difference between memory and system interfaces is the definition in the memory map. You can configure memory interfaces to be interleaved across the memory region, for example striping, to enable higher utilization of memory. See the *ARM® AMBA® AXI and ACE Protocol Specification* for more information about AXI4.

2.2.4 APB slave interface

APB is a low-cost AMBA bus protocol that can reduce power consumption when connecting to the main system bus.

The CCI-500 has an APB slave interface for programming the internal registers and reading from the status, PMU and debug registers. This interface runs synchronously with the other CCI-500 interfaces.

2.2.5 Clock and power control interfaces

The CCI-500 includes the ARM Q-Channel and P-Channel interfaces to control clock and power states.

To save power, the CCI-500 has:

- A Q-Channel interface that you can use to determine how to control the clock state of devices.
- A P-Channel interface that you can use to control the power state of the snoop filter RAMs within the CCI-500, for example, enabling a retention state.

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

2.2.6 Debug and performance monitoring interface

The CCI-500 supports debug and performance monitoring using a combination of standard signals and a dedicated error pin.

The CCI-500 includes standard ARM interface signals for:

- Debug configuration.
- Event outputs.
- PMU counter overflow interrupts.

In addition, the **nERRIRQ** output pin indicates a transaction error that cannot be signaled precisely. See *2.4.7 Error responses* on page 2-38 for more information.

2.2.7 DFT interface

The CCI-500 incorporates testing interfaces.

For in-silicon testing of logic and RAMs, the CCI-500 includes DFT and MBIST interfaces. See *A. 5 DFT signals* on page Appx-A-80 for more information.

2.3 Clocking and reset

The CCI-500 uses **ACLK** and **ARESETn** signals for clock and reset, respectively.

The CCI-500 has a single main clock signal, **ACLK**, that it distributes to all sub-blocks. You must use external clock-domain-crossing bridges when masters and slaves connecting to the CCI-500 are in different clock domains.

The CCI-500 has a single reset domain with an active-LOW reset input signal, **ARESETn**. This is synchronized with the **ACLK** with a double-register on the input to the CCI-500 signal. You can replace this synchronizer with appropriate cells from your target library.

——— **Note** ———

You must ensure that there is no activity on the slave interfaces, and the configuration inputs must be static for at least three **ACLK** cycles after the **ARESETn** signal goes HIGH.

———————————

The CCI-500 supports the following power saving features:

Internal regional clock gating
The CCI-500 automatically gates the clock to internal blocks that do not require the clock.

External architectural clock gating
The CCI-500 provides signaling to support implementation of your architectural clock gating strategy. The CCI-500 provides this support using the ARM Q-Channel.

When the Q-Channel is in the Q_STOPPED state, you can safely disable the clock to the CCI-500. Any incoming transactions are stalled, and any changes to other inputs are not registered until the clock is reapplied. This clock gating can reduce dynamic power to zero when the system is idle.

External architectural power state control
The CCI-500 provides signaling to support implementing your architectural power state control. The CCI-500 provides this support using the ARM P-Channel.

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

——— **Note** ———

The Q-Channel and P-Channel supersede the AXI low-power interface that products such as the CCI-400 use. For legacy IP, the *Low Power Interface Specification, ARM® Q-Channel and P-Channel Interfaces* provides information on linking to IP that contains an AXI low-power interface.

———————————

2.4 Operation

This section groups information based on the operation of various features of the CCI-500.

This section contains the following subsections:

- *2.4.1 Connectivity and address map* on page 2-28.
- *2.4.2 Snoop filter* on page 2-28.
- *2.4.3 Snoop connectivity and control* on page 2-29.
- *2.4.4 Performance Monitoring Unit* on page 2-30.
- *2.4.5 In-silicon debug features* on page 2-37.
- *2.4.6 Security* on page 2-37.
- *2.4.7 Error responses* on page 2-38.
- *2.4.8 Cache maintenance operations* on page 2-39.
- *2.4.9 Barriers* on page 2-39.
- *2.4.10 Exclusive accesses* on page 2-39.
- *2.4.11 DVM messages* on page 2-39.
- *2.4.12 Quality of Service* on page 2-40.

2.4.1 Connectivity and address map

The interconnect topology and the address map are factors that can affect whether a particular master can communicate with a particular slave.

The CCI-500 is a fully-connected interconnect, meaning that any master can communicate with any slave, subject to the address map you define. See your SoC documentation for the address map that your implementation supports.

2.4.2 Snoop filter

The CCI-500 contains an inclusive snoop filter that records the addresses of data stored in the ACE master caches. This means that the filter can respond to the snoop in the case of a miss, and snoop appropriate masters only in the case of a hit. Snoop filter entries are maintained by observing transactions from ACE masters to determine when entries have to be allocated and deallocated.

The snoop filter can respond to many of the coherency requests without it being necessary to broadcast to all ACE interfaces. For example, if the address is not in any cache, the snoop filter responds with a miss and directs the request to memory. If the address is in a processor cache, then it is considered to be a hit and the snoop is directed to the appropriate ACE port containing that address in its cache.

ARM recommends that you configure the snoop filter directory to be 0.75-1 times the total size of exclusive caches of processors attached to the CCI-500. The snoop filter is 8-way set associative and, to minimize conflicts, stores twice as many tags as the configured size. An example of a conflict is when the CCI-500 is unable to insert a new entry in an available position in the snoop filter. In the case of a conflict, an existing entry is evicted, and the snoop filter issues a CleanInvalid snoop to the processors that might be holding the evicted lines. This is known as a back-invalidation, and is expected to be a rare occurrence if you configure the snoop filter size as ARM recommends.

The snoop filter is updated by monitoring transactions from the attached masters, that allocate and deallocate data into their caches. In the ACE protocol, the deallocation of clean data is indicated using the Evict transaction.

Note

You must ensure that masters connected to the CCI-500 issue Evict transactions when they deallocate clean data. For ARM processors, you can control the issuing of Evict transactions using bit[3] of the L2 Auxiliary Control Register.

2.4.3 Snoop connectivity and control

The CCI-500 has a fully-connected snoop interconnect and a snoop filter for efficient management of snoop request transactions.

You can control whether each interface is enabled for snoop requests and DVM message requests using the *3.3.8 Snoop Control Registers* on page 3-59.

A shareable read request from an ACE master, that allocates data to the cache of the master, also allocates an entry in the snoop filter to record that the master has a copy of that data.

For requests for which it might be necessary to retrieve or invalidate data in the cache of another master, the CCI-500 looks up the address in the snoop filter. If the snoop filter indicates that a master has a copy of that data, then either:

- A snoop request is issued, if snoops to the master are enabled.
- The snoop filter entry is updated, if snoops to the master are disabled.

If the snoop filter indicates that no ACE master contains that address, then the request is directed to the appropriate master interface. DVM requests are broadcast through all slave interfaces that are enabled for DVM messages and do not interact with the snoop filter.

The programmable bits of the *3.3.8 Snoop Control Registers* on page 3-59 are LOW at reset. You must program them HIGH for each master in the shareable domain before the CCI-500 receives shareable transactions or DVM messages. Before disabling a master, you must disable snoop and DVM messages for the master by programming the relevant bits of the *3.3.8 Snoop Control Registers* on page 3-59 LOW.

If snoops are sent to interfaces where the master is disabled or not present, the system is likely to deadlock. A hardware mechanism of disabling snoops is provided to prevent software errors causing deadlocks in cases where masters are not present or do not support DVMs. Each slave interface has an **ACCHANNELENSx** signal input that controls whether snoops and DVM messages can be issued from that interface. This input overrides any programmable settings.

Note

These bits are sampled only at reset, and changing them after **ARESETn** is HIGH has no effect.

Removing a master from the coherent domain

To ensure correct system operation, you must follow a specific process to remove the master from the CCI-500 coherent domain before powering down your master.

Several steps in this process require you to take action on the processor that you want to power down. For these steps, see the appropriate processor documentation.

Procedure

1. Configure the master so that it does not allocate shareable data into its cache, for example by disabling the data cache.
2. Clean and invalidate all shareable data from the caches in the master.
3. Program the Snoop Control Register to prevent the CCI-500 from sending snoops or DVM messages to the master.

   See *3.3.8 Snoop Control Registers* on page 3-59 for more information.
4. Execute a barrier instruction to ensure that the previous step is complete.
5. Poll the Status Register to confirm that the Snoop Control Register changes are effected.

   See *3.3.3 Status Register* on page 3-52 for more information.

Postrequisites

After you complete these actions, the master is no longer in the coherent domain and you can power it down or disable it.

Adding a master to the coherent domain

To ensure correct system operation, you must follow a specific process to add a master to the CCI-500 coherent domain.

Before a master allocates any shareable data into its caches, you must add it to the CCI-500 coherent domain.

Procedure

1. Enable the master to respond to snoops.
2. Program the Snoop Control Register to enable snoops to the master being added.

   See *3.3.8 Snoop Control Registers* on page 3-59 for more information.
3. Execute a barrier instruction to ensure that the previous step is complete.
4. Poll the *3.3.3 Status Register* on page 3-52 to confirm that the Snoop Control Register changes are effected.
5. Configure the master so that it can issue cacheable, shareable transactions.

2.4.4 Performance Monitoring Unit

The PMU events and counters indicate the run-time performance of the CCI-500.

The CCI-500 includes logic to gather various statistics on the operation of the interconnect during run-time, using events and counters. These events provide useful information about the behavior of the interconnect to use when debugging or profiling traffic.

The PMU provides eight counters. Each counter can count any of the events available in the CCI-500. To keep the PMU logic overhead to a minimum, the absolute count and timing of events might vary slightly. This has a negligible effect except when the counters are enabled for a very short time.

The PMU consists of:

- Performance event counters that are readable through the internal registers.
- A global start or stop bit that enables the counters to increment when HIGH. The default is LOW.
- A global reset bit that resets all counters to zero.
- A parallel event bus, **EVNTBUS**, that you can export from the CCI-500 to capture all events concurrently.
- Eight 32-bit event counters that you can program to count an event from the event bus.
- Input signals **DGBEN** and **NIDEN**. If either is HIGH, the counting and exporting of events is enabled.
- Input signals **DGBEN** and **SPIDEN**, that enable the counting of both Non-secure and Secure events.
- A set of counter overflow outputs, **nEVNTCNTOVERFLOW**, that can raise an interrupt when a number of events have occurred.

The PMU obeys the following rules:

- Each master and slave interface emits events separately from any other interface.
- The snoop filter emits events separately from any interface.
- Events that are marked ACE only, can only fire for ACE interfaces.
- Each event can only fire once per cycle.

This section describes:

- *PMU event list* on page 2-31.
- *PMU registers* on page 2-35.
- *Using the PMU* on page 2-35.

PMU event list

The CCI-500 can generate a wide range of events, attributed to a specific interface or globally where they apply to central functions. A list of these events and interface identifiers enables you to identify and then program the events and source locations you want to monitor.

To program the CCI-500 use the code column in each respective table to identify the value to program in to each register field. If you monitor events using the **EVNTBUS**, then use the **EVNTBUS** offset column to identify each position of the bit.

Each event has a 9-bit configuration identifier comprising a source identifier and an event code concatenated `{identifier,code}`. The source identifier is a 4-bit code that indicates the interface that generated the 5-bit event code.

The following table shows the possible 4-bit source identifiers.

Table 2-2 Event source identifiers

Code[8:5]	Source
`0x0`	Slave interface 0, SI0
`0x1`	Slave interface 1, SI1
`0x2`	Slave interface 2, SI2
`0x3`	Slave interface 3, SI3
`0x4`	Slave interface 4, SI4
`0x5`	Slave interface 5, SI5
`0x6`	Slave interface 6, SI6
`0x7`	Reserved
`0x8`	Master interface 0, MI0
`0x9`	Master interface 1, MI1
`0xA`	Master interface 2, MI2
`0xB`	Master interface 3, MI3
`0xC`	Master interface 4, MI4
`0xD`	Master interface 5, MI5
`0xE`	Reserved
`0xF`	Global

Note

As CCI-500 is a configurable product not all interfaces might be present, but the source encodings remain the same. If you select an interface that is not present in the specific implementation, then no events are generated.

The following tables show the 5-bit event codes for slave interfaces, master interfaces, and global events.

- *Table 2-3 Slave interface event codes* on page 2-32.
- *Table 2-4 Master interface event codes* on page 2-34.
- *Table 2-5 Event codes for global events* on page 2-34.

Table 2-3 Slave interface event codes

Slave event	Code[4:0]	EVNTBUS offset	Secure exempt	ACE only
Read request handshake, where both: • **ARVALID** is HIGH. • **ARREADY** is HIGH.	0x00	0	-	-
Read request handshake: device.	0x01	1	-	-
Read request handshake: normal, non-shareable.	0x02	2	-	-
Read request handshake: normal, shareable, non-allocating. This applies to ReadOnce transactions.	0x03	3	-	-
Read request handshake: normal, shareable allocating. This applies to ReadClean, ReadShared, ReadNotSharedDirty, and ReadUnique transactions.	0x04	4	-	Y
Read request handshake: invalidation. This applies to MakeUnique and CleanUnique transactions.	0x05	5	-	Y
Read request handshake: cache maintenance operation. This applies to CleanInvalid, MakeInvalid, and CleanShared transactions.	0x06	6	-	-
Read request handshake: DVM. This applies to DVM Message and DVM Complete transactions.	0x07	7	-	-
Read data handshake, where both: • **RVALID** is HIGH. • **RREADY** is HIGH.	0x08	8	Y	-
Read data handshake with **RLAST** HIGH, for a snoop hit.	0x09	9	Y	-
Write request handshake, where both: • **AWVALID** is HIGH. • **AWREADY** is HIGH.	0x0A	10	-	-
Write request handshake: device.	0x0B	11	-	-
Write request handshake: non-shareable.	0x0C	12	-	-
Write request handshake: shareable. This applies to WriteBack and WriteClean transactions.	0x0D	13	-	Y
Write request handshake: shareable. This applies to WriteLineUnique transactions.	0x0E	14	-	-
Write request handshake: shareable. This applies to WriteUnique transactions.	0x0F	15	-	-
Write request handshake. This applies to Evict transactions.	0x10	16	-	Y

Table 2-3 Slave interface event codes (continued)

Slave event	Code[4:0]	EVNTBUS offset	Secure exempt	ACE only
Write request handshake. **Note** This applies to WriteEvict transactions. However, because WriteEvict is not supported in the CCI-500, this event does not fire.	0x11	17	-	Y
Write data handshake, where both: • **WVALID** is HIGH. • **WREADY** is HIGH.	0x12	18	Y	-
Snoop request handshake, where both: • **ACVALID** is HIGH. • **ACREADY** is HIGH.	0x13	19	-	-
Snoop request handshake: read. This applies to ReadOnce, ReadClean, ReadNotSharedDirty, ReadShared, and ReadUnique transactions.	0x14	20	-	Y
Snoop request handshake: clean or invalidate. This applies to MakeInvalid, CleanInvalid, and CleanShared transactions.	0x15	21	-	Y
Snoop response handshake: Data Transfer bit, indicated by **CRRESP[0]** LOW.	0x16	22	Y	-
Read request stall, where both: • **ARVALID** is HIGH. • **ARREADY** is LOW.	0x17	23	-	-
Read data stall, where both: • **RVALID** is HIGH. • **RREADY** is LOW.	0x18	24	Y	-
Write request stall, where both: • **AWVALID** is HIGH. • **AWREADY** is LOW.	0x19	25	-	-
Write data stall, where both: • **WVALID** is HIGH. • **WREADY** is LOW.	0x1A	26	Y	-
Write response stall, where both: • **BVALID** is HIGH. • **BREADY** is LOW.	0x1B	27	Y	-
Snoop request stall, where both: • **ACVALID** is HIGH. • **ACREADY** is LOW.	0x1C	28	-	-
Snoop data stall, where both: • **CDVALID** is HIGH. • **CDREADY** is LOW.	0x1D	29	Y	Y

Table 2-3 Slave interface event codes (continued)

Slave event	Code[4:0]	EVNTBUS offset	Secure exempt	ACE only
Request stall cycle because of OT transaction limit.	0x1E	30	-	-
Read stall because of arbitration.	0x1F	31	-	-

The following table shows the event codes for master interfaces.

Table 2-4 Master interface event codes

Master event	Code[4:0]	EVNTBUS offset	Secure exempt
Read data handshake.	0x00	0	Y
Write data handshake.	0x01	1	Y
Read request stall, where both: • **ARVALID** is HIGH. • **ARREADY** is LOW.	0x02	2	-
Read data stall, where both: • **RVALID** is HIGH. • **RREADY** is LOW.	0x03	3	Y
Write request stall, where both: • **AWVALID** is HIGH. • **AWREADY** is LOW.	0x04	4	-
Write data stall, where both: • **WVALID** is HIGH. • **WREADY** is LOW.	0x05	5	Y
Write response stall, where both: • **BVALID** is HIGH. • **BREADY** is LOW.	0x06	6	Y

The following table shows the event codes for global events.

Table 2-5 Event codes for global events

Global event	Code[4:0]	EVNTBUS offset	Secure exempt
Access to snoop filter bank 0 or 1, any response.	0x00	0	-
Access to snoop filter bank 2 or 3, any response.	0x01	1	-
Access to snoop filter bank 4 or 5, any response.	0x02	2	-
Access to snoop filter bank 6 or 7, any response.	0x03	3	-
Access to snoop filter bank 0 or 1, miss response.	0x04	4	-
Access to snoop filter bank 2 or 3, miss response.	0x05	5	-
Access to snoop filter bank 4 or 5, miss response.	0x06	6	-
Access to snoop filter bank 6 or 7, miss response.	0x07	7	-
Back-invalidation from snoop filter.	0x08	8	-
Requests that allocate into a snoop filter bank might be stalled because all ways are used. The snoop filter RAM might be too small.	0x09	9	Y

Table 2-5 Event codes for global events (continued)

Global event	Code[4:0]	EVNTBUS offset	Secure exempt
Stall because TT full. Increase `TT_DEPTH` parameter to avoid performance degradation.	`0x0A`	10	-
CCI-generated write request.	`0x0B`	11	-
CD handshake in snoop network. Each event corresponds to 16 bytes of snoop data. Use this to measure snoop data bandwidth.	`0x0C`	12	Y
Request stall because of address hazard.	`0x0D`	13	-
Snoop request stall because of snoop TT being full.	`0x0E`	14	Y
Snoop request type override for TZMP1 protection.	`0x0F`	15	Y

Event bus

The CCI-500 exports a vector of event signals providing information from the *Performance Monitor Unit* (PMU) using the **EVNTBUS** signal. The width of this bus varies depending on the number of master and slave interfaces in your CCI-500 implementation.

The **EVNTBUS** output is a concatenation of all events that is, global events and events on each *Master Interface* (MI) and *Slave Interface* (SI). The global events are always the least significant bits from [14:0] irrespective of the number of interfaces. The bit offsets in the **EVNTBUS** output can be found in *Table 2-5 Event codes for global events* on page 2-34.

Note

By default, only events for Non-secure transactions are recorded. However, if the **SPNIDEN** input signal is HIGH, or if both **DBGEN** and **SPIDEN** inputs are HIGH, then the CCI-500 counts and exports both Secure and Non-secure events. Events marked in the tables as Secure exempt do not have a security classification, so they are counted and exported in either case.

PMU registers

The CCI-500 contains the following performance-related registers:

- *3.3.13 Event Select Registers* on page 3-64.
- *3.3.14 Event Count Registers* on page 3-65.
- *3.3.15 Count Control Registers* on page 3-65.

Using the PMU

You can run performance and monitor tests to check the CCI-500 performance.

For each performance and monitor test that you run, you can:

- Select a maximum of eight events to monitor during the test.
- Read the value of each event counter at the end of the test.
- Detect counter overflows.

Note

The CCI-500 PMU does not include a clock counter because the clock can be disabled to save power. To make time-related measurements, you must use another system timer, for example, the clock counter in the processor PMU.

Use the following registers to set up your test, and to monitor each event:

- Event Select Register to select the event.
- Event Count Register to indicate how many events occur.
- Event Counter Control Register to enable or disable the event counter.
- Event Overflow Flag Status Register to detect the event counter overflow.

Example of how to use the PMU

Use the following example to run a test scenario and show how to use the PMU to measure the snoop hit rate for shareable read requests for one ACE master and one ACE-Lite master.

In this example, it is assumed that the ACE master is connected to slave interface 3 and the ACE-Lite master is connected to slave interface 2.

Procedure

1. Set up the performance counters as follows:
   a. Program the Event Select Registers as follows:
      - Program the counter 0 register to count shareable, non-allocating read requests through slave interface 3:
        — Program bits[8:5] to `0x3` to select slave interface 3.
        — Program bits[4:0] to `0x03` to select the event for Read request handshake: normal, shareable, non-allocating.
      - Program the counter 1 register to count shareable, allocating read requests through slave interface 3:
        — Program bits[8:5] to `0x3` to select slave interface 3.
        — Program bits[4:0] to `0x04` to select the event for Read request handshake: normal, shareable, non-allocating.
      - Program the counter 2 register to count slave interface 3 snoop hits:
        — Program bits[8:5] to `0x3`.
        — Program bits[4:0] to `0x09`.
      - Program the counter 3 register to count shareable non-allocating read requests through slave interface 2:
        — Program bits[8:5] to `0x2`.
        — Program bits[4:0] to `0x03`.
      - Program the counter 4 register to count slave interface 2 snoop hits:
        — Program bits[8:5] to `0x2`.
        — Program bits[4:0] to `0x09`.
   b. Enable the event counters by programming the Count Control Registers as follows:
      - Set counter 0 register bit[0] to 1.
      - Set counter 1 register bit[0] to 1.
      - Set counter 2 register bit[0] to 1.
      - Set counter 3 register bit[0] to 1.
      - Set counter 4 register bit[0] to 1.
2. Ensure that the **NIDEN** and **SPNIDEN** input are HIGH.
3. Program the following bits in the *Performance Monitor Control Register* (PMCR):
   - Program bit[1] to 1 to reset event counters.
   - Program bit[0] to 1 to enable all counters.
4. Permit the test to run for an appropriate amount of time.
5. Program the PMCR bit[0] to 0 to disable all counters to stop the test:
6. Read the results of the test from the event counters:
   - Counter 0 and 1 hold the number of shareable reads for slave interface 3.
   - Counter 2 holds the number of snoop hits for slave interface 3.

- Counter 3 holds the number of shareable reads for slave interface 4.
- Counter 4 holds the number of snoop hits for slave interface 4.

7. Check the overflow bits of all counters and adjust your results accordingly.

2.4.5 In-silicon debug features

The CCI-500 has monitors on all slave and master interfaces that you can use to observe interface status. Each monitor records the number of outstanding read, write, and snoop transactions. It also records the status of the handshake signal from each channel.

This feature can be helpful in the case of a deadlock, for example, the monitors can help to determine outstanding transactions or where back-pressure is being applied.

The monitors are situated inside the outermost registers of the CCI-500, meaning that the values indicated are affected by the numbers of pipeline stages configured in a specific implementation.

Note

If the debug registers are accessed through the CCI-500, you might not be able to read the registers in the case of a deadlock.

2.4.6 Security

To build a system based on the Secure and Non-secure capabilities that ARM TrustZone technology provides, you must consider the following security issues.

This section describes:

- *Security status of the internal programmers view* on page 2-37.
- *Making a non-TrustZone aware master Secure* on page 2-37.
- *Security of master interfaces* on page 2-37.
- *Security considerations for the PMU* on page 2-38.

Security status of the internal programmers view

You can configure the programmers view of the CCI-500 for access by Secure or Non-secure requests.

With the exception of the PMU registers, the programmers view defaults to Secure access only, as follows:

- Non-secure reads of Secure registers receive an error response and zeroed data.
- Non-secure writes to Secure registers receive an error response and are *Write-Ignored* (WI).

You can change the security model by writing to the *3.3.2 Secure Access Register* on page 3-51. This enables Non-secure access to all registers except the *3.3.1 Control Override Register* on page 3-50 and the *3.3.2 Secure Access Register* on page 3-51. You can also make the PMU registers accessible to Secure requests only.

Making a non-TrustZone aware master Secure

For a master that is not TrustZone-aware, you can tie the **ARPROT[1]** and **AWPROT[1]** input signals LOW to place it permanently in the Secure domain. This means that the master can access Secure data in the caches of the ACE masters and Secure registers in the CCI-500, so the resulting system might not be secure under all circumstances.

Security of master interfaces

Transactions from the CCI-500 master interfaces always retain the security setting of the originating transactions.

The security settings of the originating transactions apply to:

- Non-shareable transactions.
- Shareable transactions that miss in the snoop filter or receive a snoop miss response.
- Writes generated by the CCI-500.

Security considerations for the PMU

You can configure the PMU to count only Non-secure events or both Secure and Non-secure events, depending on the **SPNIDEN**, **SPIDEN** and **DBGEN** input signals.

For more information on configuring, and the input signals **SPNIDEN**, **SPIDEN** and **DBGEN**, see the *2.4.4 Performance Monitoring Unit* on page 2-30 section.

If you configure the PMU to count both Secure and Non-secure events, then there is a potential security risk because Non-secure software can observe Secure activity through the performance counters. ARM recommends that you consider the security to be breached for devices placed in this state and take appropriate action.

If the PMU changes from counting all events to counting only Non-secure events, the counters can contain information relating to Secure transactions. Therefore, ARM recommends that the software sets the event counters to zero after changing the configuration to avoid a potential security risk.

——— **Note** ———

Unlike ARM processors, the CCI-500 makes no distinction between events from user or privileged transactions.

———————————

Support for TrustZone Media Protection

In systems that require hardware protection of media data, you can configure the CCI-500 to support ARM TZMP1.

To differentiate between Protected and Non-Trusted entities, ARM defines 16 states that mark all processes within hardware and software. These states are defined using the *Non-secure Access ID* (NSAID), and each initiating device in the SoC has one or more NSAID values assigned in hardware. The NSAID enables other components to identify the initiating device for a particular transaction, and to identify whether the device is treated as Non-protected and therefore permitted to read data from other Non-protected masters.

2.4.7 Error responses

The CCI-500 uses a combination of precise and imprecise error responses. Precise errors are signaled on the response to the request that caused the error. With the exception of DVM or Evict requests, for accesses to regions that are not mapped in the address decoder, the CCI-500 generates a DECERR response and any snoops, or snoop filter updates, are suppressed. The address map is implementation specific. See your platform documentation for more information.

A snoop error response to a CleanInvalid, CleanShared, or MakeInvalid transaction generates a SLVERR response to the originating device.

There are certain circumstances when it is not possible to signal an error precisely. In these cases, the CCI-500 signals an error imprecisely, using the **nERRIRQ** output pin. You can identify the interface that received the error response by reading the *3.3.4 Imprecise Error Register* on page 3-54.

The following table shows the errors that are signaled as imprecise. All other sources of error are signaled precisely.

——— **Note** ———

An error is signaled either precisely or imprecisely, but never both.

———————————

Table 2-6 Imprecise errors

Error condition	Channel receiving error	Imprecise error indicator from
A snoop hit response with the error bit set, where data from another snooped master is returned instead of this one.	CR	Slave interface receiving the CR response.
A snoop miss response with the error bit set.	CR	Slave interface receiving the CR response.
Write access generated by the CCI-500.	B	Master interface receiving the B response.
A snoop response with the error bit set where the snoop was generated from a WriteLineUnique or WriteUnique transaction.	CR	Master interface receiving the CR response.
A snoop response with the error bit set where the snoop was generated from a back-invalidation.	CR	Slave interface receiving the CR response.

The CCI-500 generates a precise error response for a security violation on a CCI-500 register access. See *2.4.6 Security* on page 2-37.

2.4.8 Cache maintenance operations

The CCI-500 supports snooping of cache-maintenance operations based on the Snoop Control Register.

You can use snooping and cache maintenance to manage Level 1 and Level 2 caches within the same domain as the CCI-500. The CCI-500 does not support the propagation of cache maintenance operations downstream of its master interfaces.

2.4.9 Barriers

The CCI-500 does not support barrier transactions. You must ensure that barriers are terminated upstream of the CCI. For example, set **SYSBARDISABLE** HIGH in Cortex-A processors.

2.4.10 Exclusive accesses

The CCI-500 supports the propagation of exclusive accesses to shareable and non-shareable locations. It does not contain master or slave exclusive access monitors, but does have *Point of Serialization* (PoS) exclusive monitors to avoid livelock.

Note

- See the *ARM® AMBA® AXI and ACE Protocol Specification* for more information on shareable and Non-shareable locations and PoS exclusive monitors.
- The *ARM® AMBA® AXI and ACE Protocol Specification* permits shareable exclusive accesses on ACE interfaces only.

2.4.11 DVM messages

All slave interfaces on the CCI-500 support DVM messages. For ACE-Lite interfaces, this is through the addition of AC and CR channels. Each slave interface has a hardware enable input and programmable enable bit to determine whether it supports the issuing of AC requests for DVM messages.

The *3.3.8 Snoop Control Registers* on page 3-59 and *3.3.1 Control Override Register* on page 3-50 control DVM message requests.

Note

A master that issues DVM messages must also be able to receive DVM messages. The slave interface through which the master connects must have DVM messages enabled.

2.4.12 Quality of Service

The CCI-500 provides a set of QoS regulation and control mechanisms.

The following mechanisms are supported:

- *QoS value as a priority indicator* on page 2-40. This is the reservation of resource based on a QoS threshold.
- *Regulation based on outstanding transactions* on page 2-40.

QoS value as a priority indicator

The CCI-500 uses the QoS value as a priority indicator for arbitration of requests. The QoS value can be from an input to a slave interface, or it can be overwritten by a programmed value.

The CCI-500 uses the QoS value when selecting the request to admit into the main transaction queue. Requests with the highest QoS have the highest priority unless an anti-starvation mechanism is activated. The CCI-500 uses a *Least Recently Granted* (LRG) scheme when two or more transactions share the highest priority. The arbiter has starvation avoidance mechanisms to prevent high bandwidth requests from stalling lower priority requests indefinitely.

The CCI-500 propagates QoS values. This determines the service rate when downstream interconnect and slave devices are sensitive to the QoS value. The NIC-400 Network Interconnect is sensitive to the QoS value.

——— **Note** ———

Ensure that you balance the relative priorities of all slave interfaces. For example, setting each one to the highest QoS value reduces the arbitration to LRG, and there is no advantage in using the QoS value.

———————————

You can override the **ARQOS** and **AWQOS** input signals from each slave interface by using a programmable register. The value from this register is only applied if the relevant static input signal, **QOSOVERRIDE[6:0]**, is HIGH. CCI-500-generated transactions use the QoS value of the trigger transaction or the override value if the **QOSOVERRIDE** signal is set.

——— **Note** ———

The **QOSOVERRIDE** signal only applies to transactions for which the **ARQOS** or **AWQOS** signals are set to a value of zero. Therefore, each interface can have a mixture of overridden traffic and other traffic, with an unaffected non-zero QoS value.

———————————

Regulation based on outstanding transactions

Each slave interface has a programmable mechanism for limiting the number of outstanding read and write transactions.

An *Outstanding Transaction* (OT) is a read request that has not yet received its last beat of read data, or a write request that has not yet received a response. You can use this mechanism in conjunction with QoS value mechanisms or when the system is not sensitive to the QoS value.

There is a combined OT count for read and write transactions, and this includes all possible request types. Two-part DVM messages count as two outstanding transactions, and transactions that the CCI-500 splits into 64-byte granules count as multiple transactions.

When programming the OT register, the hardware implementation sets the maximum value. This is the value of the register from reset. The minimum value for the OT register is `SIx_W_MIN` + 2, and this is the number of tracker slots reserved for requests from each slave interface to prevent deadlock. If you write a value outside these limits, then the limited value is set and read back.

The OT limit sets a maximum bandwidth for the attached master, based on the average response latency expected from downstream. For ACE masters, from the response to the **RACK** or **WACK** acknowledgement must be included in the response latency. This is approximately:

- OT limit = maximum bandwidth * average latency / bytes per request

For example, if the average latency between arrival at the main CCI-500 tracking structures and downstream response is 128ns, the maximum required bandwidth is 8GB/s, and requests are 64 bytes in length, then the necessary OT limit for an ACE-Lite master assuming a negligible hit rate is:

- max OT = 8 * 128 / 64 = 16

In this way, you can allocate memory bandwidth resource amongst various masters in the system.

Chapter 3
Programmers Model

This chapter describes the programmers model of the CoreLink CCI-500 Cache Coherent Interconnect

It contains the following sections:

- *3.1 About this programmers model* on page 3-43.
- *3.2 Register summary* on page 3-44.
- *3.3 Register descriptions* on page 3-50.
- *3.4 Address map* on page 3-70.

3.1 About this programmers model

This section provides general information about the CCI-500 register properties.

The following information applies to the CCI-500 registers:

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

RW	Read and write.
RO	Read only.
WO	Write only.
RAZ	Read as zero.
WI	Write ignored.

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

The CCI-500 registers are accessed using the APB slave interface and cannot be accessed directly through the ACE or ACE-Lite slave interfaces.

The programmers model contains regions for control, slave interface, and performance counter registers. Accesses to unmapped or reserved registers are WI/RAZ. Non-secure accesses to Secure registers are WI/RAZ.

The programmers model is not affected by the number of interfaces on a specific CCI-500 configuration. Writing to registers pertaining to interfaces that are not present on your specific implementation has no effect.

3.2 Register summary

The register summary lists all CCI-500 registers and some key characteristics.

The following table shows the registers in offset order. The base address of the CCI-500 is not fixed, and can be different for any particular system implementation. Consult your SoC implementation documentation for more information. The offset of each register from the base address is fixed.

Table 3-1 Register summary

Offset	Name	Type	Reset	Width	Description
0x00000	ctrl_ovr	RW	0x00000000	32	*3.3.1 Control Override Register* on page 3-50
0x00008	secr_acc	RW	0x00000000	32	*3.3.2 Secure Access Register* on page 3-51
0x0000C	status	RO	0x00000000 **Note** Assuming requested power state is OFF at reset.	32	*3.3.3 Status Register* on page 3-52
0x00010	impr_err	RW	0x00000000	32	*3.3.4 Imprecise Error Register* on page 3-54
0x00100	pmu_ctrl	-	0x00004000	32	*3.3.5 Performance Monitor Control Register (PMCR)* on page 3-57
0x00104	debug_ctrl	RW	0x00000000	32	*3.3.6 Interface Monitor Control Register* on page 3-58
0x00FD0	peripheral_id4	RO	0x00000084	32	*3.3.7 Component and Peripheral ID Registers* on page 3-58
0x00FD4	peripheral_id5	RO	0x00000000	32	*3.3.7 Component and Peripheral ID Registers* on page 3-58
0x00FD8	peripheral_id6	RO	0x00000000	32	*3.3.7 Component and Peripheral ID Registers* on page 3-58
0x00FDC	peripheral_id7	RO	0x00000000	32	*3.3.7 Component and Peripheral ID Registers* on page 3-58
0x00FE0	peripheral_id0	RO	0x00000022	32	*3.3.7 Component and Peripheral ID Registers* on page 3-58
0x00FE4	peripheral_id1	RO	0x000000B4	32	*3.3.7 Component and Peripheral ID Registers* on page 3-58
0x00FE8	peripheral_id2	RO	0x0000000B for r0p0 0x0000001B for r0p1	32	*3.3.7 Component and Peripheral ID Registers* on page 3-58
0x00FEC	peripheral_id3	RO	0x00000000	32	*3.3.7 Component and Peripheral ID Registers* on page 3-58
0x00FF0	component_id0	RO	0x0000000D	32	*3.3.7 Component and Peripheral ID Registers* on page 3-58
0x00FF4	component_id1	RO	0x000000F0	32	*3.3.7 Component and Peripheral ID Registers* on page 3-58

Table 3-1 Register summary (continued)

Offset	Name	Type	Reset	Width	Description
0x00FF8	component_id2	RO	0x00000005	32	*3.3.7 Component and Peripheral ID Registers* on page 3-58
0x00FFC	component_id3	RO	0x000000B1	32	*3.3.7 Component and Peripheral ID Registers* on page 3-58
Slave interface 0 registers					
0x01000	snoop_ctrl	-	[31:30] ACCHANNELENS0 [29:0] 0x00000000	32	*3.3.8 Snoop Control Registers* on page 3-59.
0x01004	share_ovr	RW	0x00000000	32	*3.3.9 Shareable Override Register* on page 3-61.
0x01100	arqos_ovr	RW	0x00000000	32	*3.3.10 Read Channel QoS Value Override Register* on page 3-62.
0x01104	awqos_ovr	RW	0x00000000	32	*3.3.11 Write Channel QoS Value Override Register* on page 3-63.
0x01110	qos_max_ot	RW	Maximum number of OT supported by this implementation, based on SI0_RW_MAX parameter	32	*3.3.12 Maximum Outstanding Transactions Registers* on page 3-63.
Slave interface 1 registers					
0x02000	snoop_ctrl	-	[31:30] ACCHANNELENS1 [29:0] 0x00000000	32	*3.3.8 Snoop Control Registers* on page 3-59.
0x02004	share_ovr	RW	0x00000000	32	*3.3.9 Shareable Override Register* on page 3-61.
0x02100	arqos_ovr	RW	0x00000000	32	*3.3.10 Read Channel QoS Value Override Register* on page 3-62.
0x02104	awqos_ovr	RW	0x00000000	32	*3.3.11 Write Channel QoS Value Override Register* on page 3-63.
0x02110	qos_max_ot	RW	Maximum number of OT supported by this implementation, based on SI1_RW_MAX parameter	32	*3.3.12 Maximum Outstanding Transactions Registers* on page 3-63.
Slave interface 2 registers					
0x03000	snoop_ctrl	-	[31:30] ACCHANNELENS2 [29:0] 0x00000000	32	*3.3.8 Snoop Control Registers* on page 3-59.
0x03004	share_ovr	RW	0x00000000	32	*3.3.9 Shareable Override Register* on page 3-61.
0x03100	arqos_ovr	RW	0x00000000	32	*3.3.10 Read Channel QoS Value Override Register* on page 3-62.
0x03104	awqos_ovr	RW	0x00000000	32	*3.3.11 Write Channel QoS Value Override Register* on page 3-63.

Table 3-1 Register summary (continued)

Offset	Name	Type	Reset	Width	Description
0x03110	qos_max_ot	RW	Maximum number of OT supported by this implementation, based on SI2_RW_MAX parameter	32	*3.3.12 Maximum Outstanding Transactions Registers* on page 3-63.
Slave interface 3 registers					
0x04000	snoop_ctrl	-	[31:30] ACCHANNELENS3 [29:0] 0x00000000	32	*3.3.8 Snoop Control Registers* on page 3-59.
0x04004	share_ovr	RW	0x00000000	32	*3.3.9 Shareable Override Register* on page 3-61.
0x04100	arqos_ovr	RW	0x00000000	32	*3.3.10 Read Channel QoS Value Override Register* on page 3-62
0x04104	awqos_ovr	RW	0x00000000	32	*3.3.11 Write Channel QoS Value Override Register* on page 3-63.
0x04110	qos_max_ot	RW	Maximum number of OT supported by this implementation, based on SI3_RW_MAX parameter	32	*3.3.12 Maximum Outstanding Transactions Registers* on page 3-63.
Slave interface 4 registers					
0x05000	snoop_ctrl	-	[31:30] ACCHANNELENS4 [29:0] 0x00000000	32	*3.3.8 Snoop Control Registers* on page 3-59.
0x05004	share_ovr	RW	0x00000000	32	*3.3.9 Shareable Override Register* on page 3-61.
0x05100	arqos_ovr	RW	0x00000000	32	*3.3.10 Read Channel QoS Value Override Register* on page 3-62.
0x05104	awqos_ovr	RW	0x00000000	32	*3.3.11 Write Channel QoS Value Override Register* on page 3-63.
0x05110	qos_max_ot	RW	Maximum number of OT supported by this implementation, based on SI4_RW_MAX parameter	32	*3.3.12 Maximum Outstanding Transactions Registers* on page 3-63.
Slave interface 5 registers					
0x06000	snoop_ctrl	-	[31:30] ACCHANNELENS5 [29:0] 0x00000000	32	*3.3.8 Snoop Control Registers* on page 3-59.
0x06004	share_ovr	RW	0x00000000	32	*3.3.9 Shareable Override Register* on page 3-61.
0x06100	arqos_ovr	RW	0x00000000	32	*3.3.10 Read Channel QoS Value Override Register* on page 3-62.
0x06104	awqos_ovr	RW	0x0000000	32	*3.3.11 Write Channel QoS Value Override Register* on page 3-63.
0x06110	qos_max_ot	RW	Maximum number of OT supported by this implementation, based on SI5_RW_MAX parameter	32	*3.3.12 Maximum Outstanding Transactions Registers* on page 3-63. .

Table 3-1 Register summary (continued)

Offset	Name	Type	Reset	Width	Description
Slave interface 6 registers					
0x07000	snoop_ctrl	-	[31:30] ACCHANNELENS6 [29:0] 0x00000000	32	*3.3.8 Snoop Control Registers* on page 3-59.
0x07004	share_ovr	RW	0x00000000	32	*3.3.9 Shareable Override Register* on page 3-61.
0x07100	arqos_ovr	RW	0x00000000	32	*3.3.10 Read Channel QoS Value Override Register* on page 3-62.
0x07104	awqos_ovr	RW	0x00000000	32	*3.3.11 Write Channel QoS Value Override Register* on page 3-63.
0x07110	qos_max_ot	RW	Maximum number of OT supported by this implementation, based on SI6_RW_MAX parameter	32	*3.3.12 Maximum Outstanding Transactions Registers* on page 3-63.
Performance counter 0 registers					
0x10000	evnt_sel	RW	0x00000000	32	*3.3.13 Event Select Registers* on page 3-64
0x10004	ecnt_data	RW	0x00000000	32	*3.3.14 Event Count Registers* on page 3-65
0x10008	ecnt_ctrl	RW	0x00000000	32	*3.3.15 Count Control Registers* on page 3-65
0x1000C	ecnt_clr_ovfl	RW	0x00000000	32	*3.3.16 Overflow Flag Status Registers* on page 3-66
Performance counter 1 registers					
0x20000	evnt_sel	RW	0x00000000	32	*3.3.13 Event Select Registers* on page 3-64
0x20004	ecnt_data	RW	0x00000000	32	*3.3.14 Event Count Registers* on page 3-65
0x20008	ecnt_ctrl	RW	0x00000000	32	*3.3.15 Count Control Registers* on page 3-65
0x2000C	ecnt_clr_ovfl	RW	0x00000000	32	*3.3.16 Overflow Flag Status Registers* on page 3-66
Performance counter 2 registers					
0x30000	evnt_sel	RW	0x00000000	32	*3.3.13 Event Select Registers* on page 3-64
0x30004	ecnt_data	RW	0x00000000	32	*3.3.14 Event Count Registers* on page 3-65
0x30008	ecnt_ctrl	RW	0x00000000	32	*3.3.15 Count Control Registers* on page 3-65
0x3000C	ecnt_clr_ovfl	RW	0x00000000	32	*3.3.16 Overflow Flag Status Registers* on page 3-66
Performance counter 3 registers					
0x40000	evnt_sel	RW	0x00000000	32	*3.3.13 Event Select Registers* on page 3-64
0x40004	ecnt_data	RW	0x00000000	32	*3.3.14 Event Count Registers* on page 3-65
0x40008	ecnt_ctrl	RW	0x00000000	32	*3.3.15 Count Control Registers* on page 3-65

Table 3-1 Register summary (continued)

Offset	Name	Type	Reset	Width	Description
0x4000C	ecnt_clr_ovfl	RW	0x00000000	32	*3.3.16 Overflow Flag Status Registers* on page 3-66
Performance counter 4 registers					
0x50000	evnt_sel	RW	0x00000000	32	*3.3.13 Event Select Registers* on page 3-64
0x50004	ecnt_data	RW	0x00000000	32	*3.3.14 Event Count Registers* on page 3-65
0x50008	ecnt_ctrl	RW	0x00000000	32	*3.3.15 Count Control Registers* on page 3-65
0x5000C	ecnt_clr_ovfl	RW	0x00000000	32	*3.3.16 Overflow Flag Status Registers* on page 3-66
Performance counter 5 registers					
0x60000	evnt_sel	RW	0x00000000	32	*3.3.13 Event Select Registers* on page 3-64
0x60004	ecnt_data	RW	0x00000000	32	*3.3.14 Event Count Registers* on page 3-65
0x60008	ecnt_ctrl	RW	0x00000000	32	*3.3.15 Count Control Registers* on page 3-65
0x6000C	ecnt_clr_ovfl	RW	0x00000000	32	*3.3.16 Overflow Flag Status Registers* on page 3-66
Performance counter 6 registers					
0x70000	evnt_sel	RW	0x00000000	32	*3.3.13 Event Select Registers* on page 3-64
0x70004	ecnt_data	RW	0x00000000	32	*3.3.14 Event Count Registers* on page 3-65
0x70008	ecnt_ctrl	RW	0x00000000	32	*3.3.15 Count Control Registers* on page 3-65
0x7000C	ecnt_clr_ovfl	RW	0x00000000	32	*3.3.16 Overflow Flag Status Registers* on page 3-66
Performance counter 7 registers					
0x80000	evnt_sel	RW	0x00000000	32	*3.3.13 Event Select Registers* on page 3-64
0x80004	ecnt_data	RW	0x00000000	32	*3.3.14 Event Count Registers* on page 3-65
0x80008	ecnt_ctrl	RW	0x00000000	32	*3.3.15 Count Control Registers* on page 3-65
0x8000C	ecnt_clr_ovfl	RW	0x00000000	32	*3.3.16 Overflow Flag Status Registers* on page 3-66
Slave Interface Monitor Registers					

Table 3-1 Register summary (continued)

Offset	Name	Type	Reset	Width	Description
0x90000	slave_debug, slave interface 0	R0	0x00000000	32	*3.3.17 Slave Interface Monitor Registers* on page 3-66
0x90004	slave_debug, slave interface 1	RO	0x00000000	32	
0x90008	slave_debug, slave interface 2	RO	0x00000000	32	
0x9000C	slave_debug, slave interface 3	RO	0x00000000	32	
0x90010	slave_debug, slave interface 4	RO	0x00000000	32	
0x90014	slave_debug, slave interface 5	RO	0x00000000	32	
0x90018	slave_debug, slave interface 6	RO	0x00000000	32	
Master Interface Monitor Registers					
0x90100	master_debug, master interface 0	RO	0x00000000	32	*3.3.18 Master Interface Monitor Registers* on page 3-68
0x90104	master_debug, master interface 1	RO	0x00000000	32	
0x90108	master_debug, master interface 2	RO	0x00000000	32	
0x9010C	master_debug, master interface 3	RO	0x00000000	32	
0x90110	master_debug, master interface 4	RO	0x00000000	32	
0x90114	master_debug, master interface 5	RO	0x00000000	32	

3.3 Register descriptions

Each register description provides information about the register, such as usage constraints, configurations, attributes, and bit assignments.

This section contains the following subsections:

- *3.3.1 Control Override Register* on page 3-50.
- *3.3.2 Secure Access Register* on page 3-51.
- *3.3.3 Status Register* on page 3-52.
- *3.3.4 Imprecise Error Register* on page 3-54.
- *3.3.5 Performance Monitor Control Register (PMCR)* on page 3-57.
- *3.3.6 Interface Monitor Control Register* on page 3-58.
- *3.3.7 Component and Peripheral ID Registers* on page 3-58.
- *3.3.8 Snoop Control Registers* on page 3-59.
- *3.3.9 Shareable Override Register* on page 3-61.
- *3.3.10 Read Channel QoS Value Override Register* on page 3-62.
- *3.3.11 Write Channel QoS Value Override Register* on page 3-63.
- *3.3.12 Maximum Outstanding Transactions Registers* on page 3-63.
- *3.3.13 Event Select Registers* on page 3-64.
- *3.3.14 Event Count Registers* on page 3-65.
- *3.3.15 Count Control Registers* on page 3-65.
- *3.3.16 Overflow Flag Status Registers* on page 3-66.
- *3.3.17 Slave Interface Monitor Registers* on page 3-66.
- *3.3.18 Master Interface Monitor Registers* on page 3-68.

3.3.1 Control Override Register

This register provides a fail-safe override for some CCI-500 functions. Use this register to resolve problems that you cannot work around in another way.

Usage constraints

If you have to write to this register, you must do so before issuing any shareable transactions or DVM messages to the CCI-500. For example, you can do it very early in the boot sequence before installing any Secure OS.

You can access this register using Secure transactions only, irrespective of the programming of the *3.3.2 Secure Access Register* on page 3-51.

Configurations

Available in all CCI-500 configurations.

Attributes

See *Table 3-1 Register summary* on page 3-44.

The following figure shows the bit assignments.

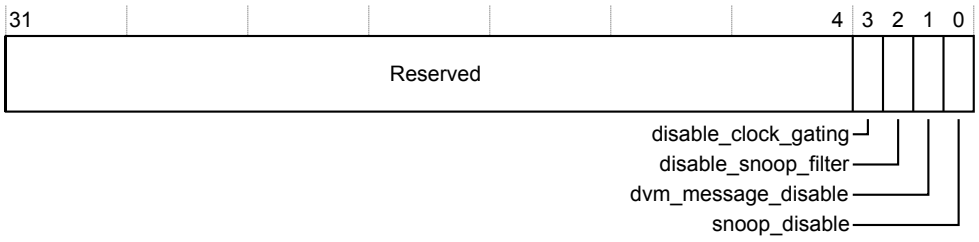

Figure 3-1 ctrl_ovr register bit assignments

The following table shows the bit assignments.

Table 3-2 ctrl_ovr register bit assignments

Bits	Name	Function
[31:4]	Reserved	-
[3]	disable_clock_gating	Disable regional clock gating: **0** Regional clock gating operates in the CCI-500. See *1.7 Test features* on page 1-18 and *2.3 Clocking and reset* on page 2-27. **1** Disable regional clock gating in the CCI-500.
[2]	disable_snoop_filter	Disable the snoop filter: **0** Snoop filter operation is defined by the power state input, **PSTATE**. **1** Disable snoop filter operation.
[1]	dvm_message_disable	DVM message disable: **0** Send DVM messages according to the Snoop Control Registers. See *3.3.8 Snoop Control Registers* on page 3-59. **1** Disable propagation of all DVM messages.
[0]	snoop_disable	Snoop disable: **0** Send snoop requests according to the Snoop Control Registers. See *3.3.8 Snoop Control Registers* on page 3-59. **1** Disable all snoops but not DVM messages.

3.3.2 Secure Access Register

This register controls whether only Non-secure transactions can read and program the CCI-500 registers.

Usage constraints
You can only write to this register only using Secure transactions.

Configurations
Available in all CCI-500 configurations.

Attributes
See *Table 3-1 Register summary* on page 3-44.

Warning

This register enables Non-secure access for all masters to the CCI-500 registers. This compromises the security of your system.

The following figure shows the bit assignments.

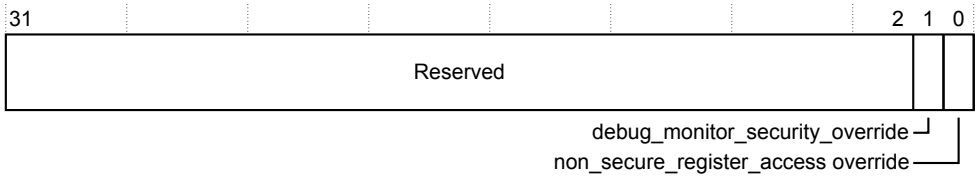

Figure 3-2 secr_acc register bit assignments

The following table shows the bit assignments.

Table 3-3 secr_acc register bit assignments

Bits	Name	Function
[31:2]	Reserved	-
[1]	debug_monitor_security_override	Debug monitor security override: **0** Enable Non-secure access to the PMU and Interface Monitor Registers. **1** Disable Non-secure access to the PMU and Interface Monitor Registers, unless overridden by bit[0].
[0]	non_secure_register_access_override	Non-secure register access override: **0** Disable Non-secure access to the CCI-500 registers. **1** Enable Non-secure access to the CCI-500 registers.

3.3.3 Status Register

This register permits snooping to be enabled and disabled safely by indicating when changes made to the enable_snoops or enable_dvms bits in the Snoop Control Registers have not taken effect for all transactions outstanding in the system.

When changing these bits, the CCI-500 goes through a transition period where a mixture of transactions with the old value and transactions with the new value are in flight. During this time, the change_pending bit stays set to 1. You must wait for the change_pending bit to change to 0 before removing or adding masters into the coherency domain. See also *2.4.3 Snoop connectivity and control* on page 2-29.

Note

You must wait for the completion of write to the Snoop Control Register before testing the change_pending bit.

This register indicates whether:

- There are any changes to the enables that have not yet been applied.
- A slave interface has been disabled for future snoop and DVM messages, but has outstanding AC requests.

Other bits in the Status Register indicate:

- Current power state.
- Requested power state.
- Power state change pending.
- Snoop filter initialization phase.

Usage constraints

There are no usage constraints.

Configurations

Available in all CCI-500 configurations.

Attributes

See *Table 3-1 Register summary* on page 3-44.

The following figure shows the bit assignments.

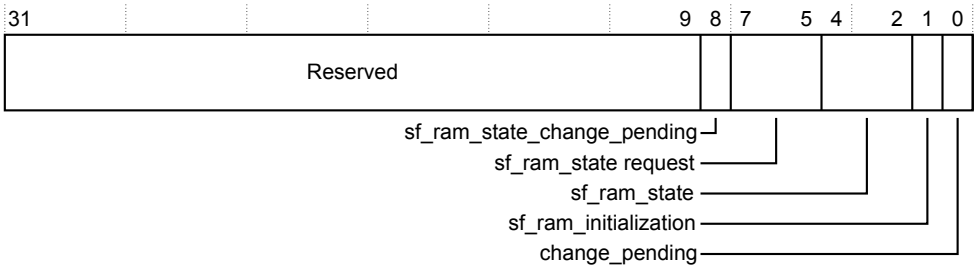

Figure 3-3 status register bit assignments

The following table shows the bit assignments.

Table 3-4 status register bit assignments

Bits	Name	Function
[31:9]	Reserved	-
[8]	sf_ram_state_change_pending	Snoop filter RAM power state change pending. This bit reads back the **PREQ** input. **0** No change pending, any previous requests have been accepted or denied. **1** State change is pending and might be accepted or denied.
[7:5]	sf_ram_state_request	This indicates the last requested power state of the snoop filter RAMs. The possible values of this field are the same as those of sf_ram_state.
[4:2]	sf_ram_state	The snoop filter RAM power states are: `0b000` Off. `0b001` Static snoop filter RAM retention. `0b010` Reserved. `0b011` Dynamic snoop filter RAM retention. `0b100` On. `0b101-0b111` Reserved. **Note** This register is readable only when the interconnect is in either the dynamic retention or the On state.

Table 3-4 status register bit assignments (continued)

Bits	Name	Function
[1]	sf_ram_initialization	Indicates when the snoop filter RAM is initialized. Shareable requests are not serviced during this period. **0** Snoop filter RAM initialization is complete. **1** Snoop filter RAM initialization is in progress. **Note** If you use the interconnect to access the CCI-500 registers when the trackers are full of shareable requests waiting for initialization completion, it might not be possible to read this register until initialization is complete.
[0]	change_pending	Indicates whether any changes to the *3.3.8 Snoop Control Registers* on page 3-59 or the *3.3.1 Control Override Register* on page 3-50 are pending in the CCI-500: **0** No changes are pending. **1** Changes are pending.

3.3.4 Imprecise Error Register

This register records the CCI-500 interfaces that have encountered an error that cannot be signaled precisely.

A register bit corresponding to a CCI-500 interface is set when one or more error responses are detected on that interface. Each bit is reset on a write of 1 to that bit.

Usage constraints
Accessible using only Secure accesses, unless you set the Secure Access Register. See *3.3.2 Secure Access Register* on page 3-51.

Configurations
Available in all CCI-500 configurations.

Attributes
See *Table 3-1 Register summary* on page 3-44.

Note

If any bits are set in this register, the **nERRIRQ** signal is asserted, active LOW.

The following figure shows the bit assignments.

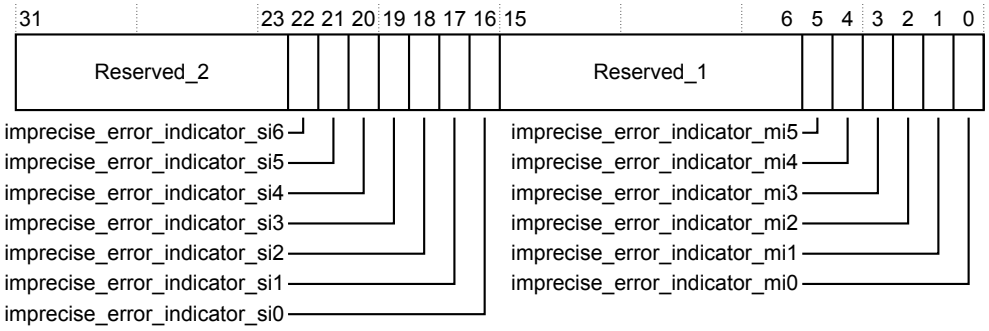

Figure 3-4 impr_err register bit assignments

The following table shows the bit assignments.

Table 3-5 impr_err register bit assignments

Bits	Name	Function
[31:23]	Reserved_2	-
[22]	imprecise_error_indicator_si6	Imprecise error indicator for slave interface 6: **0** No error from the time this bit was last reset. **1** An error response has been received, but not signaled precisely.
[21]	imprecise_error_indicator_si5	Imprecise error indicator for slave interface 5: **0** No error from the time this bit was last reset. **1** An error response has been received, but not signaled precisely.
[20]	imprecise_error_indicator_si4	Imprecise error indicator for slave interface 4: **0** No error from the time this bit was last reset. **1** An error response has been received, but not signaled precisely.
[19]	imprecise_error_indicator_si3	Imprecise error indicator for slave interface 3: **0** No error from the time this bit was last reset. **1** An error response has been received, but not signaled precisely.
[18]	imprecise_error_indicator_si2	Imprecise error indicator for slave interface 2: **0** No error from the time this bit was last reset. **1** An error response has been received, but not signaled precisely.
[17]	imprecise_error_indicator_si1	Imprecise error indicator for slave interface 1: **0** No error from the time this bit was last reset. **1** An error response has been received, but not signaled precisely.
[16]	imprecise_error_indicator_si0	Imprecise error indicator for slave interface 0: **0** No error from the time this bit was last reset. **1** An error response has been received, but not signaled precisely.
[15:6]	Reserved_1	-
[5]	imprecise_error_indicator_mi5	Imprecise error indicator for master interface 5: **0** No error from the time this bit was last reset. **1** An error response has been received, but not signaled precisely.
[4]	imprecise_error_indicator_mi4	Imprecise error indicator for master interface 4: **0** No error from the time this bit was last reset. **1** An error response has been received, but not signaled precisely.
[3]	imprecise_error_indicator_mi3	Imprecise error indicator for master interface 3: **0** No error from the time this bit was last reset. **1** An error response has been received, but not signaled precisely.

Table 3-5 impr_err register bit assignments (continued)

Bits	Name	Function
[2]	imprecise_error_indicator_mi2	Imprecise error indicator for master interface 2: **0** No error from the time this bit was last reset. **1** An error response has been received, but not signaled precisely.
[1]	imprecise_error_indicator_mi1	Imprecise error indicator for master interface 1: **0** No error from the time this bit was last reset. **1** An error response has been received, but not signaled precisely.
[0]	imprecise_error_indicator_mi0	Imprecise error indicator for master interface 0: **0** No error from the time this bit was last reset. **1** An error response has been received, but not signaled precisely.

3.3.5 Performance Monitor Control Register (PMCR)

This register controls the PMU.

Usage constraints

Accessible using both Secure and Non-secure accesses, unless you set bit[1] of the Secure Access Register to disable Non-secure accesses to this register. See *3.3.2 Secure Access Register* on page 3-51.

Configurations

Available in all CCI-500 configurations.

Attributes

See *Table 3-1 Register summary* on page 3-44.

The following figure shows the bit assignments.

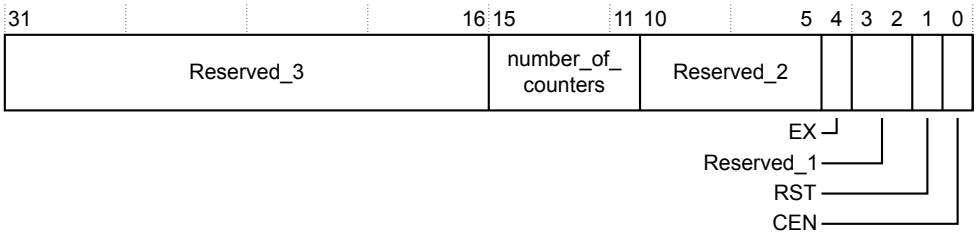

Figure 3-5 pmu_ctrl register bit assignments

The following table shows the bit assignments.

Table 3-6 pmu_ctrl register bit assignments

Bits	Name	Access	Function
[31:16]	Reserved_3	-	-
[15:11]	number_of_counters	R/WI	Specifies the number of counters implemented.
[10:5]	Reserved_2	-	-
[4]	EX	RW	Enables export of the events to the event bus, **EVNTBUS**, to permit an external monitoring block to trace events: **0** Do not export **EVNTBUS**. **1** Export **EVNTBUS**.
[3:2]	Reserved_1	-	
[1]	RST	RAZ/W	Performance counter reset: **0** No action. **1** Reset all performance counters to zero.
[0]	CEN	RW	Enable bit: **0** Disable all event counters. **1** Enable all event counters.

The following table shows the relationship between the debug enable inputs, **NIDEN** and **DBGEN**, and the PMCR register settings.

Note

In this table, X can be any value.

Table 3-7 Relationship between NIDEN and DBGEN, and PMCR register settings

NIDEN OR DBGEN	PMCR.CEN	PMCR.EX	Event counters enabled	Events exported
0	X	X	No	No
1	0	X	No	No
1	1	0	Yes	No
1	1	1	Yes	Yes

3.3.6 Interface Monitor Control Register

This register enables all interface monitor control.

Usage constraints

Accessible using both Secure and Non-secure accesses, unless you set bit[1] of the Secure Access Register to disable Non-secure accesses to this register. See *3.3.2 Secure Access Register* on page 3-51.

Configurations

Available in all CCI-500 configurations.

Attributes

See *Table 3-1 Register summary* on page 3-44.

The following figure shows the bit assignments.

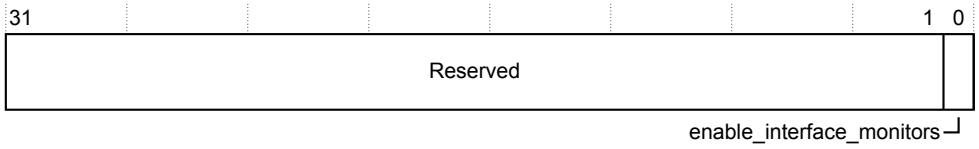

Figure 3-6 debug_ctrl register bit assignments

The following table shows the bit assignments.

Table 3-8 debug_ctrl register bit assignments

Bits	Name	Function
[31:1]	Reserved	-
[0]	enable_interface_monitors	Enable bit: **0** Interface Monitor counters and flags are set to 0. **1** Enable all Interface Monitors.

3.3.7 Component and Peripheral ID Registers

The component and peripheral identity registers are standard JEP106 registers. They provide key information about the CCI-500 hardware, including the product and associated revision number. They also identify ARM as the manufacturer.

These registers are all read-only. Each field is a single byte. This means you must read the most significant 24 bits as zero and only the least significant byte is valid. The least significant 8 bits of the four Component ID registers form a single 32-bit conceptual ID register. In a similar way, the defined fields of the eight Peripheral ID registers form a conceptual 64-bit ID register.

Table 3-9 Component and Peripheral ID registers bit assignments

Register	Offset	Bits	Value	Function
Peripheral ID4	0xFD0	[7:4]	0x8	4KB region count.
		[3:0]	0x4	JEP106 continuation code for ARM.
Peripheral ID5	0xFD4	[7:0]	0x00	Reserved.
Peripheral ID6	0xFD8	[7:0]	0x00	Reserved.
Peripheral ID7	0xFDC	[7:0]	0x00	Reserved.
Peripheral ID0	0xFE0	[7:0]	0x22	Part number[7:0].
Peripheral ID1	0xFE4	[7:4]	0xB	JEP106 ID code[3:0] for ARM.
		[3:0]	0x4	Part number[11:8].
Peripheral ID2	0xFE8	[7:4]	0x0 or 0x1	CCI-500 revision: • 0x0 for r0p0. • 0x1 for r0p1.
		[3]	0x1	IC uses a manufacturer's identity code allocated by JEDEC according to the JEP106 specification.
		[2:0]	0x3	JEP106 ID code[6:4] for ARM.
Peripheral ID3	0xFEC	[7:4]	0x0	ARM-approved ECO number. Use the **ECOREVNUM** inputs to modify this value.
		[3:0]	0x0	Customer modification number. You must not modify this number unless you have permission from ARM.
Component ID0	0xFF0	[7:0]	0x0D	These values identify the CCI-500 as an ARM component.
Component ID1	0xFF4	[7:0]	0xF0	
Component ID2	0xFF8	[7:0]	0x05	
Component ID3	0xFFC	[7:0]	0xB1	

ECO revision number

To track any *Engineering Change Order* (ECO) fixes in the CCI-500, you can change part of the peripheral ID register using the **ECOREVNUM** input pins. You must tie these signals LOW unless you have an ECO from ARM.

The **ECOREVNUM[3:0]** input corresponds to bits[7:4] of the Peripheral ID3 register, MSB to MSB. Driving an input bit HIGH inverts the associated Peripheral ID3 bit.

Note

ARM recommends that each of the signal drivers is distinct and readily identifiable to facilitate possible metal layer modification.

3.3.8 Snoop Control Registers

These registers control the issuing of snoop and DVM requests on slave interfaces.

You can read the register to determine if the interface supports snoops or DVM messages. Enabling snoop or DVM requests on an interface that does not support them has no effect.

Usage constraints

Accessible using only Secure accesses, unless you set the Secure Access Register to permit Non-secure accesses. See *3.3.2 Secure Access Register* on page 3-51.

Configurations

Available in all CCI-500 configurations.

A copy of this register exists for each slave interface.

Attributes

See *Table 3-1 Register summary* on page 3-44.

The following figure shows the bit assignments.

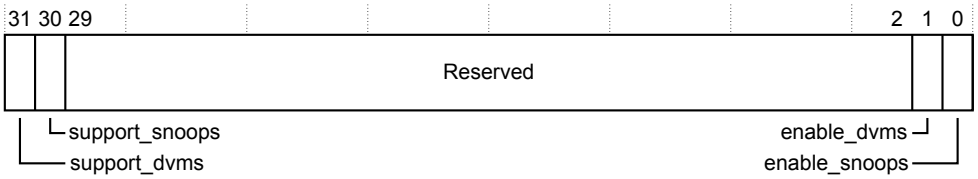

Figure 3-7 snoop_ctrl register bit assignments

The following table shows the bit assignments.

Table 3-10 snoop_ctrl register bit assignments

Bits	Name	Reset	Access	Function
[31]	support_dvms	**ACCHANNELENSx[0]** input	R/WI	Indicates whether the slave interface supports DVM messages. **0** The interface does not support DVM messages. **1** The interface supports DVM messages. This is overridden to 0 if you set the Control Override Register bit[1]. See *3.3.1 Control Override Register* on page 3-50.
[30]	support_snoops	**ACCHANNELENSx[1]** input for ACE interfaces. This bit is set to 0 for ACE-Lite interfaces.	R/WI	Indicates whether the slave interface supports snoop requests. **0** The interface does not support snoops. **1** The interface supports snoops. This is overridden to 0 if you set the Control Override Register bit[0]. See *3.3.1 Control Override Register* on page 3-50. **Note** This bit only affects the operation of ACE interfaces.
[29:2]	Reserved	-	-	-

Table 3-10 snoop_ctrl register bit assignments (continued)

Bits	Name	Reset	Access	Function
[1]	enable_dvms	0	RW	When the slave interface supports DVM messages, enables issuing of DVM message requests from this slave interface: **0** Disable DVM message requests. **1** Enable DVM message requests. This bit is RAZ/WI for interfaces that do not support DVM messages. **Note** This bit is writable only when bit[31] is set to 1.
[0]	enable_snoops	0	RW	When the slave interface supports snoops, enables issuing of snoop requests from this slave interface: **0** Disable snoop requests. **1** Enable snoop requests. This bit is RAZ/WI for interfaces that do not support snoops. **Note** • This bit only affects the operation of ACE interfaces. • This bit is writable only when bit[30] is set to 1.

3.3.9 Shareable Override Register

This register overrides the shareability characteristics of Normal transactions received on the relevant interface. Overriding of the shareability settings does not occur for FIXED-type bursts, Device transactions, or DVM message transactions.

Usage constraints

This register is for ACE-Lite slave interfaces only.

Accessible using only Secure accesses, unless you set the Secure Access Register to permit Non-secure accesses. See *3.3.2 Secure Access Register* on page 3-51.

Configurations

Available in all CCI-500 configurations.

A copy of this register exists for each slave interface.

Attributes

See *Table 3-1 Register summary* on page 3-44.

Note

Exclusive accesses must not be issued on an interface that is being overridden as shareable. If the CCI-500 is programmed to override transactions as shareable, exclusive accesses are overridden to normal accesses. An exclusive write then receives an OKAY response to indicate that the slave does not support exclusive accesses.

The following figure shows the bit assignments.

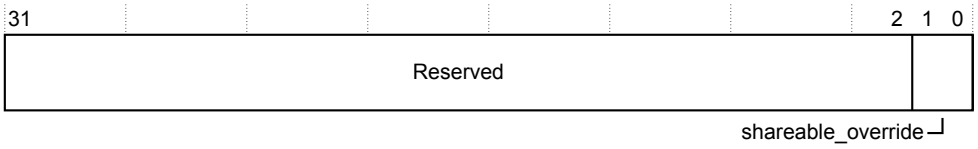

Figure 3-8 share_ovr register bit assignments

The following table shows the bit assignments.

Table 3-11 share_ovr register bit assignments

Bits	Name	Function
[31:2]	Reserved	-
[1:0]	shareable_override	Shareable override for slave interface: `0b00-0b01` Do not override **AxDOMAIN** inputs. `0b10` Override **AxDOMAIN** inputs to `0b00`, meaning that all transactions are treated as Non-shareable: • ReadOnce becomes ReadNoSnoop. • WriteUnique and WriteLineUnique become WriteNoSnoop. • CleanShared, CleanInvalid, and MakeInvalid transactions do not generate snoops. `0b11` Override **AxDOMAIN** inputs to `0b01`, meaning that all Normal transactions are treated as Shareable: • ReadNoSnoop becomes ReadOnce. • WriteNoSnoop becomes WriteUnique. • CleanShared, CleanInvalid, and MakeInvalid transactions generate snoops.

3.3.10 Read Channel QoS Value Override Register

This register stores the override value for the **ARQOS** signal when there is a separate register for each slave interface. This value is applied to transactions when the **QOSOVERRIDE** input signal bit is HIGH for the relevant slave interface and the **ARQOS** input is zero for that request.

Usage constraints

Accessible using only Secure accesses, unless you set the *3.3.2 Secure Access Register* on page 3-51 to permit Non-secure accesses.

Configurations

Available in all CCI-500 configurations.

A copy of this register exists for each slave interface.

Attributes

See *Table 3-1 Register summary* on page 3-44.

The following figure shows the bit assignments.

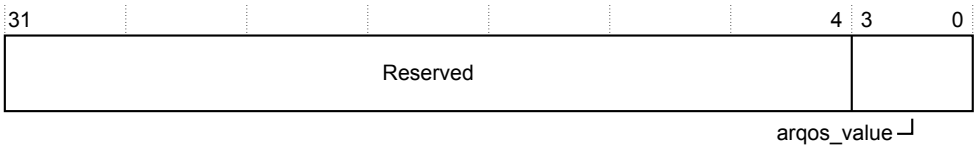

Figure 3-9 arqos_ovr register bit assignments

The following table shows the bit assignments.

Table 3-12 arqos_ovr register bit assignments

Bits	Name	Function
[31:4]	Reserved	-
[3:0]	arqos_value	**ARQOS** value override for the slave interface. **Note** This value is applied to transactions with an **ARQOS** value of zero, if the **QOSOVERRIDE** input is HIGH for this interface.

3.3.11 Write Channel QoS Value Override Register

This register stores the override value for the **AWQOS** signal. If the **QOSOVERRIDE** input bit is HIGH for the relevant slave interface, this override value is applied to requests that have an **AWQOS** value of zero.

Usage constraints

Accessible using only Secure accesses, unless you set the *3.3.2 Secure Access Register* on page 3-51 to permit Non-secure accesses.

Configurations

Available in all CCI-500 configurations.

A copy of this register exists for each slave interface.

Attributes

See *Table 3-1 Register summary* on page 3-44.

The following figure shows the bit assignments.

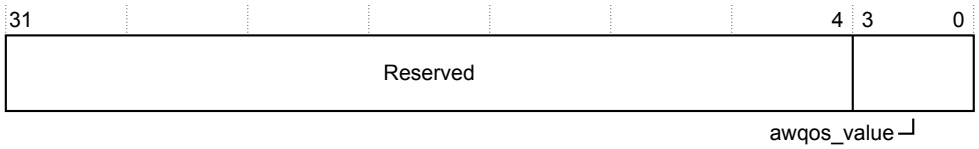

Figure 3-10 awqos_ovr register bit assignments

The following table shows the bit assignments.

Table 3-13 awqos_ovr register bit assignments

Bits	Name	Function
[31:4]	Reserved	-
[3:0]	awqos_value	**AWQOS** value override for the slave interface. **Note** This value is applied to transactions with an **AWQOS** value of zero, if the **QOSOVERRIDE** input is HIGH for this interface.

3.3.12 Maximum Outstanding Transactions Registers

These registers determine how many *Outstanding Transactions* (OTs) are permitted when the OT regulator is enabled for the relevant slave interface.

Usage constraints

If you set the maximum OT size greater than that configured in the RTL, then the value of `SIx_RW_MAX` is written into the register. The minimum value of `SIx_RW_MAX` is 4. Writing values lower than this writes a value of 4 into the register.

Accessible using only Secure accesses, unless you set the *3.3.2 Secure Access Register* on page 3-51 to permit Non-secure accesses.

Configurations

Available in all CCI-500 configurations.

A copy of this register exists for each slave interface.

Attributes

See *Table 3-1 Register summary* on page 3-44.

The following figure shows the bit assignments.

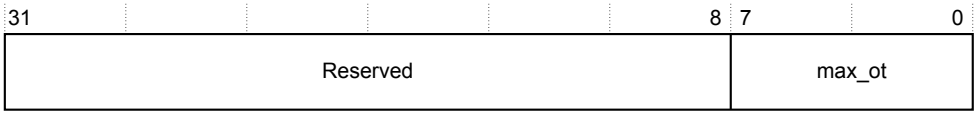

Figure 3-11 qos_max_ot register bit assignments

The following table shows the bit assignments.

Table 3-14 qos_max_ot register bit assignments

Bits	Name	Reset	Function
[31:8]	Reserved	-	-
[7:0]	max_ot	`SIx_RW_MAX`	The maximum number of OTs for the interface. This is a combined issuing limit. It represents the maximum number of transactions that the upstream master can issue when the AR and AW channels are considered as one issuing source. **Note** Additional transactions can be issued into the CCI-500 at the boundary of the device. This is because of the presence of configurable registering between the boundary and the main trackers.

3.3.13 Event Select Registers

These registers determine the event that a particular counter tracks.

Usage constraints

There are no usage constraints.

Configurations

Available in all CCI-500 configurations.

One register exists per counter.

Attributes

See *Table 3-1 Register summary* on page 3-44.

Note

You can use event counters in different ways, for example:

- To measure traffic across all interfaces by using a counter for each interface.
- To analyze a particular interface by using all the counters to measure a different aspect of the interface.

The following figure shows the bit assignments.

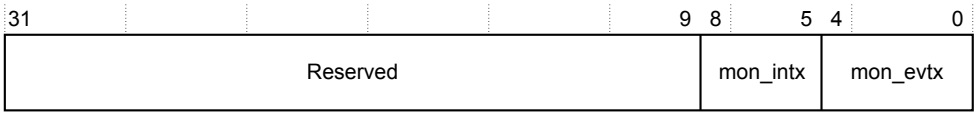

Figure 3-12 evnt_sel register bit assignments

The following table shows the bit assignments.

Table 3-15 evnt_sel register bit assignments

Bits	Name	Function
[31:9]	Reserved	-
[8:5]	mon_intx	Event code that defines the interface to monitor. See *PMU event list* on page 2-31.
[4:0]	mon_evtx	Event code that defines the event to monitor. See *PMU event list* on page 2-31.

3.3.14 Event Count Registers

These are 32-bit RW registers. There is one for each of the eight corresponding event counters.

You can reset all event counter values to zero by writing a 1 to the **PMCR** bit[1].

3.3.15 Count Control Registers

These registers enable or disable the event counters.

Usage constraints
There are no usage constraints.

Configurations
Available in all CCI-500 configurations.
One register exists per counter.

Attributes
See *Table 3-1 Register summary* on page 3-44.

The following figure shows the bit assignments.

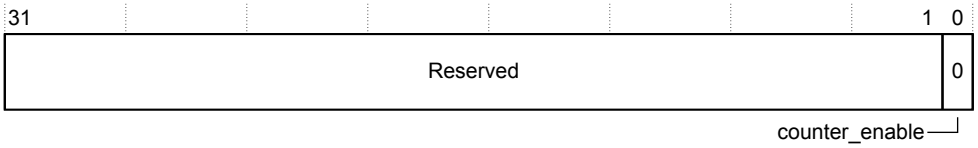

Figure 3-13 ecnt_ctrl register bit assignments

The following table shows the bit assignments.

Table 3-16 ecnt_ctrl register bit assignments

Bits	Name	Function
[31:1]	Reserved	-
[0]	counter_enable	Counter enable: **0** Counter disabled. **1** Counter enabled.

3.3.16 Overflow Flag Status Registers

These registers contain the state of the overflow flags for the event counters.

Usage constraints
There are no usage constraints.
Configurations
Available in all CCI-500 configurations.
One register exists for each event counter.
Attributes
See *Table 3-1 Register summary* on page 3-44.

The following figure shows the bit assignments.

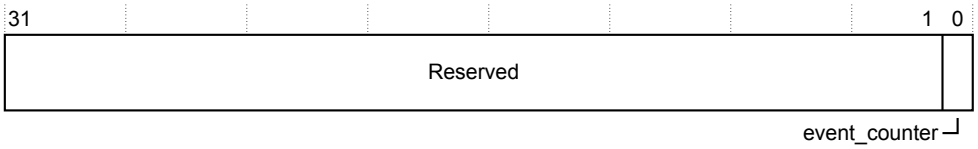

Figure 3-14 ecnt_clr_ovfl register bit assignments

The following table shows the bit assignments.

Table 3-17 ecnt_clr_ovfl register bit assignments

Bits	Name	Function
[31:1]	Reserved	-
[0]	event_counter	Event counter overflow flag: **0** The counter has not overflowed. **1** The counter has overflowed.

When writing to this register, any overflow flag written with a value of 0 is ignored, that is, no change. An overflow flag written with a value of 1 clears the counter overflow flag. The negated counter overflow bits are exported from the CCI-500 on the **nEVNTCNTOVERFLOW[7:0]** signal. You can use this to trigger interrupts. The MSB corresponds to the cycle count overflow.

3.3.17 Slave Interface Monitor Registers

These 32-bit RO registers monitor each slave interface.

Usage constraints
There are no usage constraints.
Configurations
Available in all CCI-500 configurations.
A copy of this register exists for each slave interface.
Attributes
See *Table 3-1 Register summary* on page 3-44.

The following figure shows the bit assignments.

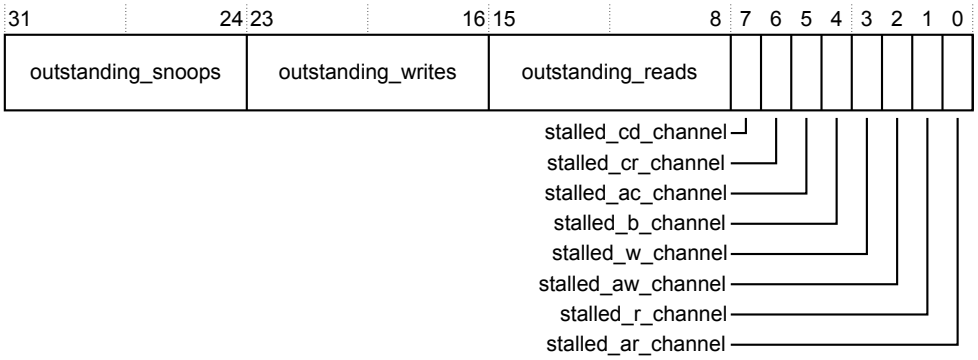

Figure 3-15 slave_debug register bit assignments

The following table shows the bit assignments.

Table 3-18 slave_debug register bit assignments

Bits	Name	Function
[31:24]	outstanding_snoops	Number of outstanding snoop requests or DVM messages. From request handshake to response or snoop data for a hit.
[23:16]	outstanding_writes	Number of outstanding write transactions. From request handshake to response for ACE-Lite interfaces, or **WACK** for ACE interfaces.
[15:8]	outstanding_reads	Number of outstanding read transactions. From request handshake to response for ACE-Lite interfaces, or **RACK** for ACE interfaces.
[7]	stalled_cd_channel	When this bit is set to 1, a transfer is stalled on the CD channel, where both: • **CDVALID** is HIGH. • **CDREADY** is LOW. This bit applies to ACE slaves only.
[6]	stalled_cr_channel	When this bit is set to 1, a transfer is stalled on the CR channel, where both: • **CRVALID** is HIGH. • **CRREADY** is LOW.
[5]	stalled_ac_channel	When this bit is set to 1, a transfer is stalled on the AC channel, where both: • **ACVALID** is HIGH. • **ACREADY** is LOW.
[4]	stalled_b_channel	When this bit is set to 1, a transfer is stalled on the B channel, where both: • **BVALID** is HIGH. • **BREADY** is LOW.
[3]	stalled_w_channel	When this bit is set to 1, a transfer is stalled on the W channel, where both: • **WVALID** is HIGH. • **WREADY** is LOW.

Table 3-18 slave_debug register bit assignments (continued)

Bits	Name	Function
[2]	stalled_aw_channel	When this bit is set to 1, a transfer is stalled on the AW channel, where both: • **AWVALID** is HIGH. • **AWREADY** is LOW.
[1]	stalled_r_channel	When this bit is set to 1, a transfer is stalled on the R channel, where both: • **RVALID** is HIGH. • **RREADY** is LOW.
[0]	stalled_ar_channel	When this bit is set to 1, a transfer is stalled on the AR channel, where both: • **ARVALID** is HIGH. • **ARREADY** is LOW.

3.3.18 Master Interface Monitor Registers

These 32-bit RO registers monitor each master interface.

Usage constraints

There are no usage constraints.

Configurations

Available in all CCI-500 configurations.

A copy of this register exists for each master interface.

Attributes

See *Table 3-1 Register summary* on page 3-44.

The following figure shows the bit assignments.

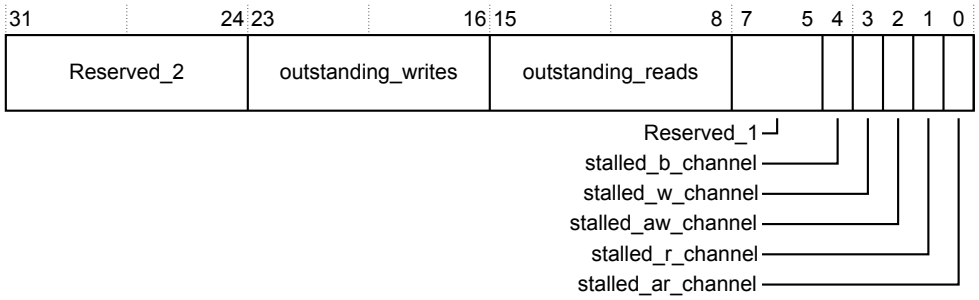

Figure 3-16 master_debug register bit assignments

The following table shows the bit assignments.

Table 3-19 master_debug register bit assignments

Bits	Name	Function
[31:24]	Reserved_2	-
[23:16]	outstanding_writes	Number of outstanding write transactions. From request handshake to response.
[15:8]	outstanding_reads	Number of outstanding read transactions. From request handshake to response.
[7:5]	Reserved_1	-

Table 3-19 master_debug register bit assignments (continued)

Bits	Name	Function
[4]	stalled_b_channel	When this bit is set to 1, a transfer is stalled on the B channel, where: • **BVALID** is HIGH. • **BREADY** is LOW.
[3]	stalled_w_channel	When this bit is set to 1, a transfer is stalled on the W channel, where both: • **WVALID** is HIGH. • **WREADY** is LOW.
[2]	stalled_aw_channel	When this bit is set to 1, a transfer is stalled on the AW channel, where both: • **AWVALID** is HIGH. • **AWREADY** is LOW.
[1]	stalled_r_channel	When this bit is set to 1, a transfer is stalled on the R channel, where both: • **RVALID** is HIGH. • **RREADY** is LOW.
[0]	stalled_ar_channel	When this bit is set to 1, a transfer is stalled on the AR channel. **ARVALID** is HIGH. **ARREADY** is LOW.

3.4 Address map

The CCI-500 uses an address map to route requests from slave interfaces to master interfaces. It is supplied with an example address decoder, that defines a global address map. You can rewrite the address decoders to define any address map at implementation time and also reconfigure the decoders at reset-time.

There is an address decoder per slave interface for each of read and write requests. You can, for example, use the same address map for each, or have a different decoder per slave interface. You must ensure that accesses to the same address are routed to the same slaves downstream of the CCI-500.

——— **Note** ———

The following text and diagram describes the operation of the address decoder supplied with CCI-500. However, the implementer is permitted to modify the address decoder. See your platform documentation to determine the address map for a particular implementation.

———————————

The example decoder supplied with the CCI-500 defines address regions, as specified in the document *Principles of ARM® Memory Maps*. The CCI-500 address width is configurable, so the address map extends up to 44 bits. If your implementation uses a smaller address width, then some regions are not addressable. *Figure 3-17 Example decoder address regions* on page 3-71 shows the region sizes and offsets, with associated **ADDRMAP** inputs and address width limits.

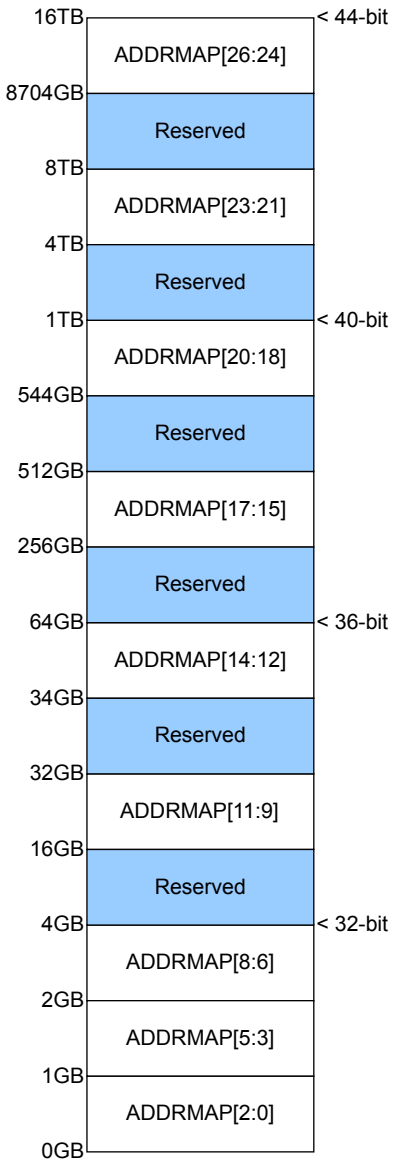

Figure 3-17 Example decoder address regions

In the example decoder, accesses to reserved regions generate a DECERR response.

For each non-reserved region, accesses are mapped to one of the master interfaces, or can be striped across a number of master interfaces. The mapping is determined using configuration input signals, **ADDRMAP**, that are sampled at reset. In the example address map, there are three **ADDRMAP** bits per region, with the encoding defined as:

Table 3-20 Decoder mapping

ADDRMAP[2:0]	Decode
`0b000`	Master interface 0
`0b001`	Master interface 1
`0b010`	Master interface 2
`0b011`	Master interface 3
`0b100`	Master interface 4

Table 3-20 Decoder mapping (continued)

ADDRMAP[2:0]	Decode
0b101	Master interface 5
0b110	Reserved
0b111	The behavior depends on the number of memory interfaces that are configured: • With one memory port, all accesses in the region are to that port. • With two memory ports, striping occurs across both ports. • With three memory ports, striping occurs across the two highest numbered ports. • With four memory ports, striping occurs across all four ports. **Note** When using the supplied address decoder, memory ports are the highest numbered master interfaces.

If the **ADDRMAP** input maps a region to a master interface that is not present in the specific configuration, then a DECERR response is generated for any requests that target that region.

The example decoder uses a stripe size of 256 bytes.

Appendix A
Signal Descriptions

This appendix describes the external signals of the CoreLink CCI-500 Cache Coherent Interconnect.

It contains the following sections:

- *A.1 Clock and reset signals* on page Appx-A-74.
- *A.2 Power and clock control signals* on page Appx-A-75.
- *A.3 Configuration signals* on page Appx-A-77.
- *A.4 Debug signals* on page Appx-A-79.
- *A.5 DFT signals* on page Appx-A-80.
- *A.6 APB4 signals* on page Appx-A-81.
- *A.7 ACE and ACE-Lite slave interface signals* on page Appx-A-82.
- *A.8 AXI Master interface signals* on page Appx-A-87.
- *A.9 Miscellaneous signals* on page Appx-A-90.

A.1 Clock and reset signals

The CCI-500 uses a single set of standard clock and reset signals.

The following table shows the clock and reset signals.

Table A-1 Clock and reset signals

Signal	Direction	Description
ACLK	Input	Global clock.
ARESETn	Input	Global reset.

A.2 Power and clock control signals

The CCI-500 uses a range of signals to communicate with the Q-Channel and P-Channel interfaces.

The following table shows the power and clock control signals.

Table A-2 Power and clock control signals

Signal	Direction	Description
AWAKEUPSx	Input	HIGH when transfers are pending on the AR, AW, or W channels of the associated slave interface. If any of these inputs is HIGH, the CCI-500 takes **ACLKQACTIVE** HIGH to request that the CCI clock is enabled. There is one input for each slave interface. Where x = 0-6, depending on the configuration. **Note** The number of interfaces is 2-7.
AWAKEUPMy	Output	HIGH when transfers are pending on the AR, AW, or W channels of the associated master interface. You can use this signal to request that the clock is turned on to downstream components. There is one output for each master interface. Where y = 0-5, depending on the configuration. **Note** The number of interfaces is 1-6.
PWAKEUP	Input	Indicates that the APB interface requires a clock because a transaction is incoming.
ACLKQREQn	Input	Request to disable the **ACLK** input. If the clock control channel is not used, then tie **ACLKQREQn** HIGH.
ACLKQACCEPTn	Output	Clock disable acceptance response.
ACLKQDENY	Output	Clock disable denial response.
ACLKQACTIVE	Output	Indicates that the CCI-500 requires the **ACLK** input to run.
PREQ	Input	Request to change power state.
PSTATE[2:0]	Input	Required power state. The encodings for this are: `0b000` Off. `0b001` Static snoop filter RAM retention. `0b010` Reserved. `0b011` Dynamic snoop filter RAM retention. `0b100` On. `0b101-0b111` Reserved. If the P channel is not used, you must tie **PSTATE** to `0b100`, On state.
PACCEPT	Output	Power state transition acceptance.

Table A-2 Power and clock control signals (continued)

Signal	Direction	Description
PDENY	Output	Power state transition denial.
PACTIVE[4:0]	Output	Hint from the CCI-500 to indicate the power states that it can accept. Each bit corresponds to a power state, if HIGH then that state is a legal power transition: `[0]` Off. `[1]` Static snoop filter RAM retention. `[2]` Reserved. `[3]` Dynamic snoop filter RAM retention. `[4]` On.

A.3 Configuration signals

The CCI-500 samples configuration signals only when the **ARESETn** signal transitions from LOW to HIGH.

The following table shows the configuration signals.

Table A-3 Configuration signals

Signal	Direction	Description
ADDRMAPx[ADDRMAP_WIDTH-1:0]	Input	Configuration inputs that you can use to define the mapping scheme of the address decoder. In the example decoder, there are 3 bits for each of the possible nine address regions. **Note** It is the reset sampled version of the ADDRMAP that is passed to the address decode irrespective of whether it is the ARM supplied address map or a modified version.
QOSOVERRIDE[n:0]	Input	If HIGH, the internally generated values override the **ARQOS** and **AWQOS** input signals. See *2.4.12 Quality of Service* on page 2-40 for more information. One bit exists for each slave interface.
ACCHANNELENSx[1:0] for ACE interfaces **ACCHANNELENSx[0]** for ACE-Lite interfaces	Input	AC channel enables, one input per slave interface. These inputs override any software enables. **Bit[0], DVM message enable** This bit is encoded as follows: **0** DVM messages disabled. **1** DVM messages enabled. **Bit[1], Snoop enable** This bit applies to ACE interfaces only, and is encoded as follows: **0** Snoop requests disabled. **1** Snoop requests enabled. **Note** Snoops and DVM messages must still be enabled in the Snoop Control Registers.

Table A-3 Configuration signals (continued)

Signal	Direction	Description
ORDERED_WRITE_OBSERVATION[n:0]	Input	Controls whether an ACE-Lite slave interface supports the Ordered Write Observation property. **Note** ACE interfaces do not support the Ordered Write Observation property. This input is ignored for ACE slave interfaces. This bit is encoded as follows: **0** Interface does not support Ordered Write Observation. **1** Interface supports Ordered Write Observation. One bit exists for each slave interface.
BURST_SPLIT_ALL[n:0]	Input	If HIGH, all incoming requests are split into 64-byte transfers, rather than shareable requests only. This signal has no effect on an interface where the `SIx_BURST_SPLITTER` parameter is set to 0. One bit exists for each ACE-Lite slave interface.
NSAID_ENABLED_Sx	Input	If HIGH, indicates that this interface is protected under TZMP1 and uses NSAID. Snoop data from an ACE interface is not returned directly to an initiator. One input signal exists for each ACE-Lite slave interface. This signal is present only when the `non_secure_access_ID_support` configuration parameter is set to 1.
MI_DEPENDENT_ON_SI_Mx	Input	If HIGH, indicates that this master interface is connected to a component with both slave and master interfaces, where there is a dependency between them. For example, a PCIe root complex usually has a slave interface where completion of a write is dependent on the progress of transactions on its master interface.

A.4 Debug signals

The inputs can change at runtime and you must synchronize them to the CCI-500 clock to prevent timing hazards.

The following table shows the debug signals.

Table A-4 Debug signals

Signal	Direction	Description
NIDEN	Input	Non-invasive debug enable. If HIGH, the signal enables counting and export of PMU events.
SPNIDEN	Input	Secure privileged non-invasive debug enable. If both **SPNIDEN** and **NIDEN** are HIGH, the signal enables counting of both Non-secure and Secure events.
DBGEN	Input	Invasive debug enable. If HIGH, enables the counting and export of PMU events.
SPIDEN	Input	Secure invasive debug enable. If both **SPIDEN** and **DBGEN** are HIGH, enables the counting of both Non-secure and Secure events.
EVNTBUS[n:0]	Output	The CCI-500 events exported if enabled in the PMCR. See *PMU event list* on page 2-31 for information on pin allocations of this vector. The vector width is dependent on the number of master interfaces, M, and the number of slave interfaces, S, and is defined as: S*32 + M*7 + 15.
nEVNTCNTOVERFLOW[7:0]	Output	Overflow flags for the PMU clock and counters. This is an active-LOW signal. Each bit represents the overflow for the event counter with that number.
nERRIRQ	Output	Indicates that an error response, DECERR or SLVERR, is received on the **RRESP**, **BRESP**, or **CRRESP** input signals, and it cannot be signaled precisely. If LOW, the signal indicates that an error has occurred.

See the *ARM® CoreSight™ Architecture Specification* for more information.

A.5 DFT signals

The CCI-500 uses the *Design For Test* (DFT) signals to communicate with the DFT and MBIST interfaces.

The following table shows the DFT signals.

Table A-5 DFT signals

Signal	Direction	Description
DFTRSTDISABLE	Input	Disables reset during scan shift.
DFTCGEN	Input	Assert HIGH during scan shift to enable architectural clock gates for **ACLK** clocks.
DFTRAMHOLD	Input	Blocks chip select to RAMs to preserve state.
DFTMCPHOLD	Input	Limits number of multi-cycle path toggles during ATPG delay test.
nMBISTRESET0	Input	Resets MBIST mode.
MBISTREQ0	Input	Selects MBIST mode.
MBISTACK0	Output	Acknowledges MBIST mode.

A.6 APB4 signals

The following table shows the APB4 Slave Interface signals.

Table A-6 APB4 signals

Signal	Direction	Description
PADDR[31:0]	Input	Address.
PPROT[2:0]	Input	Protection type.
PSEL	Input	Peripheral select.
PENABLE	Input	Enable for transfer.
PWRITE	Input	Write transaction indicator.
PWDATA[31:0]	Input	Write data.
PSTRB[3:0]	Input	Write data strobe.
PREADY	Output	Transfer ready.
PRDATA[31:0]	Output	Read data.
PSLVERR	Output	Error response.

A.7 ACE and ACE-Lite slave interface signals

The CCI-500 has a configurable number of ACE and ACE-Lite slave interfaces. The suffix is Sx, where x is 0-6, depending on the configuration.

This section contains the following subsections:

- *A.7.1 Write address channel signals* on page Appx-A-82.
- *A.7.2 Write data channel signals* on page Appx-A-82.
- *A.7.3 Write data response channel signals* on page Appx-A-83.
- *A.7.4 Read address channel signals* on page Appx-A-83.
- *A.7.5 Read data channel signals* on page Appx-A-84.
- *A.7.6 Coherency address channel signals* on page Appx-A-84.
- *A.7.7 Coherency response channel signals* on page Appx-A-85.
- *A.7.8 Coherency data channel signals for ACE interfaces* on page Appx-A-85.
- *A.7.9 Acknowledge signals for ACE interfaces* on page Appx-A-85.

A.7.1 Write address channel signals

These signals carry control information that describes the nature of the data to be transferred. The data is transferred between master and slave using either a read data channel or a write data channel.

The following table shows the write address channel signals.

Table A-7 Write address channel signals

Signal	Direction	Description
AWIDSx[n:0]	Input	Write address ID. You can configure the width of this signal.
AWADDRSx[n:0]	Input	Write address. You can configure the CCI-500 to support between a 32-bit and a 44-bit signal width.
AWREGIONSx[3:0]	Input	Write address region. You can tie this signal LOW if the master does not drive it.
AWLENSx[7:0]	Input	Write burst length.
AWSIZESx[2:0]	Input	Write burst size.
AWBURSTSx[1:0]	Input	Write burst type.
AWLOCKSx	Input	Write lock type.
AWCACHESx[3:0]	Input	Write cache type.
AWPROTSx[2:0]	Input	Write protection type.
AWSNOOPSx[2:0]	Input	Write snoop request type.
AWDOMAINSx[1:0]	Input	Write domain.
AWQOSSx[3:0]	Input	Write QoS value.
AWUSERSx[n:0]	Input	Specified extension to AW payload. You can configure the width of this signal.
NSAIDWSx[3:0]	Input	Optional extension to AW payload, that transmits the Non-secure access identifier for a request.
AWVALIDSx	Input	Write address valid.
AWREADYSx	Output	Write address ready.

A.7.2 Write data channel signals

Write data channel signals carry the write data from the master to the slave, and include the data bus and a byte lane strobe signal.

The following table shows the write data channel signals.

Table A-8 Write data channel signals

Signal	Direction	Description
WDATASx[127:0]	Input	Write data.
WSTRBSx[15:0]	Input	Write byte-lane strobes.
WLASTSx	Input	Write last. This signal indicates the last transfer in a write burst.
WUSERSx[n:0]	Input	The specified extension to the W payload.
WCHECKSUMSx[n:0]	Input	An optional extension to the W payload that can transmit checksum or parity information for the data. You can configure the width of this signal.
WVALIDSx	Input	Write data is valid.
WREADYSx	Output	Write data is ready.

A.7.3 Write data response channel signals

A slave uses the write response channel to respond to write transactions. All write transactions require completion signaling on the write response channel.

The following table shows the write data response channel signals.

Table A-9 Write data response channel signals

Signal	Direction	Description
BIDSx[n:0]	Output	Write response ID. You can configure the width.
BRESPSx[1:0]	Output	Write response.
BUSERSx[n:0]	Output	The specified extension to the B payload.
BVALIDSx	Output	Write response is valid.
BREADYSx	Input	Write response is ready.

A.7.4 Read address channel signals

The following table shows the read address channel signals.

Table A-10 Read address channel signals

Signal	Direction	Description
ARIDSx[n:0]	Input	Read address ID. You can configure the width of this signal.
ARADDRSx[n:0]	Input	Read address. You can configure the CCI-500 to support 40-bit [39:0], or 44-bit [43:0] DVM transactions.
ARREGIONSx[3:0]	Input	Read address region. You can tie this signal LOW if the master does not drive it.
ARLENSx[7:0]	Input	Read burst length.
ARSIZESx[2:0]	Input	Read burst size.
ARBURSTSx[1:0]	Input	Read burst type.
ARLOCKSx	Input	Read lock type.

Table A-10 Read address channel signals (continued)

Signal	Direction	Description
ARCACHESx[3:0]	Input	Read cache type.
ARPROTSx[2:0]	Input	Read protection type.
ARDOMAINSx[1:0]	Input	Read domain.
ARSNOOPSx[3:0]	Input	Read snoop request type.
ARQOSSx[3:0]	Input	Read QoS.
ARUSERSx[n:0]	Input	The specified extension to the AR payload.
NSAIDRSx[3:0]	Input	Optional extension to AR payload, that transmits the Non-secure access identifier for a request.
ARVALIDSx	Input	Read address is valid.
ARREADYSx	Output	Read address is ready.

A.7.5 Read data channel signals

Read data channel signals carry the read data and the read response information from the slave to the master, and include the data bus and a read response signal.

The following table shows the read data channel signals.

Table A-11 Read data channel signals

Signal	Direction	Description
RIDSx[n:0]	Output	Read data ID. You can configure the width of this signal.
RDATASx[127:0]	Output	Read data.
RRESPSx[3:0]	Output	Read data response for ACE interfaces S3 and S4.
RRESPSx[1:0]	Output	Read data response for ACE-Lite interfaces S0, S1, and S2.
RLASTSx	Output	Read last. This signal indicates the last transfer in a read burst.
RUSERSx[n:0]	Output	The specified extension to the R payload.
RCHECKSUMSx[n:0]	Output	An optional extension to the R payload that can transmit checksum or parity information for the data. You can configure the width of this signal.
RVALIDSx	Output	Read data is valid.
RREADYSx	Input	Read data is ready.

A.7.6 Coherency address channel signals

Coherency address channel signals provide address and associated control information for snoop transactions.

The following table shows the coherency address channel signals.

Table A-12 Coherency address channel signals

Signal	Direction	Description
ACADDRSx[n:0]	Output	Snoop address. You can configure the CCI-500 to support 40-bit [39:0], or 44-bit [43:0] snoop transactions.
ACPROTSx[2:0]	Output	Snoop protection type.
ACSNOOPSx[3:0]	Output	Snoop request type.
ACVALIDSx	Output	Snoop address is valid.
ACREADYSx	Input	The master interface is ready to accept the snoop address.

A.7.7 Coherency response channel signals

Coherency response channel signals provide the response to snoop transactions.

The following table shows the coherency response channel signals.

Table A-13 Coherency response channel signals

Signal	Direction	Description
CRRESPSx[4:0]	Input	Snoop response.
NSAIDCRSx[3:0]	Input	Optional extension to CR payload, that transmits the Non-secure access identifier for a snoop response.
CRVALIDSx	Input	Snoop response is valid.
CRREADYSx	Output	The slave interface is ready to accept the snoop response.

A.7.8 Coherency data channel signals for ACE interfaces

Coherency data channel signals, if used, pass snoop data out from an ACE master.

The following table shows the coherency data channel signals for ACE interfaces.

Table A-14 Coherency data channel signals, ACE interfaces

Signal	Direction	Description
CDDATASx[127:0]	Input	Snoop data.
CDLASTSx	Input	Snoop data last transfer.
CDCHECKSUMSx[n:0]	Input	An optional extension to the CD payload that can transmit checksum or parity information for the data. You can configure the width of this signal.
CDVALIDSx	Input	Snoop data is valid.
CDREADYSx	Output	The slave interface is ready to accept snoop data.

A.7.9 Acknowledge signals for ACE interfaces

Acknowledge signals indicate that a master or slave has completed a read or write transaction.

The following table shows the acknowledge signals for ACE interfaces.

Table A-15 Acknowledge signals, ACE interfaces

Signal	Direction	Description
RACKSx	Input	Read acknowledge.
WACKSx	Input	Write acknowledge.

A.8 AXI Master interface signals

The CCI-500 has a configurable number of AXI4 master interfaces. The suffix is `My`, where `y` is 0-5, depending on the configuration.

This section contains the following subsections:

- *A.8.1 Write address channel signals* on page Appx-A-87.
- *A.8.2 Write data channel signals* on page Appx-A-87.
- *A.8.3 Write data response channel signals* on page Appx-A-88.
- *A.8.4 Read address channel signals* on page Appx-A-88.
- *A.8.5 Read data channel signals* on page Appx-A-89.

A.8.1 Write address channel signals

These signals carry control information that describes the nature of the data to be transferred. The data is transferred between master and slave using either a read data channel or a write data channel.

The following table shows the write address channel signals.

Table A-16 Write address channel signals

Signal	Direction	Description
AWIDMy[n:0]	Output	Write address ID. The width is the maximum **AWID** signal width across the slave interfaces + 3 bits, and it is a minimum of 9 bits.
AWADDRMy[n:0]	Output	Write address. You can configure the CCI-500 to support between a 32-bit and a 44-bit signal width.
AWREGIONMy[3:0]	Output	Write address region.
AWLENMy[7:0]	Output	Write burst length.
AWSIZEMy[2:0]	Output	Write burst size.
AWBURSTMy[1:0]	Output	Write burst type.
AWLOCKMy	Output	Write lock type.
AWCACHEMy[3:0]	Output	Write cache type.
AWPROTMy[2:0]	Output	Write protection type.
AWQOSMy[3:0]	Output	Write QoS value.
AWUSERMy[n:0]	Output	The specified extension to the AW payload.
AWVALIDMy	Output	Write address is valid.
NSAIDWMy[3:0]	Output	Optional extension to AW payload, that transmits the Non-secure access identifier for a request.
AWREADYMy	Input	Write address is ready.

A.8.2 Write data channel signals

Write data channel signals carry the write data from the master to the slave, and include the data bus and a byte lane strobe signal.

The following table shows the write data channel signals where **y** represents the master interface number.

Table A-17 Write data channel signals

Signal	Direction	Description
WIDMy[n:0]	Output	Write ID tag. Included to help connection to AXI3 slave interfaces.
WDATAMy[127:0]	Output	Write data.
WSTRBMy[15:0]	Output	Write strobes.
WLASTMy	Output	Write last.
WUSERMy[n:0]	Output	User signal.
WCHECKSUMMy[n:0]	Output	An optional extension to the W payload that you can use to transmit checksum or parity information for the data. You can configure the width of this signal.
WVALIDMy	Output	Write valid.
WREADYMy	Input	Write ready.

A.8.3 Write data response channel signals

A slave uses the write response channel to respond to write transactions. All write transactions require completion signaling on the write response channel.

The following table shows the write data response channel signals.

Table A-18 Write data response channel signals

Signal	Direction	Description
BIDMy[n:0]	Input	Write response ID.
BRESPMy[1:0]	Input	Write response.
BUSERMy[n:0]	Input	The specified extension to the B payload.
BVALIDMy	Input	Write response is valid.
BREADYMy	Output	Write response is ready.

A.8.4 Read address channel signals

These signals carry control information that describes the nature of the data to be transferred. The data is transferred between master and slave using either a read data channel or a write data channel.

The following table shows the read address channel signals.

Table A-19 Read address channel signals

Signal	Direction	Description
ARIDMy[n:0]	Output	Read address ID. The width is the maximum **ARID** signal width across slave interfaces + 3 bits, and it is a minimum of 6 bits.
ARADDRMy[n:0]	Output	Read address. You can configure the CCI-500 to support between a 32-bit and a 44-bit signal width.
ARREGIONMy[3:0]	Output	Read address region.
ARLENMy[7:0]	Output	Read burst length.
ARSIZEMy[2:0]	Output	Read burst size.
ARBURSTMy[1:0]	Output	Read burst type.

Table A-19 Read address channel signals (continued)

Signal	Direction	Description
ARLOCKMy	Output	Read lock type.
ARCACHEMy[3:0]	Output	Read cache type.
ARPROTMy[2:0]	Output	Read protection type.
ARQOSMy[3:0]	Output	Read QoS value.
ARUSERMy[n:0]	Output	The specified extension to the AR payload.
ARVALIDMy	Output	Read address is valid.
NSAIDRMy[3:0]	Output	Optional extension to AR payload, that transmits the Non-secure access identifier for a request
ARREADYMy	Input	Read address is ready.

A.8.5 Read data channel signals

Read data channel signals carry the read data and the read response information from the slave to the master.

The following table shows the read data channel signals.

Table A-20 Read data channel signals

Signal	Direction	Description
RIDMy[n:0]	Input	Read data ID.
RDATAMy[127:0]	Input	Read data.
RRESPMy[3:0]	Input	Read data response.
RLASTMy	Input	Read last. This signal indicates the last transfer in a read burst.
RUSERMy[n:0]	Input	The specified extension to the R payload.
RCHECKSUMMy[n:0]	Input	An optional extension to the R payload that you can use to transmit checksum or parity information for the data. You can configure the width of this signal.
RVALIDMy	Input	Read data is valid.
RREADYMy	Output	Read data is ready.

A.9 Miscellaneous signals

Some signals in the CCI-500 are not applicable for general operation, implementation, or debug of the CCI-500. However, if there is a problem to resolve, you might require these signals.

The following table shows a signal that you can use to indicate an ECO applies to this implementation of the CCI-500.

Table A-21 Miscellaneous signals

Signal	Direction	Description
ECOREVNUM[3:0]	Input	Use this signal to update the revision field register if you have an ECO to apply to the CCI-500. See *ECO revision number* on page 3-59.

Appendix B
Revisions

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

It contains the following sections:

- *B.1 Revisions* on page Appx-B-92.

B.1 Revisions

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

Table B-1 Issue 0000-00

Change	Location	Affects
First release	-	-

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

Change	Location	Affects
Clarified information about adding and removing masters from the coherent domain.	*2.4.3 Snoop connectivity and control* on page 2-29.	All revisions.
Updated information about relationship between debug enable inputs and PMCR settings.	*3.3.5 Performance Monitor Control Register (PMCR)* on page 3-57.	All revisions.

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

Change	Location	Affects
Clarified description of snoop filter directory configuration.	*2.4.2 Snoop filter* on page 2-28.	All revisions.
Clarified procedure for removing a master from the coherent domain.	*2.4.3 Snoop connectivity and control* on page 2-29.	All revisions.
Clarified the example of how to use the PMU.	*2.4.4 Performance Monitoring Unit* on page 2-30.	All revisions.
Amended register names to match RTL names.	*3.2 Register summary* on page 3-44.	All revisions.
Revised peripheral_id2 entry.	*3.2 Register summary* on page 3-44	r0p1.
Amended some register bit names and field names for consistency with other ARM documents.	*3.3 Register descriptions* on page 3-50.	All revisions.
Clarified configurations descriptions to indicate that certain registers have multiple copies.	*3.3 Register descriptions* on page 3-50.	All revisions.
Amended usage constraints. Amended access permissions of bits[15:11].	*3.3.5 Performance Monitor Control Register (PMCR)* on page 3-57.	r0p1.
Amended usage constraints.	*3.3.6 Interface Monitor Control Register* on page 3-58	r0p1.
Revised peripheral_id2.	*3.3.7 Component and Peripheral ID Registers* on page 3-58	r0p1.
Clarified bit assignment descriptions.	*3.3.8 Snoop Control Registers* on page 3-59.	All revisions.
Clarified bit assignment descriptions for bits[7:0].	*3.3.17 Slave Interface Monitor Registers* on page 3-66.	All revisions.
Clarified bit assignment descriptions for bits[4:0].	*3.3.18 Master Interface Monitor Registers* on page 3-68.	All revisions.
Amended example address map.	*3.4 Address map* on page 3-70.	All revisions.
Added note to clarify description of `0b111` setting for **ADDRMAP[2:0]**.	*3.4 Address map* on page 3-70.	All revisions.

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

Change	Location	Affects
Clarified **ORDERED_WRITE_OBSERVATION[n:0]** description.	*A.3 Configuration signals* on page Appx-A-77	r0p1.
Added **NSAID_ENABLED_Sx** signal.	*A.3 Configuration signals* on page Appx-A-77	All revisions.
Added **MI_DEPENDENT_ON_SI_Mx** signal.	*A.3 Configuration signals* on page Appx-A-77	r0p1.