ARM® CoreLink™ GIC-500 Generic Interrupt Controller

Revision: r0p0

Technical Reference Manual

ARM®

ARM DDI 0516B (ID060914)

ARM CoreLink GIC-500 Generic Interrupt Controller
Technical Reference Manual

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
30 April 2014	A	Non-Confidential	First release for r0p0
21 May 2014	B	Non-Confidential	Second release for r0p0

Proprietary Notice

Confidentiality Status

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

Product Status

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

Web Address

`http://www.arm.com`

Contents
ARM CoreLink GIC-500 Generic Interrupt Controller Technical Reference Manual

Preface

This preface introduces the *ARM® CoreLink™ GIC-500 Generic Interrupt Controller Technical Reference Manual*. It contains the following sections:

About this book

This technical reference manual is for the CoreLink GIC-500 *Generic Interrupt Controller* (GIC).

Product revision status

The r*m*p*n* identifier indicates the revision status of the product described in this book, for example, r1p2, 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 for system designers, system integrators, and programmers who are designing or programming a *System-on-Chip* (SoC) that uses the GIC-500.

Using this book

This book contains the following chapters:

Chapter 1 *Introduction*

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

Chapter 2 *Functional Description*

Read this for a description of the major interfaces and for the implementation-defined behavior of the GIC-500.

Chapter 3 *Programmers Model*

Read this for a description of the memory map and registers, and for information about programming the device.

Appendix A *Signal Descriptions*

Read this for a description of the input and output signals.

Appendix B *Revisions*

Read this for a description of the technical changes between released issues of this book.

Glossary

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

See the ARM® Glossary
http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html.

Conventions

This book uses the conventions that are described in:

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

Typographical conventions

The following table describes the typographical conventions:

Typographical conventions

Style	Purpose
italic	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`mono`space	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace` *`italic`*	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Signals

The signal conventions are:

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

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

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

Additional reading

This section lists publications by ARM and by third parties.

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

ARM publications

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

- *ARM® AMBA® AXI and ACE Protocol Specification* (ARM IHI 0022).
- *ARM® AMBA® 4 AXI4-Stream Protocol Specification* (ARM IHI 0051).
- *ARM® Generic Interrupt Controller Stream Protocol Interface Specification* (ARM IHI 0066).
- *ARM® Generic Interrupt Controller Architecture Specification version 3.0 and version 4.0* (ARM IHI 0069).
- *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architectural profile* (ARM DDI 0487).

The following confidential books are only available to licensees:

- *ARM® CoreLink™ GIC-500 Generic Interrupt Controller Implementation Guide* (ARM DII 0288).
- *ARM® CoreLink™ GIC-500 Generic Interrupt Controller Integration Manual* (ARM DIT 0050).

Other publications

This section lists relevant documents published by third parties:

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

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

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

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

Feedback on content

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

- The title.
- The number, ARM DDI 0516B.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

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

Chapter 1
Introduction

This chapter introduces the GIC-500. It contains the following sections:

1.1 About the GIC-500

The GIC-500 is a build-time configurable interrupt controller that supports up to 128 cores. The GIC-500 only supports cores that implement the ARMv8 architecture and the GIC CPU interface with the standard GIC Stream Protocol interface, such as Cortex®-A57 and Cortex-A53. It implements the *ARM® Generic Interrupt Controller Architecture Specification version 3.0*, to enable support for:

- ARMv8 cores.
- Physical interrupt signals.
- *Software Generated Interrupts* (SGIs).
- Interrupts generated by writing to the AXI4 slave port, known as message-based interrupts.
- An *Interrupt Translation Service* (ITS) that provides ID translation and core migration for message-based interrupts.

Figure 1-1 on page 1-3 depicts the GIC-500 in an example system.

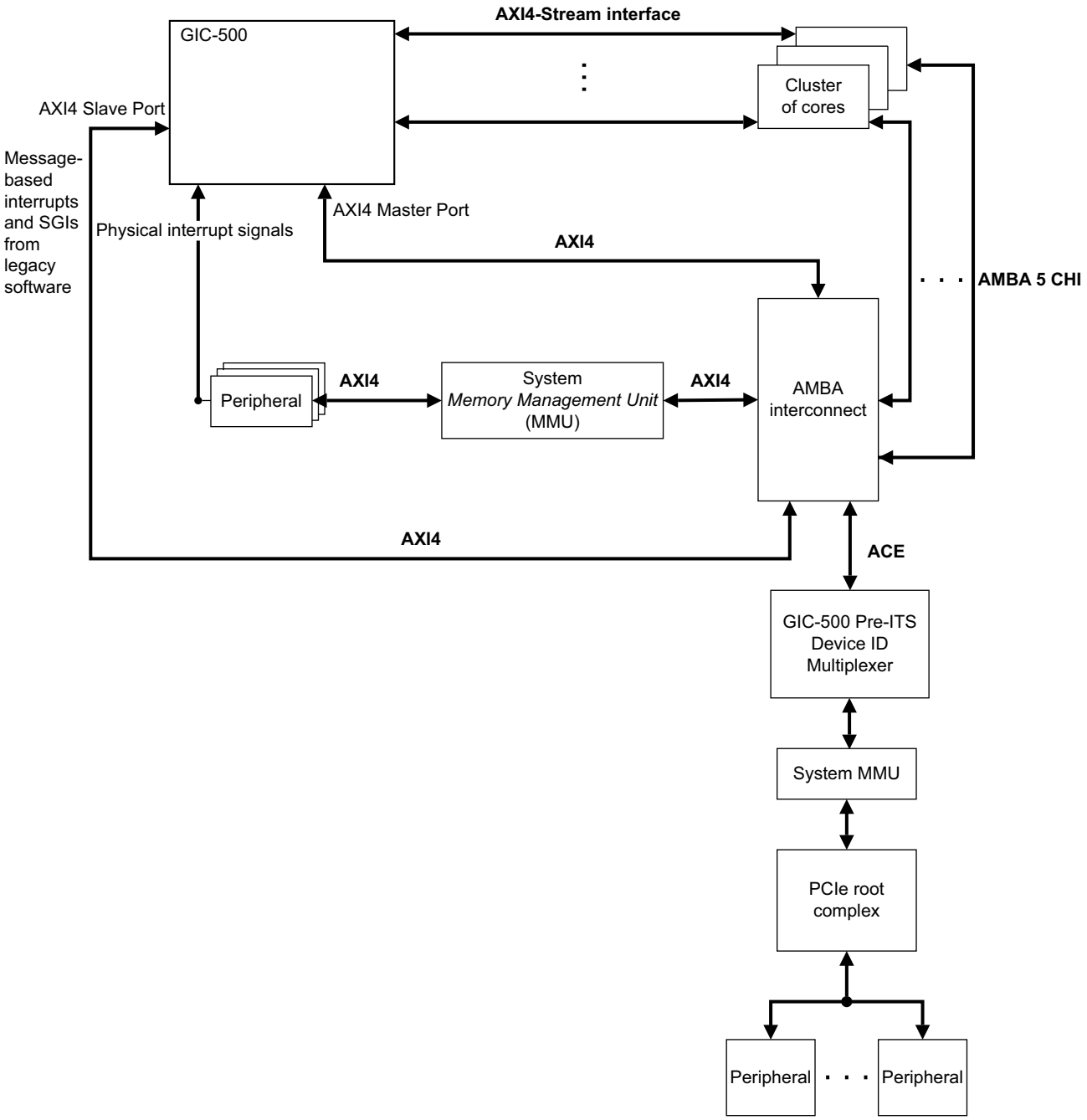

Figure 1-1 GIC-500 in an example system

The GIC-500 receives interrupts from the AXI4 slave, or from physical inputs, depending on the type of interrupt. The GIC-500:

- Supports a few different types of interrupt with different characteristics. See *Interrupt types* on page 2-7.
- Prioritizes the interrupts and ensures that the highest priority pending interrupt is sent to the CPU interface. The CPU interface is part of the interrupt controller which is not part of the GIC-500, but is instead part of compatible ARMv8 cores. ARM recommends the

CPU interfaces are programmed using System register accesses, although legacy software might program them using memory-mapped accesses, depending on what the core supports.

- Connects to these CPU interfaces using dedicated AXI4-Stream interfaces.
- Is programmed using its AXI4 Slave port.
- Supports virtualization of interrupts for each connected CPU interface that provides this feature.
- ITS provides interrupt ID translation that can allow peripherals to be programmed by a virtual machine directly.
- Provides registers for managing interrupt sources, interrupt behavior, and interrupt routing to one or more cores.

Connected ARM cores have System registers that provide the CPU interface to the GIC.

1.1.1 Topologies and terminology

Figure 1-2 shows a diagrammatic representation of the Redistributor hierarchy.

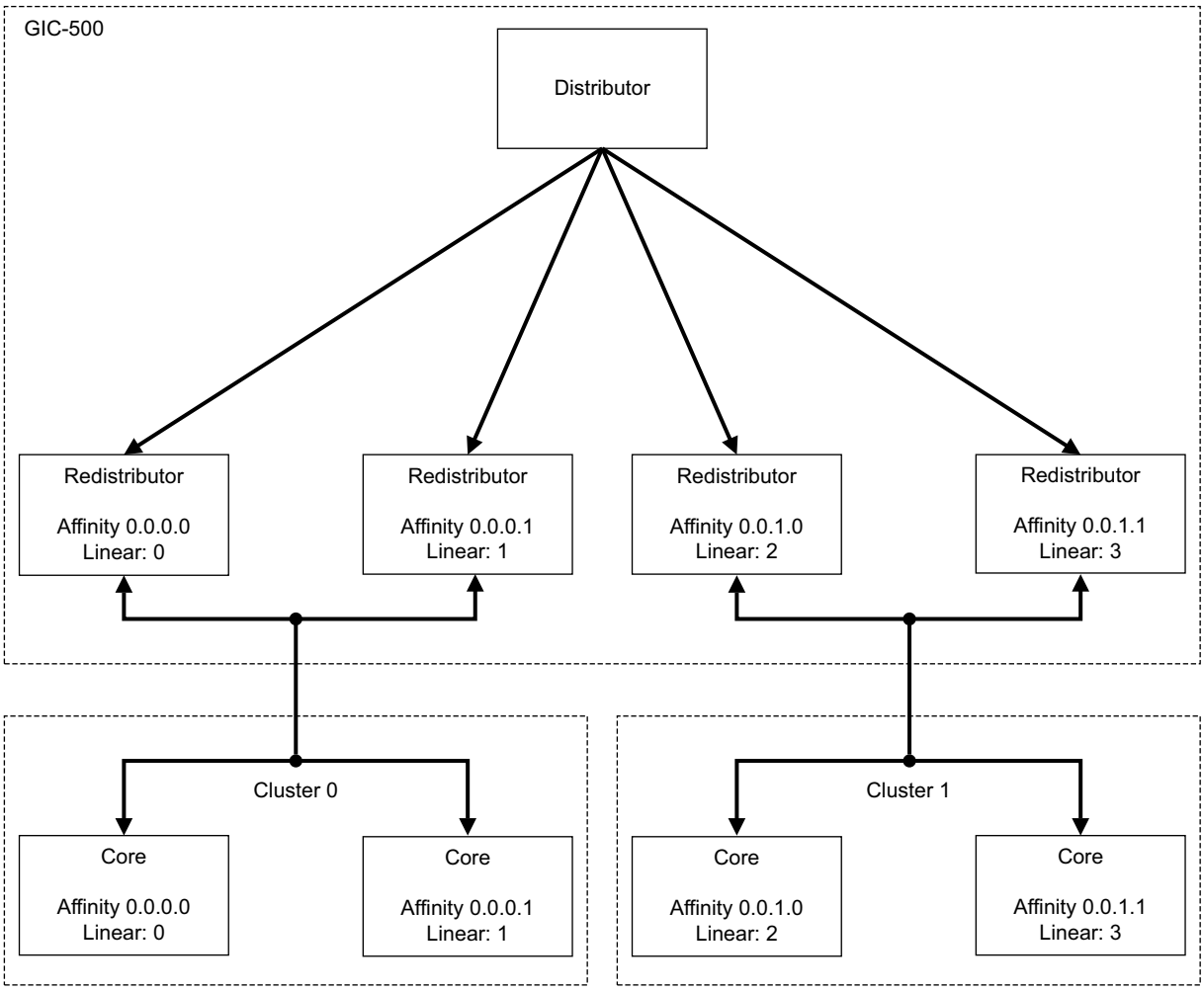

Figure 1-2 Redistributor hierarchy

Note

The linear representation indicates the order of the cores, that is sorted by the affinity values. For example 0.0.1.1 is given the value of 3 because it is the fourth lowest affinity value in the system.

The ARM architecture defines a register in a core that identifies the logical address of the core in the system. This register, known as the *Multiprocessor Affinity Register* (MPIDR), has a hierarchical format. Each level of the hierarchy is known as an affinity level, with the highest affinity levels specified first:

- For an ARMv7 processor, the MPIDR defines three levels of affinity, with an implicit affinity level 3 value of 0.
- For an ARMv8 processor, the MPIDR defines four levels of affinity.

This means the affinity can be specified, using a four-field dot-decimal notation, as <Aff3>.<Aff2>.<Aff1>.<Aff0>, where Affn is a value for affinity level n.

For most processors, for example the Cortex-A57 MPCore processor, the processor is made up from a cluster of cores that have a common affinity level 1. Each core must have a different MPIDR value, so the cores have different affinity level 0 values, beginning from 0. For example, a processor with four cores might have MPIDR values in the range 0.0.0.0-0.0.0.3. If there were two processors then the second processor might have values in the range 0.0.1.0-0.0.1.3. Typically the processor number is affinity level 1 and the core number is affinity level 0.

The GIC-500 only supports topologies where affinity levels 2 and above are the same. That is, all cores must have MPIDR values of the form 0.0.c.d, where c and d are variables. The range for c is assumed to start at 0 and be contiguous. The range for d is also assumed to start at 0 and be contiguous for each c. For example, the first processor must have IDs 0.0.0.0 to 0.0.0.x and the second processor must have IDs 0.0.1.0 to 0.0.1.y.

The GIC-500 supports up to 128 cores in up to 32 processors with a limit of eight cores per processor, and has a pair of AXI4-Stream interfaces for each processor.

When using backwards compatibility mode, GIC-500 supports the first eight cores. That is, it supports the eight cores with the lowest MPIDR values, whether they are in the same processor or not.

GIC-500 provides a Redistributor for each core, each with a corresponding set of registers. Many registers in the GIC-500 use the MPIDR value to specify cores. However, programming the Interrupt Translation Service and integrating the AXI4 slave port require an alternative linear representation. This representation is based on numbering the cores in order of increasing affinity level, starting at 0. For example, in backwards compatibility mode, GIC-500 supports cores 0-7 in linear representation.

1.2 Compliance

The GIC-500 is compliant with the following interfaces and specifications:

- The AMBA AXI4 protocol. See the *ARM® AMBA® AXI and ACE Protocol Specification*, and *AXI4 slave interface signals* on page A-6.
- The GIC Stream protocol is based on the following specifications:
  - *ARM® AMBA® 4 AXI4-Stream Protocol Specification*.
  - *ARM® Generic Interrupt Controller Stream Protocol Interface Specification*.
- Version 3.0 of the ARM GIC architecture specification. See the *ARM® Generic Interrupt Controller Architecture Specification version 3.0.*

1.3 Features

The GIC-500 provides registers for managing interrupt sources, interrupt behavior, and interrupt routing to one or more cores. It supports:

- Multiprocessor environments with up to 128 cores.
- Up to 32 affinity-level 1 clusters.
- Up to eight cores for each cluster.
- The following interrupt types:
  - *Locality-specific Peripheral Interrupt*s (LPIs). These interrupts are generated by a peripheral writing to a memory-mapped register in the GIC-500. See *Configurable options for the GIC-500 RTL* on page 1-9.
  - *Shared Peripheral Interrupts* (SPIs). See *Configurable options* on page 1-9.
  - 16 *Private Peripheral Interrupts* (PPIs), that are independent for each core and can be programmed to support either edge-triggered or level-sensitive interrupts.
  - 16 SGIs, that are generated either by using software to write to GICD_SGIR or through the GIC CPU interface of a core.
- *Interrupt Translation Service* (ITS). This provides device isolation and ID translation for message-based interrupts, which allows virtual machines to program devices directly.
- Memory-mapped access to all registers.
- Interrupt masking and prioritization.
- Programmable interrupt routing based on affinity.
- Three different interrupt groups, which allow interrupts to target different exception levels:
  - Group 0.
  - Non-secure Group 1.
  - Secure Group 1.
- A global *Disable Security* (DS) bit. This allows support for systems with and without security.
- 32 priority values, five bits for each interrupt.

1.4 Interfaces

The GIC-500 provides the following external interfaces:

- *AXI4 Slave Interface* on page 2-3.
- *AXI4 Master Interface* on page 2-4.
- *RAM Interfaces* on page 2-5.
- *Physical interrupt signals* on page 2-5.
- *GIC-500 Stream Protocol Interface* on page 2-6.
- *Other core signals* on page 2-6.

1.5 Configurable options

Table 1-1 shows the configurable options in the GIC-500 RTL.

Table 1-1 Configurable options for the GIC-500 RTL

Feature	Range of options
Number of affinity-level 1 clusters.	1-32
Number of cores for each cluster. This can be different for each cluster.	1-8
Read Address ID width.	1-32
Write Address ID width.	1-32
Number of SPIs.	32-960[a]
GICv2 backwards compatibility support.	Options include: • Both Secure and Non-secure AREs are always set in GICD_CTLR. There is no backwards compatibility. • Both Secure and Non-secure AREs are programmable. The GIC-500 resets to backward compatible mode. See *Backwards compatibility* on page 2-10 for more information.
Security Support.	Options include: • Security support programmable. Resets to supporting security. • Security support not present.
ITS and LPI support.	Options include: • ITS is present. LPIs are supported. • ITS is not present. LPIs are not supported.
ITS Device ID width.	3-20 See *AXI4 Slave Interface* on page 2-3 for more information.
Number of LPI cache entries.	16-1024 (powers of two only).

a. Range from 32-960, with increments of 32.

1.6 Test features

The GIC-500 provides *Design For Test* (DFT) signals for test mode. See *Test signals* on page A-5.

1.7 Product documentation

This section describes the GIC-500 documentation, how it relates to the design flow, and the relevant architectural standards and protocols.

Technical Reference Manual

The *Technical Reference Manual* (TRM) describes the functionality and the effects of functional options on the behavior of the GIC-500. It is required at all stages of the design flow. Some behavior described in the TRM might not be relevant because of the way that the GIC-500 is implemented and integrated. If you are programming the GIC-500 then contact:

- The implementer to determine the build configuration of the implementation.
- The integrator to determine the signal configuration of the SoC that you are using.

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

Implementation Guide

The *Implementation Guide* (IG) describes:

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

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

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

Integration Manual

The *Integration Manual* (IM) describes how to integrate the GIC-500 into a SoC. It includes a description of the signals that the integrator must tie off to configure the macrocell for the required integration. Some of the integration is affected by the configuration options you use when you implement the GIC-500.

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

1.8 Product revisions

This section describes the differences in functionality between the product revisions:

r0p0 First release.

Chapter 2
Functional Description

This chapter describes the functionality of the GIC-500. It contains the following sections:

2.1 About the functions

Figure 2-1 shows the top-level functional block diagram with interfaces:

- The interfaces that the GIC-500 provides are described in *Interfaces* on page 2-3.
- The *Interrupt Control Block* (ICB) State RAM holds the settings for SPIs, PPIs, and SGIs. It exists in all configurations of the GIC-500. See *Interrupt types* on page 2-7 for a description of the interrupt types.
- The LPI State RAM and ITE State RAM cache settings for the LPIs and ITS. They only exist in configurations where the ITS exists and LPIs are supported. See *Configurable options* on page 1-9.

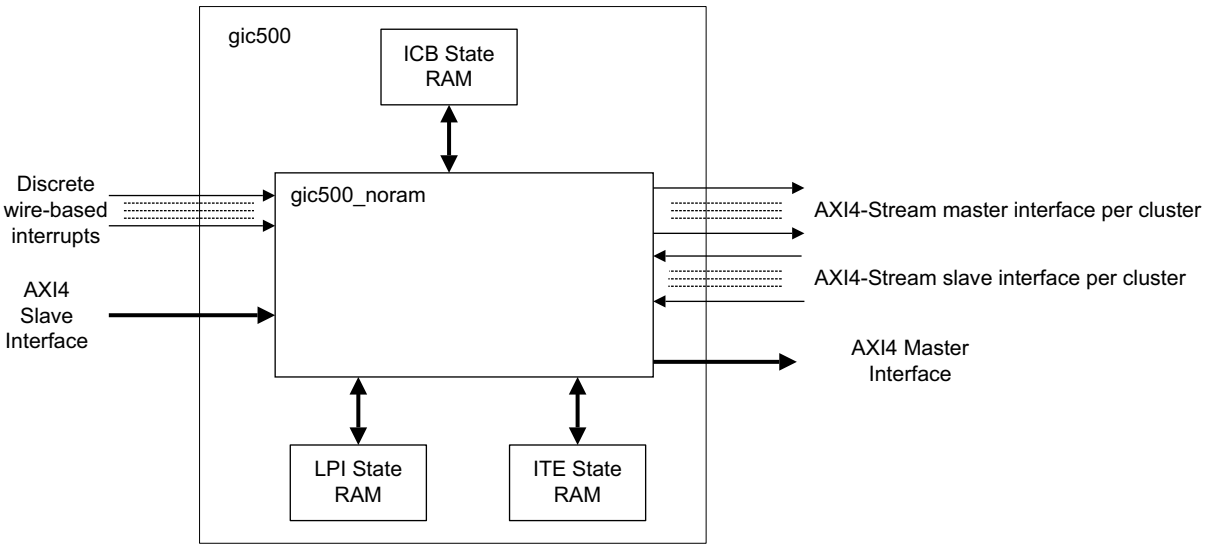

Figure 2-1 Top level functional block diagram

Note

The names of the top-level and noram blocks in a real design contain the names of the chosen configurations. The inputs and outputs are described in *Interfaces* on page 2-3.

2.2 Interfaces

The GIC-500 interfaces are:

- *AXI4 Slave Interface*.
- *AXI4 Master Interface* on page 2-4.
- *RAM Interfaces* on page 2-5.
- *Physical interrupt signals* on page 2-5.
- *GIC-500 Stream Protocol Interface* on page 2-6.
- *Other core signals* on page 2-6.

2.2.1 AXI4 Slave Interface

One 32-bit AMBA AXI4 slave interface provides access to the programming interfaces of all parts of the GIC-500:

- Distributor.
- Redistributors.
- ITS.

The slave interface also handles all message-based interrupts, which are interrupts generated by writes to the AXI4 slave interface. Message-based interrupts can generate SPIs or LPIs, depending on the register that you write to. See *Interrupt types* on page 2-7 for information about the different types of interrupts.

The slave interface uses a single contiguous address region for all registers. See Chapter 3 *Programmers Model* for the address map.

Accesses to some registers requires sideband information. When using backwards compatibility mode, some registers are banked to provide a separate copy for each core. The system integrator must drive **awuser_s[2:0]** and **aruser_s[2:0]** with the binary number of the core performing the access for writes and reads, respectively.

Note

This information is only required for the first eight cores, because only this number of cores is supported in backwards compatibility mode. See *Backwards compatibility* on page 2-10 for more information.

To generate an LPI a peripheral must write to the GITS_TRANSLATER. For the ITS to know which translations to apply to the generated interrupts, it must know which peripheral performed the write. The ID of the peripheral is known as its Device ID. ARM recommends that for a *PCI Express* (PCIe) peripheral, its GIC Device ID is its PCIe Requester ID without any modification. The AXI4 slave only requires the Device ID for writes to the GITS_TRANSLATER. For writes to that register, the GIC-500 supports two ways of receiving the Device ID:

- If **awuser_s[3]** is LOW, then for any writes made to the GITS_TRANSLATER the Device ID bits are taken from **awuser_s[23:4]**.
- If **awuser_s[3]** is HIGH, the slave port uses a mode of operation which reduces the number of AWUSER bits required by the system. In this mode, the GIC-500 assumes all write accesses are to the GITS_TRANSLATER and the Device ID bits are taken from **awaddr_s[21:2]**.

Table 2-1 shows the AXI slave attributes and their values.

Table 2-1 AXI slave interface attributes

Attribute	Value
Combined acceptance capability	6
Read acceptance capability	3
Read data reorder depth	1
Write acceptance capability	3

2.2.2 AXI4 Master Interface

The GIC-500 uses the AXI4 master interface to access main memory. One 64-bit AMBA AXI4 master port is provided to allow the ITS and Redistributors to access main memory. The main memory holds the following:

- The translation tables for the ITS.
- The ITS command queue, which software writes to, that in turn programs the ITS.
- The LPI configuration table, which holds the priorities and enable bits for LPIs.
- The LPI pending tables, which can hold information on whether each LPI is pending on each core.

Note

The AXI master interface is not present if the ITS and LPI support are removed.

The hypervisor or OS software is responsible for allocating memory to the GIC-500. The GIC architecture also requires you to write software that zeros the allocated memory before use. Software must program registers in the ITS and Redistributors with the physical addresses of the allocated memory. Therefore the AXI4 master makes accesses using physical addresses and therefore does not require address translation such as a System *Memory Management Unit* (MMU).

When software has enabled the relevant functionality in the GIC-500, the software must not access the allocated memory again unless allowed by the GIC architecture. For example, memory programmed in the GITS_BASER must not be accessed when the ITS is enabled. However, the GIC architecture always permits software to write to the LPI configuration table pointed to by GICR_PROPBASER, and defines the `INV` and `INVALL` ITS commands that make the GIC use the new property values.

The GIC-500 does not support shareability, but does have programmable cacheability settings. Therefore the GIC-500 always treats memory as non-shareable. Software must discover this by attempting to write to the shareability and cacheability fields that are present in registers such as GICR_PROPBASER and by reading back the written values. The attributes in the MMU translation tables of the core must match those programmed in the GIC.

Consequently, software must issue the appropriate cache maintenance instructions when it wants to ensure that writes made by the core are visible to the GIC and when it wants to ensure that writes made by the GIC are visible to the core.

It is a system integration requirement that accesses that the GIC-500 makes to memory can complete without depending on any other accesses in the system making progress.

The AXI4 master interface only makes certain types of accesses. All transactions are 32 bytes or smaller, consisting of up to four transfers with up to eight bytes in each transfer. Only incrementing bursts are used. Accesses made by the AXI4 master involve the ITS and LPIs, which are always Non-secure. Therefore the AXI4 master always makes Non-secure accesses.

If the AXI4 master receives a bus error, such as SLVERR or DECERR, this is signaled through an external pin, **axim_err**. When this occurs, the GIC-500 might lose interrupts and cannot recover and you must reset the GIC-500. If it is not reset, the behavior becomes UNPREDICTABLE.

Table 2-2 shows the AXI master attributes and their values.

Table 2-2 AXI master interface attributes

Attribute	Value
Combined issuing capability	26
Read issuing capability	11
Write issuing capability	15

2.2.3 RAM Interfaces

The GIC-500 uses SRAMs to cache state, so that it can save area, power, and latency.

In typical operation, their existence is transparent to software, with a few exceptions:

- After reset, the RAMs are automatically initialized. Accesses made to the GIC-500 during this time could take longer than normal.
- The RAMs are protected from errors using an *Error-Correction Code* (ECC) with single error correction and double error detection (SECDED). If a double error is detected then this is signaled through an external pin, **ecc_fatal**. When this occurs, the GIC-500 might lose interrupts and cannot recover and must be reset. If it is not reset, behavior becomes UNPREDICTABLE. The GIC-500 returns corrupted data if a double error is detected during a read to any of the following:
  - GICD_IPRIORITYRn.
  - GICR_IPRIORITYRn.
  - GICD_IROUTERn.
  - GICD_ITARGETSRn.

  If you want to guarantee that software does not read corrupted data after a double error, software must never read from these registers and instead maintain its own copy of the values written to these registers.

2.2.4 Physical interrupt signals

The GIC-500 supports the generation of SPIs and PPIs through physical interrupt signals. These inputs can be programmed as either:

- Level-sensitive, where the interrupt is pending as long as the input is asserted.
- Edge-triggered, where a rising edge causes the interrupt to be set to pending.

There is one independent set of PPIs for each core. SPIs are global and therefore there is only a single set. See *Interrupt types* on page 2-7 for more information.

Note

- In the GIC-500, level-sensitive PPIs are active-LOW, whereas level-sensitive SPIs are active-HIGH.
- The GIC-500 does not synchronize interrupt inputs. They must be synchronized to the GIC-500 clock externally.

2.2.5 GIC-500 Stream Protocol Interface

For each cluster or group of cores, one upstream and one downstream 16-bit AMBA AXI4-Stream interface is provided.

The GIC-500 GIC Stream Protocol Interface consists of a pair of AXI4-Stream interfaces that the GIC-500 uses to send interrupts to the core and receive notifications when the core activates interrupts. There is a pair of physical interfaces, one in each direction, for each cluster.

The GIC Stream interfaces use the prefix **icd** to indicate the master interface, and **icc** to indicate the slave interface. The GIC Stream master interface uses the **icdtdest** signal to direct packets to one core within the cluster. The GIC Stream slave interface uses the **icctid** signal to determine which core within the cluster sent a packet.

2.2.6 Other core signals

When a core is powered down, it must first disable the sending of packets over the AXI4-Stream interface using the GICR_WAKER.ProcessorSleep bit. When this has been done, the presence of an interrupt specifically targeted at that core causes the **wake_request** signal for that core to be asserted. ARM recommends that you connect this signal to the power controller to cause that core to boot. When the core has booted, software is always expected to re-enable communication over the interface through the use GICR_WAKER.ProcessorSleep, allowing the interrupt to be processed.

The GIC-500 uses the **cpu_active** signal to decide which cores are preferred for SPIs that target multiple cores. It does not affect the operation of any other type of interrupt. It also has no effect if GICR_WAKER.ProcessorSleep is set to one for that core, because those SPIs are never sent to cores with ProcessorSleep set to one. ARM recommends that this signal is deasserted when a core is in certain software-transparent sleep states entered during WFI or WFE instructions, such as retention, so that the core is less likely to handle SPIs that target multiple cores. This can increase the amount of time that cores spend in these sleep states. If you use the **cpu_active** signal this way, software must not rely on SPIs that target multiple cores causing cores to leave WFI or WFE. Instead, software must use another mechanism to ensure this, such as an SGI, or an SPI targeted at only the core in question.

2.3 Operation

The GIC-500 is divided into three main sections:

ITS The ITS is responsible for translating message-based interrupts from peripherals into LPIs. You can also use the ITS to manage existing LPIs. The ITS is not used for other types of interrupt.

Distributor The Distributor receives interrupts from the:

- Wire interrupts.
- Programming interface.

It is responsible for prioritizing these interrupts and sending them to the CPU interface using the GIC Stream Protocol Interface.

Redistributor

There is one Redistributor for each core. Each Redistributor holds the state that is individual to a particular core, such as the settings for PPIs, and SGIs. It also stores the LPIs for that core after they have been generated using the ITS.

2.3.1 Interrupt types

This section describes the different types of interrupt that the GIC-500 handles.

SGIs

SGIs are inter-processor interrupts, that is, interrupts generated from one core and sent to other cores. Activating an SGI on one core does not affect the same interrupt ID on another core. Therefore when an SGI is sent to all cores it is handled independently on each core. The settings for each SGI are also independent between cores.

You can generate SGIs using System registers in the generating core, or, in legacy software, by writing to the Software Generated Interrupt Register, GICD_SGIR. There are 16 independent SGIs, ID0-ID15, that are recorded separately for every target core. In backwards compatibility mode, the number of the generating core is also recorded.

PPIs

PPIs are typically used for peripherals that are tightly-coupled to a particular core. Interrupts connected to the PPI inputs associated with one core are only sent to that core. Activating a PPI on one core does not affect the same interrupt ID on another core. The settings for each PPI are also independent between cores.

A PPI is an interrupt that is specific to a single core and is generated by a wire input. PPI signals are active-LOW level-sensitive, by default, but can also be programmed to be triggered on a rising edge.

SPIs

SPIs are typically used for peripherals that are not tightly-coupled to a specific core. You can program each SPI to target either a particular core or any core. Activating an SPI on one core activates the SPI for all cores. That is, the GIC-500 allows at most one core to activate an SPI. The settings for each SPIs are also shared between all cores.

SPIs can be generated either by wire inputs or by writes to the AXI4 slave programming interface. The GIC-500 can support up to 960 SPIs corresponding to the external **spi[991:32]** signal. The number of SPIs available depends on the implemented configuration. The permitted values are 32-960, in steps of 32. The first SPI has an ID number of 32. You can configure whether each SPI is triggered on a rising edge or is active-HIGH level-sensitive.

Note

All signals, including SPIs, must be synchronous to the clock. Therefore, you must synchronize any interrupt signals from an asynchronous source before they are connected to the GIC-500.

LPIs

LPIs are typically used for peripherals that produce message-based interrupts. An LPI targets only one core at a given time. LPIs are generated when the peripheral writes to the ITS, which also holds the registers to control the generation and maintenance of LPIs. The ITS provides interrupt ID translation, allowing peripherals to be owned directly by a virtual machine if a System MMU is also present for those peripherals.

Note

The interrupt ID translation in the ITS only allows interrupts from these peripherals to be translated to the ID space of the hypervisor, not sent directly to the virtual machine.

In the GIC-500, you can only generate LPIs by writing to the ITS using the AXI4 slave programming interface. The GIC-500 always supports up to 57344 LPIs, but has a cache that holds the settings for the most frequent interrupts. The settings for the ITS and LPIs are stored in main memory, so a cache miss might result in up to three round trips to memory. The first LPI has an ID number of 8192.

You can debug problems with LPI interrupt generation in the ITS using the LPI tracking registers. These allow you to set a trigger to capture information about the next interrupt that the ITS processes. This allows you to determine what happened to that interrupt and what translation was applied. See *Implementation defined test registers in the GITS control page summary* on page 3-28 for more information.

For the most efficient caching of LPIs, ARM recommends that you allocate input interrupt IDs sequentially. You can measure the performance of the cache the input interrupt ID affects using the GITS_TRKICR register. See *Debug ITE Cache Statistics* on page 3-32 for more information. The other LPI cache is only affected by the distribution of LPI physical IDs (PIDs). Therefore, if possible, ARM recommends that for best caching, PIDs are also sequentially allocated. The performance of that cache can be monitored using the GITS_TRKLCR register. See *Debug LPI Cache Statistics* on page 3-33 for more information.

Choosing between LPIs and SPIs

You can use both LPIs and SPIs for message-based interrupts. Your decision about whether to use an LPI or SPI for an interrupt can potentially be made by software, providing there are spare SPIs and the GIC-500 is integrated with ITS support. This can be achieved by either making the peripheral write to a different GIC-500 address, or by changing the address translation for the interrupt write in the System MMU. Changing only the System MMU is possible because the

two registers for Non-secure message-based interrupts, GICD_SETSPI_NSR and GITS_TRANSLATER, are at the same address offset in different pages. The following factors can help you to decide which type of interrupts is most appropriate:

- Only the ITS provides interrupt ID translation and therefore LPIs are preferable for peripherals owned by a virtual machine. This is because the hypervisor can let the virtual machine program the peripheral directly, and let the ITS convert its interrupts from the IDs used by the virtual machine to unique physical IDs.
- LPIs are always Group 1 Non-secure, so message-based interrupts that target Secure software must use SPIs.
- Only SPIs provide the ability to target all cores, which means the GIC-500 attempts to automatically balance the interrupt load to cores that are active but not handling other interrupts.
- The GIC-500 can provide a far larger number of LPIs than SPIs for the same area. The marginal area increase for each additional LPI is much less than for an SPI because the LPI settings are only cached, not stored, in the GIC-500. This means that the lowest area can often be achieved by using LPIs where possible rather than SPIs.
- In a small system where the features of the ITS are not required and there are few message-based interrupts, you might decide not to include the ITS and LPI support.
- SPIs usually have a better worst-case interrupt latency than LPIs. This is because SPIs have all their settings stored internally to the GIC-500, whereas LPIs that are not cached require external memory accesses. The cache hit rate is expected to be higher for the LPIs that occur more frequently. Therefore ARM recommends that SPIs are used for any latency sensitive interrupts that are expected to occur infrequently.

2.3.2 Interrupt groups

The GIC-500 implements the following Interrupt Group Registers:

- GICD_IGROUPRn.
- GICD_IGRPMODRn.
- GICR_IGROUPR0.
- GICR_IGRPMODR0.

These control whether each interrupt is configured as:

- Group 0.
- Group 1 Secure.
- Group 1 Non-secure.

Each interrupt is programmed to belong to an interrupt group. Each interrupt group:

- Determines the target exception level and security state for interrupts in that group.
- Has separate enable bits that control whether interrupts in that group can be forwarded to the core.
- Has an impact on later routing decisions in the CPU interfaces.

When using backwards compatibility mode or with security disabled, the meaning and number of interrupt groups are affected. See the *ARM® Generic Interrupt Controller Architecture Specification version 3.0* for more information.

2.3.3 Interrupt triggering

The GIC-500 supports two types of physical interrupt signal:

Level-sensitive

The interrupt is pending while the interrupt input is asserted.

Edge-triggered

A rising-edge on the interrupt input causes the interrupt to become pending. The pending bit is later cleared when the interrupt is activated by the CPU interface.

You must program GICD_ICFGRn and, for PPIs using ARE = 1, GICR_ICFGR1, to have the correct settings for the system.

2.3.4 Backwards compatibility

You can configure the Distributor part of the GIC-500 at build time to support limited backwards compatibility with GICv2. If this support is configured, the Distributor resets to backwards compatibility mode. This mode supports up to eight cores. The eight cores supported by the GIC-500 are the cores with the lowest affinity numbers.

The ARE setting in GICD_CTLR, which disables backwards compatibility, is programmable when backwards compatibility support is configured. The ARE setting is also banked by security. This allows Secure software to operate in backwards compatibility mode (ARE_S = 0) while Non-secure software enables ARE. If Secure software is not operating in backwards compatibility mode then Non-secure software cannot operate in backwards compatibility mode.

Note

A major limitation of operating with Non-secure using ARE_NS = 1 and Secure using ARE_S = 0 is that legacy Secure software is not able to control Non-secure interrupts. This means that existing Secure Monitor code that relies on having this control cannot run correctly using this mode of backwards compatibility. See the *ARM® Generic Interrupt Controller Architecture Specification version 3.0* for more information on this limitation.

The GIC-500 can expose software to more race conditions than previous GIC implementations. Legacy code that relies on the stronger guarantees provided in these previous GICs might not work reliably. For example, some GIC implementations ensure that a sequence of accesses to both the CPU interface and Distributor is executed in program order. However, the ordering between the Distributor and CPU interface is not architecturally guaranteed and is not guaranteed in systems with the GIC-500.

In backwards compatibility mode, the GIC-500 ensures that the effects of all interrupt programming are eventually observable, so you only have to update software that relies on the reprogramming being observable by a specific point. Software typically only requires these ordering guarantees during operations such as retargeting SPIs, if it must be ensured that the previous target does not receive the SPI, or saving the state of a Distributor.

If your software relies on these guarantees, you must update your software using the following guidance:

- To ensure that accesses to the CPU interface have completed and any side-effects have been observed by the Distributor, software must execute a DSB instruction. Use this to ensure the status of interrupts reported by the Distributor is up-to-date. Software that is not using backwards compatibility mode must also use the same approach.

- To know that updates to the Distributor have completed and have been observed by the CPU interface, you must wait until the GIC-500-specific GICD_ESTATUSR.SRWP bit reads as zero. Use this to ensure that after reprogramming an interrupt, cores cannot receive interrupts based on the old interrupt programming. Software not using backwards compatibility mode must instead disable interrupts and poll the GICD_CTLR.RWP bit to ensure that old settings are no longer visible to cores. The advantage of the SRWP bit is that it provides this guarantee even if this sequence is not followed.

The GIC-500 might also implement different IMPLEMENTATION DEFINED choices than previous implementations. For example, the GIC-400 did not set an interrupt pending if the interrupt group for that interrupt was disabled in the GICD_CTLR. In contrast, the GIC-500 does set interrupts pending in this scenario.

It is expected that legacy power management code is not compatible and has to be updated for operation with the GIC-500. For example, GICR_WAKER.ProcessorSleep must be set to zero after reset to enable interrupts to be sent to a core. Likewise, the powerdown sequence requires writing to GICR_WAKER. The GICR_WAKER register is new in version three of the GIC architecture and therefore you must update legacy code to use it.

2.3.5 Disable Security

The *Disable Security* (DS) bit removes the security support of the Distributor. It can be set by Secure software during the boot sequence or be configured to be always set when you configure the design. This configuration option must be used when the system does not have the concept of security to allow access to important registers. If you run software without security awareness on a system that supports security, then the Secure boot code can set DS before switching to a Non-secure exception level to run the software. This enables you to program the GIC-500 from any exception level and use two interrupt groups, Group 0 and Group 1. This means that interrupts can target both the FIQ and IRQ handlers on a core.

You must take care when deciding to write security-unaware software using Group 0, as it might not be portable to systems with a concept of security. This is because Group 0 is always Secure in systems with security. It is most portable for security-unaware software to always use Group 1.

If a system has a concept of security but one or more cores do not, then you must not set DS. Instead each core is only able to enable the interrupt groups corresponding to the security states that it supports.

2.3.6 Power management

The GIC-500 supports the power down of cores and can itself be powered down. The GICR_WAKER registers provide bits to control functions associated with power management. The architecture recommends how these bits can be used. Some bits within the GICR_WAKER are global, rather than separate, for each Redistributor, as GIC-500 is a monolithic implementation. This means that the components are not distinct, but are instead tightly integrated. The GIC architecture allows such implementations to behave differently. See the *ARM® Generic Interrupt Controller Architecture Specification version 3.0* for more information.

The GIC architecture defines the programming sequence to safely power down a core that is attached to a Distributor. This involves using the GICR_WAKER.ProcessorSleep bit. When all cores within a cluster have been powered down using the architectural sequence, you can power gate the AXI4-Stream interface for that cluster.

When powering down the GIC-500, with the exception of the LPI pending bits, software must preserve the state of the GIC-500. The state must be copied after the GIC-500 core power down sequence has completed to ensure that the pending information that is preserved is up to date.

You can preserve the LPI pending bits using the architectural Redistributor powerdown sequence, which ensures the memory pointed to by each GICR_PENDBASER contains the updated pending information for the LPIs. This involves using the GICR_WAKER.Sleep bit. When the GIC-500 is powered up again, you can program the GICR_PENDBASER registers to point to the same memory to reload the LPI pending status. If there is no requirement to reload the pending LPIs, ARM recommends that you zero the pending table and set the GICR_PENDBASER.PTZ bit to one to speed up the initialization of the GIC-500.

Note

- GICR_WAKER.Sleep can only be set to one when:
  - All RD0s have GICR_WAKER.ProcessorSleep == 1.
  - All RD0s have GICR_WAKER.ChildrenAsleep == 1.
- GICR_WAKER.ProcessorSleep can only be set to zero when:
  - GICR_WAKER.Sleep == 0.
  - GICR_WAKER.Quiescent == 0.

Before a core is powered down, you must set GICR_WAKER.ProcessorSleep to one and wait until GICR_WAKER.ChildrenAsleep is one to ensure there are no outstanding transactions on the AXI4-Stream interface of the core. In the typical powerdown sequence, to ensure that there are no interrupts during the powerdown of the core, you must:

1. Mask interrupts on the core.
2. Clear the CPU interface enables.
3. Set the interrupt bypass disable on the CPU interface.

When a core has been powered down and the GICR_WAKER.ProcessorSleep bit is set to one, the GIC-500 attempts to wake the core if it receives an interrupt that targets only that core. It does this by asserting the **wake_request** signal corresponding to that core. This signal connects to the power controller. See *Other core signals* on page 2-6 for more information about the **wake_request** signals.

You must not set GICR_WAKER.ProcessorSleep to one unless the core is entering a power state where the GIC-500 must use the power controller to wake the core rather than using the AXI4-Stream interface. For example, with Cortex-A53 and Cortex-A57, if the core is entering a sleep state based on the WFI or WFE instructions, such as retention, you must not set GICR_WAKER.ProcessorSleep to one. The core can enter these sleep states without software assistance. Given GICR_WAKER.ProcessorSleep is zero, the GIC-500 sends interrupts using the AXI4-Stream interface as normal. These interrupts can cause the core to leave the WFI or WFE instruction based on the standard rules in the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architectural profile*. The system integrator can use the **cpu_active** signal to ensure that interrupts that can target multiple cores are much less likely to target cores in certain sleep states. In such a system, the software has more control over when cores leave sleep states.

Note

Interrupts that target only one core are unaffected by **cpu_active** and are always sent to that core. Moreover, if the GICR_WAKER.ProcessorSleep for that core is set, the **wake_request** signal is asserted for that core. See *Other core signals* on page 2-6 for more information about the **cpu_active** signals.

See the *ARM® Generic Interrupt Controller Architecture Specification version 3.0* for information about power management, and about wakeup signals and their relation to the core outputs.

2.4 Clocking and resets

All configurations of the GIC-500 use a single clock input, **clk**, and a single reset input, **resetn**. See *Clock and reset signals* on page A-2.

Note

Clock and reset signals apply to all interfaces on the GIC-500 and all interfaces must be synchronous to this clock. Therefore, synchronizer cells might be required for certain inputs.

2.5 Constraints and limitations

ARM recommends that when ARE = 1 you avoid generating or disabling SPI interrupts that target multiple cores at times when no core is able to handle them. That is, disable or avoid generating such interrupts while all targeted CPU interfaces are either powered down or do not have the interrupt group enabled. This is because interrupts that are generated in these circumstances might cause higher power consumption inside GIC-500. The GIC architecture specifies that such interrupts do not cause a **wake_request** to become asserted so, in the absence of other stimulus, this state of higher power consumption persists.

Chapter 3
Programmers Model

This chapter describes the GIC-500 registers and provides information about programming the device. It contains the following sections:

3.1 About the GIC-500 programmers model

The following sections describe the GIC-500 registers:

- *Distributor register summary* on page 3-6.
- *Distributor registers for message-based SPIs summary* on page 3-11.
- *Redistributor registers for control and physical LPIs summary* on page 3-12.
- *Redistributor registers for SGIs and PPIs summary* on page 3-15.
- *ITS control register summary* on page 3-17.
- *ITS translation register summary* on page 3-21.
- *Implementation defined test registers in GICD page summary* on page 3-22.
- *Implementation defined test registers in the GICR page for PPIs and SGIs* on page 3-25.
- *Implementation defined test registers in the GITS control page summary* on page 3-28.

The following information applies to the GIC-500 registers:

- The GIC-500 implements only memory-mapped registers.
- The GIC-500 has a single base address. This address is not fixed and can be different for each particular system implementation.

  The offset of each register from the base address is fixed.
- Accesses to reserved or unused address locations do not result in a bus error. Reads to these locations return zero and writes are ignored.
- Unless otherwise stated in the accompanying text:
  - Do not modify reserved register bits.
  - Ignore reserved register bits on reads.
  - A system reset or a powerup reset resets all register bits to 0.
- The width of the GIC-500 AXI4 slave port is 32 bits. The *ARM® Generic Interrupt Controller Architecture Specification version 3.0* defines the permitted sizes of access.

  When byte access is permitted, halfword access is also permitted. When word access is permitted, doubleword access is also permitted.

  Byte or halfword accesses to registers that do not permit those access sizes are not successful and return a SLVERR response.

  Note

  The GIC-500 does not guarantee single-copy atomicity for doubleword accesses, although this is guaranteed for each word in the doubleword. However, none of the GIC-500 registers require atomic doubleword accesses for correct operation.
- The GIC-500 only supports data in little-endian format.
- The access types for the GIC-500 are as follows:

  RO Read only.

  RW Read and write.

  WO Write only. Reads return an UNKNOWN value.

3.2 The GIC-500 register map

All of the GIC-500 registers have names that provide a short mnemonic for the function of the register. In these names:

- The first letters indicate the logical block that the register belongs to:
  - GICD indicates a Distributor register.
  - GICR indicates a Redistributor register.
  - GITS indicates an Interrupt Translation Service register.
- The remaining letters are a mnemonic for the register, for example the GIC Distributor Control Register is called GICD_CTLR.

All pages are 64KB in size to allow for best compatibility with ARMv8. The address map within all pages is defined by the architecture specification. See the *ARM® Generic Interrupt Controller Architecture Specification version 3.0*.

The number of bits of address used by the GIC-500 address map is:

$18 + \max(1, \text{ceil}(\log_2(\text{total\_number\_of\_cpus})))$

Note

The write address bus might be larger than the result of this equation in certain cases. In these cases the *most significant bit* (MSB) referred to here might not be the MSB of the write address bus. See *Effect of Device ID multiplexing* on page 3-4 for more information.

The top bit of the address selects between two sets of pages. When the MSB is LOW, the set of pages is as shown in Table 3-1:

Table 3-1 Lower half of address map

Address[MSB]	Address[MSB-1:16]	Page description	Address range
0	0	Distributor registers (GICD_*) See *Distributor register summary* on page 3-6.	0x00000-0x0FFFF
	1	Distributor registers for message-based SPIs (GICD_*) See *Distributor registers for message-based SPIs summary* on page 3-11.	0x10000-0x1FFFF
	2	Interrupt Translation Service control registers (GITS_*) See *ITS control register summary* on page 3-17.	0x20000-0x2FFFF
	3	Interrupt Translation Service register (GITS_TRANSLATER) See *ITS translation register summary* on page 3-21.	0x30000-0x3FFFF
	Otherwise	Reserved	

When the MSB is HIGH as shown in Table 3-2, the set of pages contains the Redistributor pages in sequence. For example, the RDs that correspond to each core in the first cluster precede the RDs that correspond to each core in the second cluster.

Table 3-2 Upper half of address map

Address[MSB]	Address[MSB-1:17]	Address[16]	Page description
1	Global core number. This is a core number using the contiguous, linear representation as depicted in Figure 1-2 on page 1-4.	0	Redistributor registers for control and physical LPIs (GICR_*). See *Redistributor registers for control and physical LPIs summary* on page 3-12
		1	Redistributor registers for SGIs and PPIs (GICR_*). See *Redistributor registers for SGIs and PPIs summary* on page 3-15.
	Values that do not correspond to a core number.	x	Reserved.

For example, if there are four clusters, each with four cores, then there are `18 + max(1, 4) = 22` address bits, and the RD control registers for cluster 1, core 0 are at an offset `0x280000-0x28FFFF`, with those for SGIs and PPIs at an offset of `0x29000-0x29FFFF`.

3.2.1 Discovery

ARM recommends that the system provides the operating system with three pointers to the start of the first Distributor, ITS and Redistributor pages, respectively. These three pages can be checked against the discovery registers that start at offset `0xFFD0` for all of these pages to verify they are pages of GIC registers. These registers allow discovery of whether the architecture version and, for GIC-500, whether the page contains the Distributor, ITS, or Redistributor registers. When this is known, additional information can be obtained from registers specific to each page.

For Redistributors, ARM recommends that GICR_TYPER is examined to determine:

- Whether the implementation has two or four pages per Redistributor, based on the features implemented. It can be inferred that GIC-500 has only two pages for each Redistributor because the feature bits in that register indicate that it does not support virtual LPIs.
- Whether it is the last Redistributor in the series of pages.
- Which core it is the Redistributor for, based on affinity values.

This information allows you to iteratively search through all Redistributors, discovering all of them in a generic manner.

The GITS_TYPER register in the GIC-500 indicates that you must program the ITS with linear processor numbers, rather than physical target addresses. The GICR_TYPER contains the linear processor number that you must use to reference a Redistributor when programming the ITS.

Legacy code can still discover the identifier that the GIC-500 uses to reference the current core by reading from the GICD_ITARGETSR0.

3.2.2 Effect of Device ID multiplexing

GIC-500 supports transporting the Device ID over **awaddr_s** for GITS_TRANSLATER writes, which reduces the size of **awuser_s** in the system. See *AXI4 Slave Interface* on page 2-3. This is invisible to software and is a system integration decision.

To support this, the address space required in the system might be larger than the normal address map, with the normal address map occupying the lower part of the address space. Given the maximum 20 bits of Device ID, this can only be the case for some systems with fewer than nine cores.

If transporting the Device ID over the address bus, the total address space required is (in bits):

```
max(18 + max(1, ceil(log2(total_number_of_cpus))), device_id_width + 2)
```

3.2.3 GIC-500 register access and banking

For information on the register access and banking scheme, see the *ARM® Generic Interrupt Controller Architecture Specification version 3.0* The key characteristics of the scheme are:

- Some registers, such as the *Distributor Control Register*, GICD_CTLR, and the *Redistributor Control Register*, GICR_CTLR, are banked by security. This provides separate Secure and Non-secure copies of the registers. A Secure access to the address accesses the Secure copy of the register, and a Non-secure access accesses the Non-secure copy.
- In backwards compatibility mode, where ARE = 0 for the security state of the access, when the GIC-500 is part of a multiprocessor system, registers associated with PPIs or SGIs are banked to provide a separate copy for each connected core.
- Some registers, such as the *Interrupt Group Registers*, GICD_IGROUPRn, are only accessible using Secure accesses.
- Non-secure accesses to registers or parts of a register that are only accessible to Secure accesses are *Read-As-Zero* and *Writes Ignored* (RAZ/WI) for that part.

See the *ARM® Generic Interrupt Controller Architecture Specification version 3.0.*

3.3 Distributor register summary

Address offsets are relative to the *Distributor base address* defined by the system memory map. Unless otherwise stated in the register description, all GIC-500 registers are 32 bits wide.

Table 3-3 lists the Distributor registers in base offset order and provides a reference to the register description that either this book or the *ARM® Generic Interrupt Controller Architecture Specification version 3.0* describes.

Offsets that are not shown are Reserved and RAZ/WI.

Table 3-3 Distributor register summary

Offset	Name[a]	Type	Reset	Description[b]
0x0000	GICD_CTLR	RW	Configuration dependent[c]	Distributor Control Register
0x0004	GICD_TYPER	RO	Configuration dependent[d]	Interrupt Controller Type Register
0x0008	GICD_IIDR	RO	0x0000043B	*Distributor Implementer Identification Register* on page 3-9
0x000C-0x003C	-	-	-	Reserved
0x0040	GICD_SETSPI_NSR	WO	-	Non-secure SPI Set Register
0x0044	-	-	-	Reserved
0x0048	GICD_CLRSPI_NSR	WO	-	Non-secure SPI Clear Register
0x004C	-	-	-	Reserved
0x0050	GICD_SETSPI_SR[e]	WO	-	Secure SPI Set Register[f]
0x0054	-	-	-	Reserved
0x0058	GICD_CLRSPI_SR[e]	WO	-	Secure SPI Clear Register[f]
0x005C-0x007C	-	-	-	Reserved
0x0080-0x00F8	GICD_IGROUPRn	RW	0x00000000	Interrupt Group Registers[f]
0x0100	GICD_ISENABLERn	RW[gh]	SGIs and PPIs: 0x0000FFFF[ijkl]	Interrupt Set-Enable Registers
0x0104-0x0178			SPIs: 0x00000000	
0x0180	GICD_ICENABLERn	RW[gm]	SGIs and PPIs: 0x0000FFFF[njkl]	Interrupt Clear-Enable Registers
0x0184-0x01F8			SPIs: 0x00000000	
0x0200-0x0278	GICD_ISPENDRn[o]	RW	0x00000000	Interrupt Set-Pending Registers
0x0280-0x02F8	GICD_ICPENDRn[o]	RW	0x00000000	Interrupt Clear-Pending Registers
0x0300-0x0378	GICD_ISACTIVERn[o]	RW	0x00000000	Interrupt Set-Active Registers
0x0380-0x03F8	GICD_ICACTIVERn[o]	RW	0x00000000	Interrupt Clear-Active Registers
0x0400-0x07DC	GICD_IPRIORITYRn[p]	RW	0x00000000	Interrupt Priority Registers.
0x0800-0x081C	GICD_ITARGETSRn[p]	RO[q]	-	Interrupt Targets Registers[r]
0x0820-0x0BDC		RW	0x00000000	

Table 3-3 Distributor register summary (continued)

Offset	Name[a]	Type	Reset	Description[b]
0x0C00	GICD_ICFGRn	RO	SGIs: 0xAAAAAAAA[l]	Interrupt Configuration Registers
0x0C04		RW	PPIs: 0x00000000[l]	
0x0C08-0x0CF4		RW[s]	SPIs: 0x00000000	
0x0D00-0x0D78	GICD_IGRPMODRn[e]	RW	0x00000000	Interrupt Group Modifier Registers
0x0E00-0x0EF4	GICD_NSACRn[e]	RW	0x00000000	Non-secure Access Control Registers
0x0F00	GICD_SGIR[t]	WO	-	Software Generated Interrupt Register
0x0F10-0x0F1C	GICD_CPENDSGIRn[t]	RW	0x00000000	SGI Clear-Pending Registers
0x0F20-0x0F2C	GICD_SPENDSGIRn[e]	RW	0x00000000	SGI Set-Pending Registers
0x0F30-0x60FC	-	-	-	Reserved
0x6100-0x7EF8	GICD_IROUTERn	RW	0x0000000000000000	Interrupt Routing Registers, 64-bit
0x7F00-0xBFFC	-	-	-	Reserved
0xC000	GICD_ESTATUSR	RO	0x00000000	*Extended Status Register* on page 3-22
0xC004	GICD_ERRTESTR	WO	-	*Error Test Register* on page 3-23
0xC008-0xC080	-	-	-	Reserved
0xC084-0xC0F8	GICD_SPISRn	RO	-	GIC-500 Shared Peripheral Interrupt Status Registers
0xC100- 0xFFCC	-	-	-	Reserved
0xFFD0	GICD_PIDR4	RO	0x00000044	Peripheral ID 4 Register
0xFFD4	GICD_PIDR5	RO	0x00000000	Peripheral ID 5 Register
0xFFD8	GICD_PIDR6	RO	0x00000000	Peripheral ID 6 Register
0xFFDC	GICD_PIDR7	RO	0x00000000	Peripheral ID 7 Register
0xFFE0	GICD_PIDR0	RO	0x00000092[u]	Peripheral ID 0 Register
0xFFE4	GICD_PIDR1	RO	0x000000B4	Peripheral ID 1 Register
0xFFE8	GICD_PIDR2	RO	0x0000003B	*Peripheral ID2 Register* on page 3-9
0xFFEC	GICD_PIDR3	RO	0x00000000	Peripheral ID 3 Register
0xFFF0	GICD_CIDR0	RO	0x0000000D	Component ID 0 Register
0xFFF4	GICD_CIDR1	RO	0x000000F0	Component ID 1 Register
0xFFF8	GICD_CIDR2	RO	0x00000005	Component ID 2 Register
0xFFFC	GICD_CIDR3	RO	0x000000B1	Component ID 3 Register

a. n denotes that there are multiple registers.

b. For the description of the registers that are not specific to the GIC-500, see the *ARM® Generic Interrupt Controller Architecture Specification version 3.0*.

c. The reset value depends on the configuration of the GIC-500.
   In systems that include Security Support, the values of the configuration dependent fields of GICD_CTLR are:
   DS, bit[6] - 0.
   ARE_NS, bit[5] - no GICv2 backwards compatibility support included.
   ARE_S, bit[4] - no GICv2 backwards compatibility support included.
   In systems that do not include Security Support, the values of the configuration dependent fields of GICD_CTLR are:
   DS, bit[6] - 1.
   ARE, bit[4] - no GICv2 backwards compatibility support included.
d. The reset value depends on the configuration of the GIC-500. The values of the configuration dependent fields of the GICD_TYPER are:
   A3V, bit[24] - 0.
   IDbits, bits[23:19] - `0b01111`.
   DVIS, bit[18] - 0.
   LPIS, bit[17] - ITS and LPI support included.
   MBIS, bit[16] - 1.
   LSPI, bits[15:11] - `0b00000`.
   SecurityExtn, bit[10] - Security Support included.
   CPUNumber, bits[7:5] - the number of cores in the system, saturated to eight, minus one.
   ITLinesNumber, bits[4:0] - the Number of SPIs divided by 32.
e. The existence of this register depends on the configuration of the GIC-500. If Security Support is not included, this register does not exist.
f. This register is only accessible from a Secure access.
g. Writes to bits corresponding to the SGIs are ignored.
h. GICD_ISENABLER0 is a mixed type register.
i. GICD_ISENABLER0 SGI bits are RO, PPI bits are RW.
j. The reset value for the register that contains the SGI and PPI interrupts is `0x0000FFFF` because SGIs are always enabled. However, SGIs are Group 0 on reset, so the reset value for Non-secure reads is `0x00000000`.
k. In configurations where ARE is not programmable, the reset value for SGIs is zero.
l. The existence of this register depends on the configuration of the GIC-500. If GICv2 backwards compatibility support is not included, this register does not exist.
m. GICD_ICENABLER0 is a mixed type register.
n. GICD_ICENABLER0 SGI bits are RO, PPI bits are RW.
o. The existence of the first of these registers depends on the configuration of the GIC-500. If GICv2 backwards compatibility support is not included, the first register does not exist.
p. The existence of the first eight of these registers depends on the configuration of the GIC-500. If GICv2 backwards compatibility support is not included, the first eight registers do not exist.
q. The registers that contain the SGI and PPI interrupts are read-only and the value is the core number of the current access. It is encoded in an 8-bit one-hot field, for each implemented interrupt, and encoded as `zero` for interrupts that are not implemented. For more information on core target field bit values, see the *ARM® Generic Interrupt Controller Architecture Specification* version 3.0.
r. In a system with a single core, these registers are RAZ/WI. For more information, see the *ARM® Generic Interrupt Controller Architecture Specification version 3.0.*
s. The even bits of this register are RO. For more information, see the *ARM® Generic Interrupt Controller Architecture Specification* version 3.0.
t. The existence of this register depends on the configuration of the GIC-500. If GICv2 backwards compatibility support is not included, this register does not exist.
u. The value of PIDR0 differs between the GICD, GICR, and GITS versions, and so you can tell the difference between these pages.

3.4 Distributor register descriptions

This section describes the Distributor registers whose implementation is specific to the GIC-500. The *ARM® Generic Interrupt Controller Architecture Specification* version 3.0 describes all the other registers.

3.4.1 Distributor Implementer Identification Register

The GICD_IIDR characteristics are:

Purpose Provides information about the implementer and revision of the Distributor.

Usage constraints There are no usage constraints.

Configurations Always present in the GIC-500.

Attributes See the register summary in Table 3-3 on page 3-6.

Figure 3-1 shows the bit assignments.

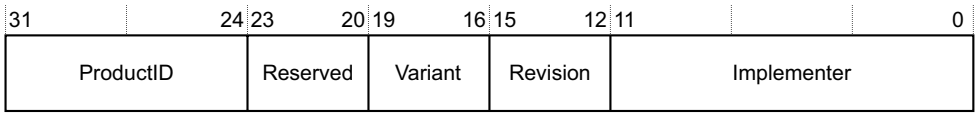

Figure 3-1 GICD_IIDR bit assignments

Table 3-4 shows the bit assignments.

Table 3-4 GICD_IIDR bit assignments

Bits	Name	Description
[31:24]	ProductID	Indicates the product ID: `0x00` GIC-500
[23:20]	-	Reserved, RAZ
[19:16]	Variant	Indicates the major revision or variant of the product: `0x0` Variant number
[15:12]	Revision	Indicates the minor revision of the product: `0x0` Revision number
[11:0]	Implementer	Indicates the implementer: `0x43B` ARM

3.4.2 Peripheral ID2 Register

The GICD_PIDR2 characteristics are:

Purpose Defines the GIC architecture version with which the GIC-500 complies.

Usage constraints There are no usage constraints.

Configurations Always present in the GIC-500.

Attributes See the register summary in Table 3-5 on page 3-10.

Figure 3-2 on page 3-10 shows the GICD_PIDR2 bit assignments.

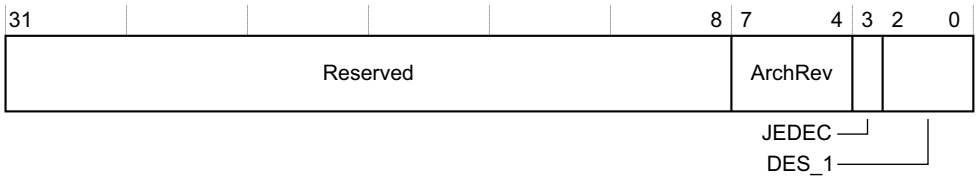

Figure 3-2 GICD_PIDR2 bit assignments

Table 3-5 shows the GICD_PIDR2 bit assignments.

Table 3-5 GICD_PIDR2 bit assignments

Bits	Name	Function
[31:8]	-	Reserved, RAZ.
[7:4]	ArchRev	Identifies the version of the GIC architecture with which the GIC-500 complies: `0x3` version 3.0.
[3]	JEDEC	Indicates that a JEDEC-assigned JEP106 identity code is used.
[2:0]	DES_1	JEP106 identity code [6:4].

3.5 Distributor registers for message-based SPIs summary

Table 3-6 lists the message-based SPI registers.

Table 3-6 Distributor registers for message-based SPI register summary

Offset	Name	Type	Reset	Description[a]
0x0000-0x003C	-	-	-	Reserved
0x0040	GICD_SETSPI_NSR	WO	-	Aliased Non-secure SPI Set Register
0x0044	-	-	-	Reserved
0x0048	GICD_CLRSPI_NSR	WO	-	Aliased Non-secure SPI Clear Register
0x004C	-	-	-	Reserved
0x0050	GICD_SETSPI_SR[b]	WO	-	Aliased Secure SPI Set Register[c]
0x0054	-	-	-	Reserved
0x0058	GICD_CLRSPI_SR[b]	WO	-	Aliased Secure SPI Clear Register[c]
0x005C-0xFFFC	-	-	-	Reserved

a. For the description of the registers that are not specific to the GIC-500, see the *ARM® Generic Interrupt Controller Architecture Specification* version 3.0.

b. The existence of this register depends on the configuration of the GIC-500. If Security Support is not included, this register does not exist.

c. This register is only accessible from a Secure access.

3.6 Redistributor registers for control and physical LPIs summary

In GICv3 and GICv4, these registers start from RD_base and the offset of each register is defined in Table 3-7.

Offsets that are not shown are Reserved and RAZ/WI.

Table 3-7 Redistributor register summary

Offset	Name[a]	Type	Reset	Description[b]
0x0000	GICR_CTLR	RW	0x00000000	Redistributor Control Register.
0x0004	GICR_IIDR	RO	0x0000043B	Redistributor ID Register.
0x0008-0x000C	GICR_TYPER	RO	Configuration dependent[c]	Redistributor Type Register, 64-bit.
0x0010	-	-	-	Reserved.
0x0014	GICR_WAKER	RW	0x00000006	Power Management Control Register.
0x0018-0x006C	-	-	-	Reserved.
0x0070-0x0074	GICR_PROPBASER[d]	RW	0x0000000000000000	Common LPI configuration table base register, 64-bit.
0x0078-0x007C	GICR_PENDBASER[de]	RW	0x0000000000000000	LPI pending table base register, 64-bit.
0x0080-0xFFCC	-	-	-	Reserved.
0xFFD0	GICR_PIDR4	RO	0x00000044	Peripheral ID 4 Register.
0xFFD4	GICR_PIDR5	RO	0x00000000	Peripheral ID 5 Register.
0xFFD8	GICR_PIDR6	RO	0x00000000	Peripheral ID 6 Register.
0xFFDC	GICR_PIDR7	RO	0x00000000	Peripheral ID 7 Register.
0xFFE0	GICR_PIDR0	RO	0x00000093	Peripheral ID 0 Register.
0xFFE4	GICR_PIDR1	RO	0x000000B4	Peripheral ID 1 Register.
0xFFE8	GICR_PIDR2	RO	0x0000003B	*Peripheral ID2 Register* on page 3-14.
0xFFEC	GICR_PIDR3	RO	0x00000000	Peripheral ID 3 Register.
0xFFF0	GICR_CIDR0	RO	0x0000000D	Component ID 0 Register.
0xFFF4	GICR_CIDR1	RO	0x000000F0	Component ID 1 Register.
0xFFF8	GICR_CIDR2	RO	0x00000005	Component ID 2 Register.
0xFFFC	GICR_CIDR3	RO	0x000000B1	Component ID 3 Register.

a. n corresponds to the number of a CPU interface.

b. For the description of the registers that are not specific to the GIC-500, see the *ARM® Generic Interrupt Controller Architecture Specification* version 3.0.

c. The reset value depends on the configuration of the GIC-500. The values of the configuration dependent fields of the GICR_TYPER are:

   A3, bits[63:56] - 0.

   A2, bits[55:48] - 0.

   A1, bits[47:40] - the Affinity Level 1 value for this Redistributor.

   A0, bits[39:32] - the Affinity Level 0 value for this Redistributor.

   Processor Number, bits [23:8] - the linear Processor Number for this Redistributor.

   DPGS, bit[5] - 0

   Last, bit[4] - this bit is only set to one for the last Redistributor in the memory-map of the configured GIC-500.

   Distributed, bit[3] - 0.

   VLPIS, bit[1] - 0.

   PLPIS, bit[0] - ITS and LPI support included.

d. The existence of this register depends on the configuration of the GIC-500. If ITS and LPI support is not included, this register does not exist.

e. ARM recommends that if possible, you set the GICR_PENDBASER Pending Table Zero bit to one. This reduces the power and time taken during initialization.

3.7 Redistributor register descriptions

This section only describes the Redistributor registers whose implementation is specific to the GIC-500. The *ARM® Generic Interrupt Controller Architecture Specification* version 3.0 describes all of the other registers.

3.7.1 Peripheral ID2 Register

The GICR_PIDR2 characteristics are:

Purpose Defines the GIC architecture version with which the GIC-500 complies.

Usage constraints There are no usage constraints.

Configurations Always present in the GIC-500.

Attributes See the register summary in Table 3-7 on page 3-12.

Figure 3-3 shows the bit assignments.

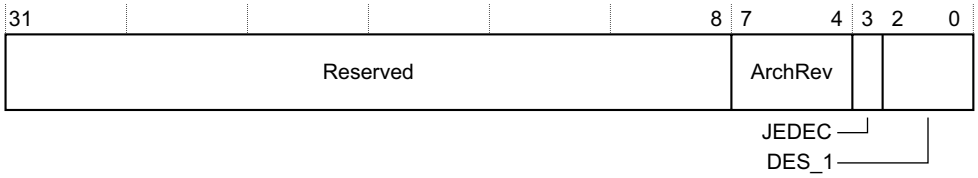

Figure 3-3 GICR_PIDR2 bit assignments

Table 3-8 shows the bit assignments.

Table 3-8 GICR_PIDR2 bit assignments

Bits	Name	Function
[31:8]	-	Reserved, RAZ.
[7:4]	ArchRev	Identifies the version of the GIC architecture with which the GIC-500 complies: `0x3` version 3.0.
[3]	JEDEC	Indicates that a JEDEC-assigned JEP106 identity code is used.
[2:0]	DES_1	JEP106 identity code [6:4].

3.8 Redistributor registers for SGIs and PPIs summary

Table 3-9 shows the address map of the Redistributor registers for SGIs and PPIs Registers.

Offsets that are not shown are Reserved and RAZ/WI.

Note

These registers are all Reserved and RAZ/WI when using backwards compatibility mode.

Table 3-9 Redistributor registers for SGIs and PPIs summary

Offset	Name	Type	Reset	Description
0x0000-0x007C	-	-	-	Reserved
0x0080	GICR_IGROUPR0	RW	0x00000000	Interrupt Group Registers[a]
0x0084-0x0FFC	-	-	-	Reserved
0x0100	GICR_ISENABLER0	RW	0x00000000	Interrupt Set-Enable Registers
0x0104-0x017C	-	-	-	Reserved
0x0180	GICR_ICENABLER0	RW	0x00000000	Interrupt Clear-Enable Registers
0x0184-0x01FC	-	-	-	Reserved
0x0200	CICR_ISPENDR0	RW	0x00000000	Interrupt Set-Pending Registers
0x0204-0x027C	-	-	-	Reserved
0x0280	GICR_ICPENDR0	RW	0x00000000	Interrupt Clear-Pending Registers
0x0284-0x02FC	-	-	-	Reserved
0x0300	GICR_ISACTIVER0	RW	0x00000000	Interrupt Set-Active Registers
0x0304-0x037C	-	-	-	Reserved
0x0380	GICR_ICACTIVER0	RW	0x00000000	Interrupt Clear-Active Registers
0x0384-0x03FC	-	-	-	Reserved
0x0400-0x041C	GICR_IPRIORITYRn	RW	0x00000000	Interrupt Priority Registers
0x0420-0x0BFC	-	-	-	Reserved
0x0C00	GICR_ICFGRn	RO	SGIs: 0xAAAAAAAA[b]	Interrupt Configuration Registers
0x0C04		RW	PPIs: 0x00000000	
0x0C08-0x0CFC	-	-	-	Reserved
0x0D00	GICR_IGRPMODR0	RW	0x00000000	Interrupt Group Modifier Registers
0x0D04-0x0DFC	-	-	-	Reserved
0x0E00	GICR_NSACR	RW	0x00000000	Non-secure Access Control Registers
0x0E04-0xBFFC	-	-	-	Reserved

Table 3-9 Redistributor registers for SGIs and PPIs summary (continued)

Offset	Name	Type	Reset	Description
0xC000	GICR_MISCSTATUSR	RO	-	*Miscellaneous Status Register* on page 3-25
0xC080	GICR_PPISR	RO	-	*Private Peripheral Interrupt Status Register* on page 3-26
0xC084-0xFFFC	-	-	-	Reserved

a. This register is only accessible from a Secure access.

b. When GIC-500 is configured to have ARE as programmable, this register is Reserved and RAZ/WI if ARE is 0 for the security state of the SGI.

3.9 ITS control register summary

Table 3-10 shows the address map of the ITS control registers.

Offsets that are not shown are Reserved and RAZ/WI.

Note

This page does not exist in configurations of the GIC-500 without LPI and ITS support.

Table 3-10 ITS control register summary

Offset	Name[a]	Type	Reset	Description[b]
0x0000	GITS_CTLR	RW	0x80000000	ITS Control Register
0x0004	GITS_IIDR	RO	0x0000043B	*ITS Identification Register* on page 3-19
0x0008-0x000C	GITS_TYPER	RO	Configuration dependent[c]	ITS Type Register, 64-bit
0x0010-0x007C	-	-	-	Reserved
0x0080-0x0084	GITS_CBASER	RW	0x0000000000000000	The command queue control register, 64-bit
0x0088-0x008C	GITS_CWRITER	RW	0x0000000000000000	The command queue write pointer, 64-bit
0x0090-0x0094	GITS_CREADR	RO	0x0000000000000000	The command queue read pointer, 64-bit
0x0098-0x00FC	-	-	-	Reserved
0x0100-0x0104	GITS_BASER0	RW	0x0107000000000000	ITS table control register, 64-bit
0x0108-0xBFFC	-	-	-	Reserved
0xC000	GITS_TRKCTLR	WO	-	*Tracking Control Register* on page 3-28
0xC004	GITS_TRKR	RO	0x00000000	*Tracking Status Register* on page 3-29
0xC008	GITS_TRKDIDR	RO	0x00000000	*Debug Tracked DID Register* on page 3-30
0xC00C	GITS_TRKPIDR	RO	0x00000000	*Debug Tracked PID Register* on page 3-31
0xC010	GITS_TRKVIDR	RO	0x00000000	*Debug Tracked ID Register* on page 3-31
0xC014	GITS_TRKTGTR	RO	0x00000000	*Debug Tracked Target Register* on page 3-32
0xC018	GITS_TRKICR	RO	0x00000000	*Debug ITE Cache Statistics* on page 3-32
0xC01C	GITS_TRKLCR	RO	0x00000000	*Debug LPI Cache Statistics* on page 3-33
0xC020-0xFFCC	-	-	-	Reserved
0xFFD0	GITS_PIDR4	RO	0x00000044	Peripheral ID 4 Register
0xFFD4	GITS_PIDR5	RO	0x00000000	Peripheral ID 5 Register
0xFFD8	GITS_PIDR6	RO	0x00000000	Peripheral ID 6 Register
0xFFDC	GITS_PIDR7	RO	0x00000000	Peripheral ID 7 Register
0xFFE0	GITS_PIDR0	RO	0x00000094	Peripheral ID 0 Register
0xFFE4	GITS_PIDR1	RO	0x000000B4	Peripheral ID 1 Register
0xFFE8	GITS_PIDR2	RO	0x0000003B	*Peripheral ID2 Register* on page 3-19

Table 3-10 ITS control register summary (continued)

Offset	Name[a]	Type	Reset	Description[b]
0xFFEC	GITS_PIDR3	RO	0x00000000	Peripheral ID 3 Register
0xFFF0	GITS_CIDR0	RO	0x0000000D	Component ID 0 Register
0xFFF4	GITS_CIDR1	RO	0x000000F0	Component ID 1 Register
0xFFF8	GITS_CIDR2	RO	0x00000005	Component ID 2 Register
0xFFFC	GITS_CIDR3	RO	0x000000B1	Component ID 3 Register

a. n corresponds to the number of a CPU interface.

b. For the description of the registers that are not specific to the GIC-500, see the *ARM® Generic Interrupt Controller Architecture Specification* version 3.0.

c. The reset value depends on the configuration of the GIC-500. See the *ARM® Generic Interrupt Controller Architecture Specification* version 3.0 GITS_TYPER, ITS Type register for more information on the acronyms. The values of the configuration dependent fields of the GITS_TYPER register are:

HCC, bits[31:24] - the number of cores in the system, plus one.

PTA, bit[19] - 0.

SEIS, bit[18] - 0.

Devbits, bits[17:13] - ITS Device ID width, minus one.

IDbits, bits[12:8] - 0b01111.

ITT Entry size, bits[7:4] - 0b0111.

Distributed, bit[3] - 0.

Virtual, bit[1] - 0.

Physical, bit[0] - 1.

3.10 ITS control register descriptions

This section only describes the Redistributor registers whose implementation is specific to the GIC-500. The *ARM® Generic Interrupt Controller Architecture Specification version 3.0.* describes all of the other registers.

3.10.1 ITS Identification Register

The GITS_IIDR characteristics are:

Purpose Provides information about the implementer and revision of the Redistributor.

Usage constraints There are no usage constraints.

Configurations Always present in the GIC-500.

Attributes See the register summary in Table 3-9 on page 3-15.

Figure 3-4 shows the bit assignments.

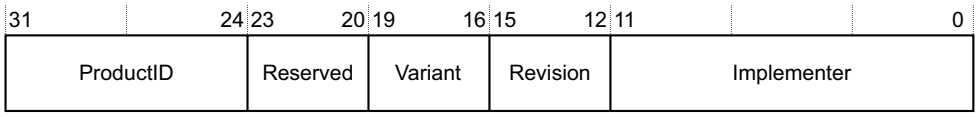

Figure 3-4 GITS_IIDR bit assignments

Table 3-11 shows the bit assignments.

Table 3-11 GITS_IIDR bit assignments

Bits	Name	Description
[31:24]	ProductID	Indicates the product ID: `0x00` GIC-500
[23:20]	-	Reserved, RAZ
[19:16]	Variant	Indicates the major revision or variant of the product: `0x0` Variant number
[15:12]	Revision	Indicates the minor revision of the product: `0x0` Revision number
[11:0]	Implementer	Indicates the implementer: `0x43B` ARM

3.10.2 Peripheral ID2 Register

The GITS_PIDR2 characteristics are:

Purpose Defines the GIC architecture version with which the GIC-500 complies.

Usage constraints There are no usage constraints.

Configurations Always present in the GIC-500.

Attributes See the register summary in Table 3-10 on page 3-17.

Figure 3-5 on page 3-20 shows the GITS_PIDR2 bit assignments.

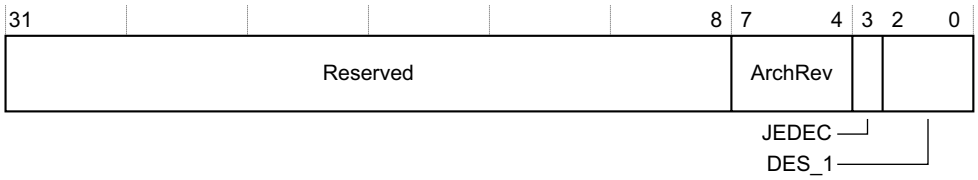

Figure 3-5 GITS_PIDR2 bit assignments

Table 3-12 shows the GITS_PIDR2 bit assignments.

Table 3-12 GITS_PIDR2 bit assignments

Bits	Name	Function
[31:8]	-	Reserved, RAZ
[7:4]	ArchRev	Identifies the version of the GIC architecture with which the GIC-500 complies: 0x3 version 3.0
[3]	JEDEC	Indicates that a JEDEC-assigned JEP106 identify code is used
[2:0]	DES_1	JEP106 identify code [6:4]

3.11 ITS translation register summary

Table 3-13 shows the address map of the ITS translation registers.

Offsets that are not shown are Reserved and RAZ/WI.

Note

This page does not exist in configurations of the GIC-500 without LPI and ITS support.

Table 3-13 ITS translation register

Offset	Name	Type	Reset	Description[a]
0x0000-0x003C	-	-	-	Reserved
0x0040	GITS_TRANSLATER	WO	-	ITS Translation Register
0x0044-0xFFFC	-	-	-	Reserved

a. For the description of the registers that are not specific to the GIC-500, see the *ARM® Generic Interrupt Controller Architecture Specification version 3.0*.

3.12 Implementation defined test registers in GICD page summary

Table 3-14 shows the address map of the Distributor test registers in the GICD page. You must not use these registers in software that is designed to be portable.

Offsets that are not shown are Reserved and RAZ/WI.

Table 3-14 Implementation defined test register summary

Offset	Name	Type	Reset	Description
0xC000	GICD_ESTATUSR	RO	0x00000000	*Extended Status Register*
0xC004	GICD_ERRTESTR	WO	-	*Error Test Register* on page 3-23
0xC084-0xC0F8	GICD_SPISRn	RO	-	*Shared Peripheral Interrupt Status Register* on page 3-24

3.12.1 Extended Status Register

The GICD_ESTATUSR characteristics are:

Purpose This register guarantees that interrupt reprogramming is complete. Use this register to support legacy software that requires this guarantee so that it works with the GIC-500.

Usage constraints There are no usage constraints.

Configurations Always present in the GIC-500.

Attributes See the register summary in Table 3-14.

Figure 3-6 shows the bit assignments.

Figure 3-6 GICD_ESTATUSR bit assignments

Table 3-15 shows the bit assignments.

Table 3-15 GICD_ESTATUSR bit assignments

Bits	Name	Description
[31]	SRWP	Super Register Write Pending. This bit denotes whether any update that changes the state of any interrupt has taken effect. If the change has not taken effect the bit reads as `0b1`, else the bit reads as `0b0`. You can use this bit to provide guarantees required by legacy software. See *Backwards compatibility* on page 2-10 for more information. This bit reads as `0b1` until any updates to the following interrupt attributes are guaranteed to be observed by all cores: • Distributor group enable. • Interrupt enable. • Group. • Priority. • Targets.
[30:0]	-	Reserved.

3.12.2 Error Test Register

The GICD_ERRTESTR characteristics are:

Purpose This register tests the integration of the **ecc_fatal** and **axim_err** signals.

Usage constraints There are no usage constraints.

Configurations Always present in the GIC-500.

Attributes See the register summary in Table 3-14 on page 3-22.

Figure 3-7 shows the bit assignments.

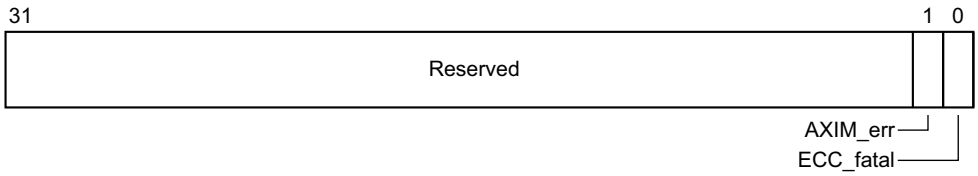

Figure 3-7 GICD_ERRTESTR bit assignments

Table 3-16 shows the bit assignments.

Table 3-16 GICD_ERRTESTR bit assignments

Bits	Name	Description
[31:2]	-	Reserved.
[1][a]	AXIM_err	Write `0b1` to this field to drive the **axim_err** pin to `0b1` for 1 cycle. You can use this bit for an integration test of the **axim_err** signal.
[0]	ECC_fatal	Write `0b1` to this field to drive the **ecc_fatal** pin to `0b1` for 1 cycle. You can use this bit for an integration test on the **ecc_fatal** signal.

a. The existence of this field depends on the configuration of the GIC-500. If ITS and LPI support is not included, this field is Reserved and RAZ/WI.

3.12.3 Shared Peripheral Interrupt Status Register

The GICD_SPISRn characteristics are:

Purpose Enables a core to access the status of the SPI wire inputs on the Distributor.

Note

In systems with two security states, Non-secure accesses can only read the status of Non-secure Group 1 interrupts.

Usage constraints The Distributor provides up to 30 registers to support 960 SPIs. If you configure the GIC-500 to use fewer than 960 SPIs, it reduces the number of registers accordingly. For locations where interrupts are not implemented, the register is RAZ/WI.

Configurations Always present in the GIC-500.

Attributes See the register summary in Table 3-14 on page 3-22.

Figure 3-8 shows the bit assignments.

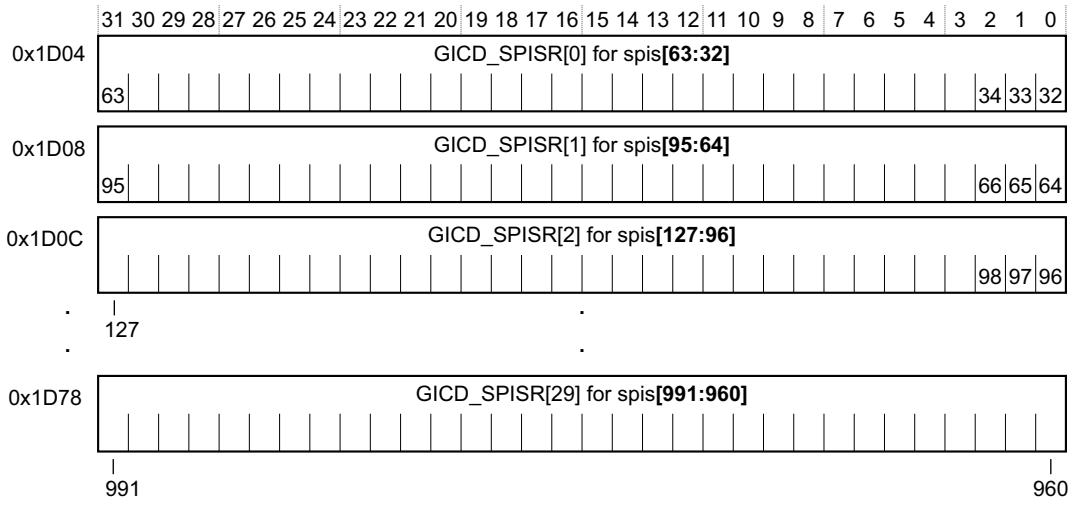

Figure 3-8 GICD_SPISRn bit assignments

Table 3-17 shows the bit assignments.

Table 3-17 GICD_SPISRn bit assignments

Bits	Name	Function
[31:0]	SPIS[N+31:N]	Returns the status of the **spis** inputs on the Distributor. For each bit: `0` **spis** is LOW. `1` **spis** is HIGH. **Note** • The **spis** that a bit refers to depends on its bit position and the base address offset of the *Shared Peripheral Interrupt Status Registers*, GICD_SPISRn. • These bits return the actual status of the **spis** input signals. The *Interrupt Set-Pending Register*, GICD_ISPENDRn and *Interrupt Clear-Pending Register*, GICD_ICPENDRn, can also provide the **spis** status but because you can write to these registers, they might not contain the actual status of the **spis** signals.

3.13 Implementation defined test registers in the GICR page for PPIs and SGIs

Table 3-18 shows the address map for the implementation defined test registers in the GICR page for PPIs and SGIs.

Offsets that are not shown are Reserved and RAZ/WI.

Table 3-18 Implementation defined test register summary

Offset	Name	Type	Reset	Description
0xC000	GICR_MISCSTATUSR	RO	-	*Miscellaneous Status Register*
0xC080	GICR_PPISR	RO	-	*Private Peripheral Interrupt Status Register* on page 3-26

3.13.1 Miscellaneous Status Register

The GICR_MISCSTATUSR characteristics are:

Purpose Use this register to test the integration of the **cpu_active** input signals and to debug the CPU interface enables as seen by the GIC-500.

Usage constraints There are no usage constraints.

Configurations Always present in the GIC-500.

Attributes See the register summary in Table 3-18.

Figure 3-9 shows the bit assignments.

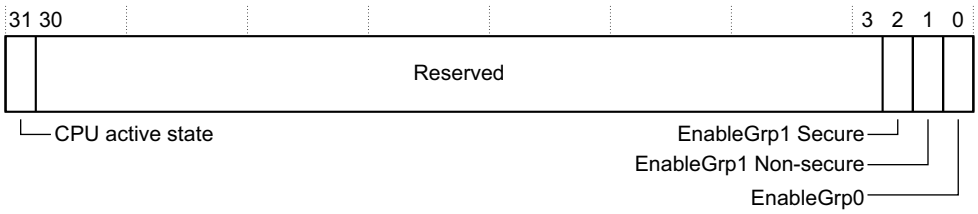

Figure 3-9 GICR_MISCSTATUSR bit assignments

Table 3-19 shows the bit assignments.

Table 3-19 GICR_MISCSTATUSR bit assignments

Bits	Name	Description
[31][a]	CPU active state	This bit returns the actual status of the **cpu_active** signal for the core corresponding to the Redistributor whose register is being read. That is, this field is one when the corresponding **cpu_active** input is HIGH.
[30:3]	-	Reserved.

Table 3-19 GICR_MISCSTATUSR bit assignments (continued)

Bits	Name	Description
[2][b]	EnableGrp1 Secure	In systems with two security states enabled, that is, when GICD_CTLR.DS is set to zero: • For Secure reads, returns the Group 1 Secure CPU interface enable. • For Non-secure reads, returns zero. In systems with only a single security state enabled, that is, when GICD_CTLR.DS is set to one: • Returns zero.
[1][b]	EnableGrp1 Non-secure	In systems with two security states enabled, that is, when GICD_CTLR.DS is set to zero: • For Secure reads, returns the Group 1 Non-secure CPU interface enable. • For Non-secure reads when GICD_CTLR.ARE_NS is one, returns the Group 1 Non-secure CPU interface enable. • For Non-secure reads when GICD_CTLR.ARE_NS is zero, returns zero. In systems with only a single security state enabled, that is, when GICD_CTLR.DS is set to one: • Returns the Group 1 CPU interface enable.
[0][b]	EnableGrp0	In systems with two security states enabled, that is, when GICD_CTLR.DS is set to zero: • For Secure reads, returns the Group 0 CPU interface enable. • For Non-secure reads when GICD_CTLR.ARE_NS is zero, returns the Group 1 Non-secure CPU interface enable. • For Non-secure reads when GICD_CTLR.ARE_NS is one, returns zero. In systems with only a single security state enabled, that is, when GICD_CTLR.DS is set to one: • Returns the Group 0 CPU interface enable.

a. This bit is undefined when ProcessorSleep or ChildrenAsleep is set for a core, because the core is presumed to be powered down.

b. These bits are a copy of the CPU interface group enables for the core corresponding to this Redistributor. These copies are undefined when ProcessorSleep or ChildrenSleep is set for a core, because the core is presumed to be powered down. These copies, which are maintained by Upstream Write packets, can become de-synchronized after an incorrect powerdown sequence. This debug register enables you to debug this scenario. For more information see the *ARM® Generic Interrupt Controller Architecture Specification* version 3.0.

3.13.2 Private Peripheral Interrupt Status Register

The GICR_PPISR characteristics are:

Purpose This register enables a core to access the status of the PPI inputs to the Distributor.

Note

In systems with two security states, Non-secure accesses can only read the status of Non-secure Group 1 interrupts.

Usage constraints There are no usage constraints.

Configurations Always present in the GIC-500.

Attributes See the register summary in Table 3-18 on page 3-25.

Figure 3-10 shows the bit assignments.

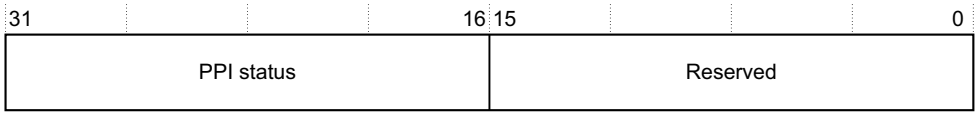

Figure 3-10 GICR_PPISR bit assignments

Table 3-20 shows the bit assignments.

Table 3-20 GICR_PPISR bit assignments

Bits	Name	Description
[31:16]	PPI status	These bits return the actual status of the PPI input signals. That is, each bit is one when the corresponding PPI input is HIGH for the core whose register is being read. The position of each bit corresponds to the interrupt ID of the PPI. For example, bit[30] corresponds to PPI ID 30. The Interrupt Set-Pending Register, GICR_ISPENDR, and Interrupt Clear-Pending Register, GICR_ICPENDR, can also provide the PPI status but because you can write to these registers, they might not contain the true status of the PPI input signals.
[15:0]	-	Reserved

3.14 Implementation defined test registers in the GITS control page summary

Table 3-21 shows the address map of the implementation defined test registers in the GITS control page.

Offsets that are not shown are Reserved and RAZ/WI.

Table 3-21 Implementation defined test register summary

Offset	Name	Type	Reset	Description
0xC000	GITS_TRKCTLR	WO	-	*Tracking Control Register*
0xC004	GITS_TRKR	RO	0x00000000	*Tracking Status Register* on page 3-29
0xC008	GITS_TRKDIDR	RO	0x00000000	*Debug Tracked DID Register* on page 3-30
0xC00C	GITS_TRKPIDR	RO	0x00000000	*Debug Tracked PID Register* on page 3-31
0xC010	GITS_TRKVIDR	RO	0x00000000	*Debug Tracked ID Register* on page 3-31
0xC014	GITS_TRKTGTR	RO	0x00000000	*Debug Tracked Target Register* on page 3-32
0xC018	GITS_TRKICR	RO	0x00000000	*Debug ITE Cache Statistics* on page 3-32
0xC01C	GITS_TRKLCR	RO	0x00000000	*Debug LPI Cache Statistics* on page 3-33

3.14.1 Tracking Control Register

The GITS_TRKCTLR characteristics are:

Purpose Use this register to control LPI related debug and performance measurement features for the GIC-500.

Usage constraints There are no usage constraints.

Configurations Present in configurations of the GIC-500 with ITS and LPI support.

Attributes See the register summary in Table 3-21.

Figure 3-11 shows the bit assignments.

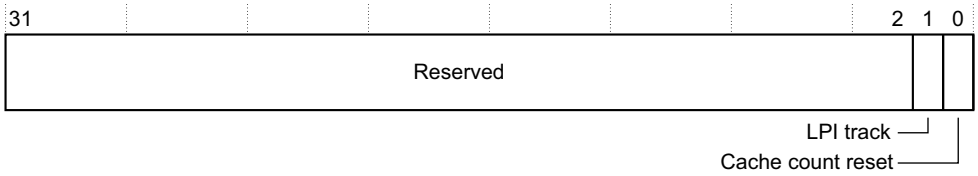

Figure 3-11 GITS_TRKCTLR bit assignments

Table 3-22 shows the bit assignments.

Table 3-22 GITS_TRKCTRL bit assignments

Bits	Name	Description
[31:2]	-	Reserved
[1]	LPI track	Write `0b1` to capture information about the next interrupt that the ITS generated, or failed to generate because of misprogramming. This information can then be inspected using the other registers in this page. Before using this bit, you must: • Enable GITS_CTLR. • Program GITS_BASER0 and GITS_CBASER to be valid. • Program GICR_WAKER to not have Sleep set. • Ensure that GICR_WAKER is also not Quiescent. **Note** This register also captures certain ITS commands. Therefore ARM recommends that it is only used when the ITS command queue is empty.
[0]	Cache count reset	Write `0b1` to reset the cache hit and miss counters in GITS_TRKICR and GITS_TRKLCR.

3.14.2 Tracking Status Register

The GITS_TRKR characteristics are:

Purpose This register provides some of the tracked state that is triggered by the LPI track bit in the GITS_TRKCTLR register. Use this register to find out whether an LPI was generated by the ITS and, if not, why it was not generated. You must examine the least significant bits first.

Usage constraints There are no usage constraints.

Configurations Present in configurations of the GIC-500 with ITS and LPI support.

Attributes See the register summary in Table 3-21 on page 3-28.

Figure 3-12 shows the bit assignments.

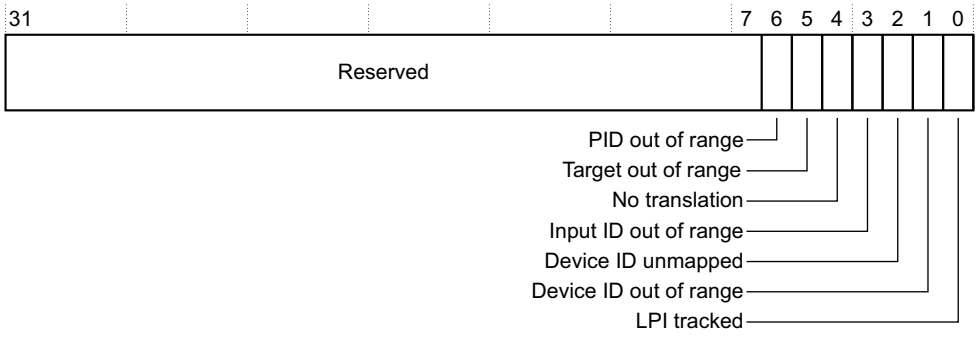

Figure 3-12 GITS_TRKR bit assignments

Table 3-23 shows the bit assignments.

Table 3-23 GITS_TRKR bit assignments

Bits	Name	Description
[31:7]	-	Reserved
[6]	PID out of range	When it is 1, this bit indicates that the LPI PID is larger than that allowed by the IDbits field in the GICR_PROPBASER. This bit is only valid if bits [4:1] are zero and bit [0] is 1.
[5]	Target out of range	When it is 1, indicates that target collection has not been successfully mapped using MAPC, or that the target core does not have LPIs enabled in GICR_CTLR. This bit is only valid if bits [4:1] are zero and bit [0] is 1.
[4]	No translation	When it is 1, indicates that no valid MAPI or MAPVI has successfully been performed for this combination of input ID and Device ID. For example, the command might have failed because of an illegal collection or PID. This bit is only valid if bits [3:1] are zero and bit [0] is 1.
[3]	Input ID out of range	When it is 1, indicates that the input ID is larger than that allowed by the Device ID, which is set during the MAPD command, or it is larger than 65535. This bit is only valid if bits [2:1] are zero and bit [0] is 1.
[2]	Device ID unmapped	When it is 1, indicates that no valid MAPD has successfully been performed for this Device ID. This bit is only valid if bit [1] is zero and bit [0] is 1.
[1]	Device ID out of range	When it is set to 1, indicates that the Device ID is larger than that allowed by the Size and Page Size in GITS_BASER0, or larger than the number of Device IDs configured. See *Configurable options* on page 1-9 for more information.
[0]	LPI tracked	When it is 1, the LPI tracking initiated by the LPI track bit in the GITS_TRKCTLR register is completed and the contents of the Debug Tracked registers are valid.

3.14.3 Debug Tracked DID Register

The GITS_TRKDIDR characteristics are:

Purpose This register returns the Device ID of the LPI that is tracked as a result of the LPI track bit in the GITS_TRKCTLR register being set to `0b1`.

Usage constraints This register is valid when the LPI tracked field in GITS_TRKR is 1.

Configurations Present in configurations of the GIC-500 with ITS and LPI support.

Attributes See the register summary in Table 3-21 on page 3-28.

Figure 3-13 shows the bit assignments.

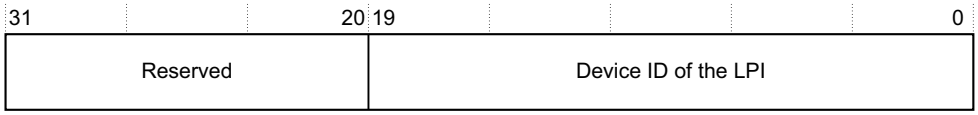

Figure 3-13 GITS_SPISRn bit assignments

Table 3-24 shows the bit assignments.

Table 3-24 GITS_SPISRn bit assignments

Bits	Name	Function
[31:20]	-	Reserved
[19:0]	Device ID of LPI	The Device ID for the interrupt that was tracked.

3.14.4 Debug Tracked PID Register

The GITS_TRKPIDR characteristics are:

Purpose This register returns the ID of the LPI that is tracked as a result of the LPI track bit in the GITS_TRKCTLR register being set to `0b1`.

Usage constraints This register is only valid when the LPI tracked field in GITS_TRKR is 1 and none of the other bits in the register is 1.

Configurations Present in configurations of the GIC-500 with ITS and LPI support.

Attributes See the register summary in Table 3-21 on page 3-28.

Figure 3-14 shows the bit assignments.

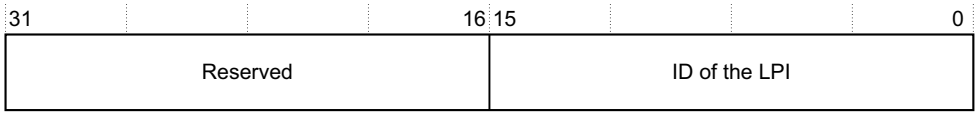

Figure 3-14 GITS_TRKPIDR bit assignments

Table 3-25 shows the bit assignments.

Table 3-25 GITS_TRKPIDR bit assignments

Bits	Name	Function
[31:16]	-	Reserved.
[15:0]	ID of LPI	The ID after translation for an interrupt that was tracked and generated an LPI successfully. This is the PID of the generated LPI.

3.14.5 Debug Tracked ID Register

The GITS_TRKVIDR characteristics are:

Purpose This register returns the ID received by the ITS that is tracked as a result of LPI track bit in the GITS_TRKCTLR register being set to `0b1`.

Usage constraints This register is valid when the LPI tracked field in GITS_TRKR is 1.

Configurations Present in configurations of the GIC-500 with ITS and LPI support.

Attributes See the register summary in Table 3-21 on page 3-28.

Figure 3-15 on page 3-32 shows the bit assignments.

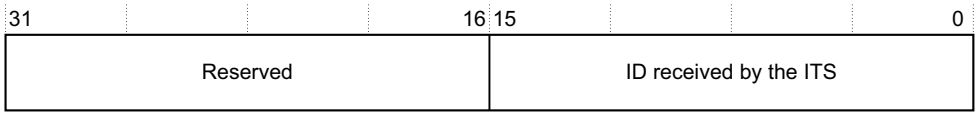

Figure 3-15 GITS_TRKVIDR bit assignments

Table 3-26 shows the bit assignments.

Table 3-26 GITS_TRKVIDR bit assignments

Bits	Name	Function
[31:16]	-	Reserved
[15:0]	ID received by ITS	The ID before translation of the interrupt that was tracked

3.14.6 Debug Tracked Target Register

The GITS_TRKTGTR characteristics are:

Purpose This register returns the target core of the LPI that is tracked as a result of the LPI track bit in the GITS_TRKCTLR register being set to `0b1`.

Usage constraints This register is only valid when the LPI tracked field in GITS_TRKR is 1 and none of the other bits in the register is 1.

Configurations Present in configurations of the GIC-500 with ITS and LPI support.

Attributes See the register summary in Table 3-21 on page 3-28.

Figure 3-16 shows the bit assignments.

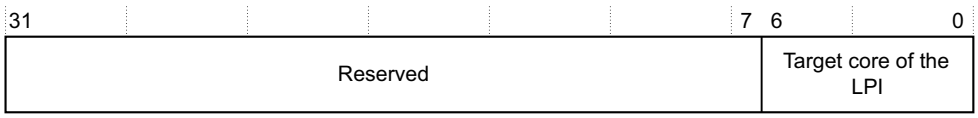

Figure 3-16 GITS_TRKTGTR bit assignments

Table 3-27 shows the bit assignments.

Table 3-27 GITS_TRKTGTR bit assignments

Bits	Name	Function
[31:7]	-	Reserved
[6:0]	Target core of the LPI	The target core for an interrupt that was tracked and generated an LPI successfully. This is given using the linerar representation described in *Topologies and terminology* on page 1-4. The linear representation also corresponds to the Processor Number field in the GICR_TYPER of the target core.

3.14.7 Debug ITE Cache Statistics

The GITS_TRKICR characteristics are:

Purpose This register gives the count of hits and misses in the ITE cache since the last time the Cache Count Reset bit in the GITS_TRKCTLR register was set to `0b1`.

Use this register to decide whether you want the ITE cache to be any larger. If the ratio of hits to misses is low, you can increase the size of the ITE cache to reduce latency and power consumption. The ITE cache has the same number of entries as the LPI cache, so you can increase its size using the LPI cache entries parameter. See *Configurable options* on page 1-9.

Usage constraints There are no usage constraints.

Configurations Present in configurations of the GIC-500 with ITS and LPI support.

Attributes See the register summary in Table 3-21 on page 3-28.

Figure 3-17 shows the bit assignments.

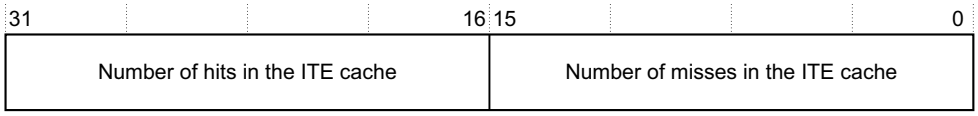

Figure 3-17 GITS_TRKICR bit assignments

Table 3-28 shows the bit assignments.

Table 3-28 GITS_TRKICR bit assignments

Bits	Name	Function
[31:16]	ITE cache hits	Records the number of hits in the ITE cache
[15:0]	ITE cache misses	Records the number of misses in the ITE cache

3.14.8 Debug LPI Cache Statistics

The GITS_TRKLCR characteristics are:

Purpose This register gives the count of hits and misses in the LPI cache since the last time the Cache Count Reset bit in the GITS_TRKCTLR register was set to `0b1`.

Use this register to decide whether you want the LPI cache to be any larger. If the ratio of hits to misses is low, you can increase the size of the LPI cache to reduce latency and power consumption. You can increase its size using the LPI cache entries parameter. See *Configurable options* on page 1-9.

Usage constraints There are no usage constraints.

Configurations Present in configurations of the GIC-500 with ITS and LPI support.

Attributes See the register summary in Table 3-21 on page 3-28.

Figure 3-18 shows the bit assignments.

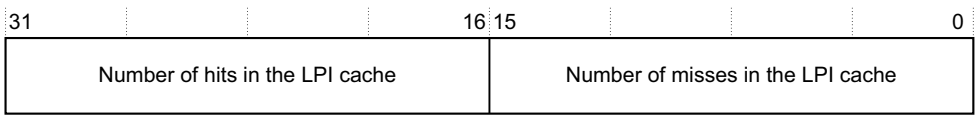

Figure 3-18 GITS_TRKLCR bit assignments

Table 3-29 shows the bit assignments.

Table 3-29 GITS_TRKLCR bit assignments

Bits	Name	Function
[31:16]	LPI cache hits	Records the number of hits in the LPI cache
[15:0]	LPI cache misses	Records the number of misses in the LPI cache

Appendix A
Signal Descriptions

This appendix describes the signals that the GIC-500 provides. It contains the following sections:

A.1 Clock and reset signals

Table A-1 shows the clock signals.

Note

The GIC-500 does not synchronize any inputs, so all input signals, including the SPI and PPI inputs, must be synchronous to **clk**.

Table A-1 Clock and reset signals

Signal name	Type	Source/destination	Description
clk	Input	Clock source	Common global clock signal for AXI and other interfaces. All clock gating is internal to the noram.
resetn	Input	Reset source	Common reset for all interfaces. Active LOW global asynchronous reset. Reset must be asserted for at least (3 + number of synchronizer stages) clock cycles. For example, 3+2 = 5 cycles.

A.2 Miscellaneous signals

Table A-2 shows the miscellaneous signals for the GIC-500.

Table A-2 Miscellaneous signals

Signal[a]	Type	Source/destination	Description
cpu_active_<x>[n:0]	Input	Power controller	Controls whether the GIC-500 considers a core as its first choice for an SPI that targets multiple cores. ARM recommends that this signal is typically driven HIGH during normal operation and driven LOW during certain sleep states, such as retention, so the core is likely to stay in the sleep state for longer. If this signal is not used, it must be tied HIGH.
wake_request_<x>[n:0]	Output		Indicates a directed interrupt is pending for a core that has set GICR_WAKER.ProcessorSleep. This signal is expected to cause the core to power up and subsequently re-enable processing interrupts.
ecc_fatal	Output	System controller	Indicates an uncorrectable ECC error.
axim_err	Output		Indicates a bus error received by the AXI4 master port, such as a SLVERR or DECERR.

a. n denotes the number of the last core in the cluster <x>.

A.3 Interrupt signals

Table A-3 shows the interrupt inputs for the GIC-500.

Table A-3 Interrupt signals

Signal[a]	Type	Source/destination	Description
spi[variable]	Input[b]	Peripherals	SPI inputs. The least significant bit of this signal is bit 32, which corresponds to interrupt ID 32. **Note** This represents a change from ARM interrupt controllers such as GIC-400, where the LSB is bit 0, but still has ID 32.
ppi31_<x>[n:0]	Input[c]	Peripherals local to a core	PPI ID31.
ppi30_<x>[n:0]			PPI ID30. Typically the Non-secure physical timer.
ppi29_<x>[n:0]			PPI ID29. Typically the Secure physical timer.
ppi28_<x>[n:0]			PPI ID28.
ppi27_<x>[n:0]			PPI ID27. Typically the virtual timer.
ppi26_<x>[n:0]			PPI ID26. Typically the hypervisor timer.
ppi25_<x>[n:0]			PPI ID25. Typically the virtual CPU interface maintenance interrupt.
ppi24_<x>[n:0]			PPI ID24. Typically the *Cross Trigger Interface* (CTI) interrupt.
ppi23_<x>[n:0]			PPI ID23. Typically the *Performance Counter* (PMU) overflow interrupt.
ppi22_<x>[n:0]			PPI ID22. Typically the *Debug Communications Channel* (DCC) interrupt.
ppi21_<x>[n:0]			PPI ID21.
ppi20_<x>[n:0]			PPI ID20.
ppi19_<x>[n:0]			PPI ID19.
ppi18_<x>[n:0]			PPI ID18.
ppi17_<x>[n:0]			PPI ID17.
ppi16_<x>[n:0]			PPI ID16.

a. n denotes the number of the last core in the cluster <x>.

b. This input is either active HIGH level-sensitive, or triggered on a rising edge, depending on the software programming.

c. All these inputs are either active LOW level-sensitive, or triggered on a rising edge, depending on the software programming.

A.4 Test signals

Table A-4 shows the test signals for the GIC-500. These signals must all be LOW during normal operation.

Table A-4 Test signals

Signal	Type	Source/destination	Description
dftrstdisable	Input	DFT control logic	Reset disable. Disables the external reset input for test mode. When this signal is HIGH, it forces the internal active-LOW reset HIGH, bypassing the reset synchronizer.
dftse	Input		Scan enable. Disables clock gates for test mode.
dftcgen	Input		Clock gate enable. When this signal is HIGH, it forces all the clock gates on so that all internal clocks always run.
dftramhold	Input		RAM hold. When this signal is HIGH, it forces all the RAM chip selects LOW, preventing accesses to the RAMs.

A.5 AXI4 slave interface signals

The GIC-500 provides a 32-bit wide AXI4 slave interface. See the *ARM® AMBA® AXI and ACE Protocol Specification*.

AXI4 signals that are not implemented in the GIC-500 are not shown in Table A-5.

Table A-5 The GIC-500 implementation of AXI4 slave signals

AXI signal	Type	Source/destination	GIC-500 implementation
Write address channel signals			
awuser_s[variable][a]	Input	AXI4 interconnect	Non-standard width with respect to the *ARM® AMBA® AXI and ACE Protocol Specification*. Bits [2:0] must contain the linear ID of the core performing the write, if backwards compatibility mode is supported. See *Topologies and terminology* on page 1-4 for more information. Bit [3] indicates a GITS_TRANSLATER write, overriding the **awaddr_s**, in which the DeviceID comes from **awaddr_s[n:2]**. If bit[3] is not set and if there is a GITS_TRANSLATER write, the DeviceID comes from the **awuser_s[n:4]**.
awaddr_s[variable][a]	Input		Non-standard width with respect to the *ARM® AMBA® AXI and ACE Protocol Specification*.
awid_s[variable][a]	Input		Non-standard width with respect to the *ARM® AMBA® AXI and ACE Protocol Specification.*
awlen_s[7:0]	Input		See the *ARM® AMBA® AXI and ACE Protocol Specification.*
awsize_s[2:0]	Input		Non-standard width with respect to the *ARM® AMBA® AXI and ACE Protocol Specification.*
awburst_s[1:0]	Input		See the *ARM® AMBA® AXI and ACE Protocol Specification.*
awprot_s[2:0]	Input		
awvalid_s	Input		
awready_s	Output		
Write data channel signals			
wstrb_s[3:0]	Input	AXI4 interconnect	See the *ARM® AMBA® AXI and ACE Protocol Specification.*
wdata_s[31:0]	Input		
wvalid_s	Input		
wready_s	Output		
Write response channel signals			
bid_s[variable][a]	Output	AXI4 interconnect	Non-standard width with respect to the *ARM® AMBA® AXI and ACE Protocol Specification.*
bvalid_s	Output		See the *ARM® AMBA® AXI and ACE Protocol Specification.*
bready_s	Input		
bresp_s[1:0]	Output		

Table A-5 The GIC-500 implementation of AXI4 slave signals (continued)

AXI signal	Type	Source/destination	GIC-500 implementation
Read address channel signals			
araddr_s[variable][a]	Input	AXI4 interconnect	Non-standard width with respect to the *ARM® AMBA® AXI and ACE Protocol Specification.*
arid_s[variable][a]	Input		Non-standard width with respect to the *ARM® AMBA® AXI and ACE Protocol Specification*.
arlen_s[7:0]	Input		See the *ARM® AMBA® AXI and ACE Protocol Specification.*
arsize_s[2:0]	Input		
aruser_s[2:0]	Input		Non-standard width with respect to the *ARM® AMBA® AXI and ACE Protocol Specification.* Bits [2:0] must contain the linear ID of the core performing the read, if backwards compatibility mode is supported. See *Topologies and terminology* on page 1-4 for more information. Bit [3] indicates a GITS_TRANSLATER write, overriding the **araddr_s**, in which the DeviceID comes from **araddr_s[n:2]**. If bit[3] is not set, if there is a GITS_TRANSLATER write, the DeviceID comes from the **aruser_s[n:4]**.
arburst_s[1:0]	Input		See the *ARM® AMBA® AXI and ACE Protocol Specification.*
arprot_s[2:0]	Input		
arvalid_s	Input		
arready_s	Output		
Read data channel signals			
rid_s[variable][a]	Output	AXI4 interconnect	Non-standard width with respect to the *ARM® AMBA® AXI and ACE Protocol Specification*.
rdata_s[31:0]	Output		See the *ARM® AMBA® AXI and ACE Protocol Specification.*
rresp_s[1:0]	Output		
rlast_s	Output		
rvalid_s	Output		
rready_s	Input		

a. The variable is configuration dependent.

A.6 AXI4 master interface signals

The GIC-500 provides a 64-bit wide AXI4 master interface. See the *ARM® AMBA® AXI and ACE Protocol Specification.*

AXI4 signals that are not implemented in the GIC-500 are not shown in Table A-6.

Table A-6 GIC-500 implementation of AXI4 master signals

AXI signal name	Type	Source/destination	Description
Write address channel signals			
awaddr_m[47:0]	Output	AXI4 interconnect	Non-standard width with respect to the *ARM® AMBA® AXI and ACE Protocol Specification*
awid_m[5:0]	Output		Non-standard width with respect to the *ARM® AMBA® AXI and ACE Protocol Specification*
awvalid_m	Output		See the *ARM® AMBA® AXI and ACE Protocol Specification*
awready_m	Input		
awlen_m[7:0]	Output		
awsize_m[2:0]	Output		
awburst_m[1:0]	Output		
awprot_m[2:0]	Output		
awcache_m[3:0]	Output		
Write data channel signals			
wstrb_m[7:0]	Output	AXI4 interconnect	See the *ARM® AMBA® AXI and ACE Protocol Specification*
wdata_m[63:0]	Output		
wlast_m	Output		
wvalid_m	Outut		
wready_m	Input		
Write response channel signals			
bid_m[5:0]	Input	AXI4 interconnect	Non-standard width with respect to the *ARM® AMBA® AXI and ACE Protocol Specification*
bvalid_m	Input		See the *ARM® AMBA® AXI and ACE Protocol Specification*
bready_m	Output		
bresp_m[1:0]	Input		
Read address channel signals			
araddr_m[47:0]	Output	AXI4 interconnect	Non-standard width with respect to the *ARM® AMBA® AXI and ACE Protocol Specification*
arid_m[5:0]	Output		Non-standard width with respect to the *ARM® AMBA® AXI and ACE Protocol Specification*

Table A-6 GIC-500 implementation of AXI4 master signals (continued)

AXI signal name	Type	Source/destination	Description
arvalid_m	Output	AXI4 interconnect	See the *ARM® AMBA® AXI and ACE Protocol Specification*
arlen_m[7:0]	Output		
arsize_m[2:0]	Output		
arburst_m[1:0]	Output		
arprot_m[2:0]	Output		
arcache_m[3:0]	Output		
arready_m	Input		
Read data channel signals			
rid_m[5:0]	Input	AXI4 interconnect	Non-standard width with respect to the *ARM® AMBA® AXI and ACE Protocol Specification*
rresp_m[1:0]	Input		See the *ARM® AMBA® AXI and ACE Protocol Specification*
rdata_m[63:0]	Input		
rvalid_m	Input		
rready_m	Output		

A.7 GIC Stream master interfaces

Table A-7 shows the GIC Stream master interfaces from the GIC-500.

Table A-7 GIC Stream master interfaces

Signal name[a]	Type	Source/destination	Description
icdtready<x>	Input	CPU interface	See the *ARM® Generic Interrupt Controller Stream Protocol Interface Specification*
icdtvalid<x>	Output		
icdtlast<x>	Output		
icdtdata<x>[15:0]	Output		
icdtdest_<x>[2:0]	Output		Contains the core number within the cluster

a. <x> denotes the cluster number.

A.8 GIC Stream slave interfaces

Table A-8 shows the GIC Stream slave interfaces to the GIC-500.

Table A-8 GIC Stream slave interfaces

Signal name[a]	Type	Source/destination	Description
icctready_<x>	Output	CPU interface	See the *ARM® Generic Interrupt Controller Stream Protocol Interface Specification*
icctvalid_<x>	Input		
icctlast_<x>	Input		
icctdata_<x>[15:0]	Input		
icctid_<x>[2:0]	Input		Contains the core number within the cluster

a. <x> denotes the cluster number.

A.9 MBIST interface signals

Table A-9 shows MBIST interface signals.

Table A-9 MBIST interface signals

Port Name	Type	Source/destination	Description
mbistack	Output	MBIST controller	MBIST Mode Ready. GIC-500 acknowledges that it is ready for MBIST testing.
mbistreq	Input		MBIST Mode Request. Request to GIC-500 to enable MBIST testing. This signal must be tied LOW during functional operation.
mbistresetn	Input		Resets MBIST logic. Resets functional logic to enable MBIST operation by an Active-LOW signal. This signal must be tied HIGH during functional operation.
mbistaddr[variable][a]	Input		Logical address. The width is based on the RAM with the largest number of words. You must drive the most significant bits to zero when accessing RAMs with fewer address bits.
mbistindata[variable][a]	Input		Data in. Write data. Width based on the RAM with the largest number of data bits.
mbistoutdata[variable][a]	Output		Data out. Read data. Width based on the RAM with the largest number of data bits.
mbistwriteen	Input		Write control (**mbistwriteen**) and Read control (**mbistreaden**) No access occurs if both enables are off. It is illegal to activate both enables simultaneously.
mbistreaden	Input		
mbistarray[1:0]	Input		Array selector. This controls which RAM array is accessed. For the single RAM configuration, this port is unused.
mbistcfg	Input		MBIST ALL enable. When enabled, allows simultaneous access to all RAM arrays for maximum array power consumption.

a. The variable is configuration dependent.

Appendix B
Revisions

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

Table B-1 Issue A

Change	Location	Affects
First release	-	-

Table B-2 Differences between issue A and issue B

Change	Location	Affects
Distributor register GICD_PIDR4 reset value updated from `0x00000004` to `0x00000044`	Table 3-3 on page 3-6	All revisions
Redistributor register GICR_PIDR4 reset value updated from `0x00000004` to `0x00000044`	Table 3-7 on page 3-12	All revisions
ITS control register GITS_PIDR4 reset value updated from `0x00000004` to `0x00000044`	Table 3-10 on page 3-17	All revisions