# Arm® CoreLink™ GIC-600 Generic Interrupt Controller

**Revision: r1p3**

## Technical Reference Manual

arm


100336_0103_00_en

# Arm® CoreLink™ GIC-600 Generic Interrupt Controller

## Technical Reference Manual



### Release Information

### Document History

| Issue | Date | Confidentiality | Change |
|---|---|---|---|
| 0000-00 | 29 July 2016 | Confidential | First release for r0p0 |
| 0000-01 | 26 October 2016 | Confidential | Second release for r0p0 |
| 0001-00 | 31 March 2017 | Confidential | First release for r0p1 |
| 0002-00 | 07 June 2017 | Confidential | First release for r0p2 |
| 0002-01 | 15 June 2017 | Non-Confidential | Second release for r0p2 |
| 0003-00 | 08 January 2018 | Non-Confidential | First release for r0p3 |
| 0102-00 | 02 February 2018 | Non-Confidential | First release for r1p2 |
| 0103-00 | 27 March 2018 | Non-Confidential | First release for r1p3 |

### Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

**Confidentiality Status**

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

Unrestricted Access is an Arm internal classification.

**Product Status**

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

**Web Address**

*http://www.arm.com*

# Contents
# Arm® CoreLink™ GIC-600 Generic Interrupt Controller Technical Reference Manual


***Preface***

*About this book* ..... *7*

*Feedback* ..... *10*

***Chapter 1 Introduction***

*1.1 About the GIC-600* ..... *1-12*

*1.2 Components* ..... *1-13*

*1.3 Compliance* ..... *1-18*

*1.4 Features* ..... *1-19*

*1.5 Test features* ..... *1-20*

*1.6 Product documentation* ..... *1-21*

*1.7 Product revisions* ..... *1-22*

***Chapter 2 Components and configuration***

*2.1 Distributor* ..... *2-24*

*2.2 Redistributor* ..... *2-30*

*2.3 ITS* ..... *2-34*

*2.4 MSI-64 Encapsulator* ..... *2-39*

*2.5 SPI Collator* ..... *2-42*

*2.6 Wake Request* ..... *2-44*

*2.7 Interconnect* ..... *2-45*

*2.8 Hierarchy* ..... *2-46*

**Chapter 3 Operation**

3.1 Interrupt types ..... 3-49
3.2 Interrupt groups ..... 3-52
3.3 Physical interrupt signals (PPIs and SPIs) ..... 3-53
3.4 Affinity routing and assignment ..... 3-54
3.5 1 of N SPI interrupt selection ..... 3-56
3.6 Power management ..... 3-58
3.7 Getting started ..... 3-61
3.8 Security ..... 3-62
3.9 Backwards compatibility ..... 3-64
3.10 Interrupt translation service (ITS) ..... 3-65
3.11 LPI caching ..... 3-68
3.12 Memory access and attributes ..... 3-69
3.13 MSI-64 ..... 3-71
3.14 RAM ..... 3-72
3.15 Performance Monitoring Unit ..... 3-73
3.16 Reliability, Accessibility, and Serviceability ..... 3-75
3.17 Multichip operation ..... 3-97

**Chapter 4 Programmers model**

4.1 The GIC-600 registers ..... 4-104
4.2 Distributor registers (GICD/GICDA) summary ..... 4-106
4.3 Distributor registers (GICA) for message-based SPIs summary ..... 4-125
4.4 Redistributor registers for control and physical LPIs summary ..... 4-126
4.5 Redistributor registers for SGIs and PPIs summary ..... 4-135
4.6 ITS control register summary ..... 4-141
4.7 ITS translation register summary ..... 4-151
4.8 GICT register summary ..... 4-152
4.9 GICP register summary ..... 4-170

**Appendix A Signal descriptions**

A.1 Common control signals ..... Appx-A-185
A.2 Power control signals ..... Appx-A-187
A.3 Interrupt signals ..... Appx-A-188
A.4 CPU interface signals ..... Appx-A-189
A.5 ACE interface signals ..... Appx-A-190
A.6 Miscellaneous signals ..... Appx-A-195
A.7 Interblock signals ..... Appx-A-196
A.8 Interdomain signals ..... Appx-A-199
A.9 Interchip signals ..... Appx-A-200

**Appendix B Implementation-defined features**

B.1 Implementation-defined features reference ..... Appx-B-202

**Appendix C Revisions**

C.1 Revisions ..... Appx-C-205

# Preface

This preface introduces the *Arm® CoreLink™ GIC-600 Generic Interrupt Controller Technical Reference Manual*.

It contains the following:


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

# About this book

This book is for the Arm® CoreLink™ GIC-600 Generic Interrupt Controller.

## Product revision status

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

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

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

## Intended audience

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

## Using this book

This book is organized into the following chapters:

***Chapter 1 Introduction***
Read this for an introduction to the GIC-600 and its features.

***Chapter 2 Components and configuration***
Read this for a description of the major components of the GIC-600.

***Chapter 3 Operation***
Read this for an operational description of the GIC-600.

***Chapter 4 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 Implementation-defined features***
Read this for a description of the features that are IMPLEMENTATION-DEFINED.

***Appendix C Revisions***
Read this for a description of changes between released issues of this book.

### Glossary

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

See the *Arm® Glossary* for more information.

### Typographic conventions

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

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

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

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

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

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

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

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

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

### Timing diagrams

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

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

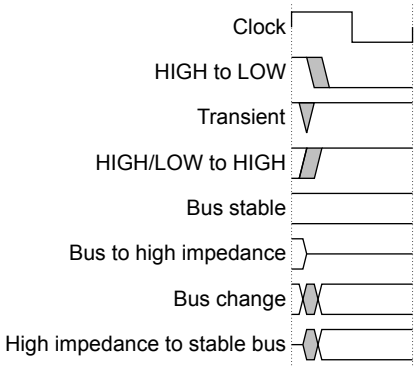


**Figure 1 Key to timing diagram conventions**

### Signals

The signal conventions are:

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

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

## Additional reading

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

**Arm publications**

- *Arm® AMBA® AXI and ACE Protocol Specification* (IHI 0022).
- *Arm® AMBA® 4 AXI4-Stream Protocol Specification* (IHI 0051).
- *AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces* (IHI 0068).
- *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0* (IHI 0069).
- *Arm® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile* (DDI 0487).
- *Arm® GICv3 and GICv4 Software Overview* (DAI 0492).
- *Arm® CoreLink™ CMN-600 Coherent Mesh Network Technical Reference Manual* (100180).

The following confidential books are only available to licensees or require registration with Arm:

- *Arm® CoreLink™ GIC-600 Generic Interrupt Controller Configuration and Integration Manual* (100337).
- *Arm® Reliability, Availability, and Serviceability (RAS) Architecture Extension* (PRD03-PRDC-010953).
- *Arm® CoreLink™ ADB-400 AMBA® Domain Bridge User Guide* (DUI 0615).
- *Arm® CoreLink™ CMN-600 Coherent Mesh Network Configuration and Integration Manual* (100557).

**Other publications**

- JEDEC, *Standard Manufacturer's Identification Code*, JEP106.

# Feedback

## Feedback on this product

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

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

## Feedback on content

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

- The title *Arm CoreLink GIC-600 Generic Interrupt Controller Technical Reference Manual*.
- The number 100336_0103_00_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

**Note**

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

# Chapter 1
# **Introduction**

Read this for an introduction to the GIC-600 and its features.

It contains the following sections:


- *1.1 About the GIC-600* on page 1-12.
- *1.2 Components* on page 1-13.
- *1.3 Compliance* on page 1-18.
- *1.4 Features* on page 1-19.
- *1.5 Test features* on page 1-20.
- *1.6 Product documentation* on page 1-21.
- *1.7 Product revisions* on page 1-22.

## 1.1 About the GIC-600

The GIC-600 is a generic interrupt controller that handles interrupts from peripherals to the cores and between cores. The GIC-600 supports a distributed microarchitecture containing several individual blocks that are used to provide a flexible GIC implementation.

The GIC-600 supports the GICv3 architecture, see the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

The microarchitecture scales from a single core to coherent multichip environments containing up to 16 chips of up to 64 cores each.

**Note**

- This manual defines a *chip* as a SoC that is integrated with the GIC-600. A single-chip system has one SoC. A multichip system can have several SoCs that are connected externally, or a SoC comprising several SoCs connected inside a single physical package. In all cases, each SoC is integrated with the GIC-600.

All the GIC-600 blocks communicate through fully credited AXI4-Stream interface channels. This means that the interface exerts transient backpressure only on their **ic<xy>tready** signals, enabling packets to be routed over any free-flowing interconnect. Channels can be routed over dedicated AXI4-Stream buses, or over any available free-flowing transport layer in the system. A channel is described as free-flowing if all transactions on that channel complete without a non-transient dependency on any other transaction.

The GIC-600 includes build scripts that can create appropriate levels of hierarchy for any particular configuration. In small configurations, the distribution can be hidden and internally optimized.

## 1.2 Components

The GIC-600 comprises several significant blocks that work in combination to create a single architecturally compliant GICv3 implementation within the system. The GIC-600 top level can have one of several optional structures.

The GIC-600 consists of the following blocks:

**Distributor**

The Distributor is the hub of all the GIC communications and contains the functionality for all *Shared Peripheral Interrupts* (SPIs) and *Locality-specific Peripheral Interrupts* (LPIs). It is responsible for the entire GIC programmers model, except for the GITS_TRANSLATER register, which is hosted in the *Interrupt Translation Service* (ITS) block.

The Distributor also maintains the coherency of the SPI register space in multichip configurations.

——— **Note** ———

The LPI functionality for all cores on a chip is combined into a single cache in the Distributor.

———————————

See *2.1 Distributor* on page 2-24 and *3.1 Interrupt types* on page 3-49.

**Redistributor**

The Redistributor maintains the *Private Peripheral Interrupts* (PPIs) and *Software Generated Interrupts* (SGIs) for a particular set of cores. A Redistributor can scale from 1-64 cores and is best placed next to the processors that it is servicing to reduce wiring to the cores.

A Redistributor is also referred to as a PPI block.

The GICv3 architecture specifies a Redistributor address space containing two pages per core. The SGI page functionality is contained in the GIC-600 Redistributor. However, the command and control pages for all cores on a chip are contained in the Distributor.

The GIC-600 supports powering down the Redistributors and the associated cores.

See *2.2 Redistributor* on page 2-30 and *3.1 Interrupt types* on page 3-49.

**Interrupt Translation Service**

The ITS translates message-based interrupts, *Message-Signaled Interrupts* (MSI/MSIx), from an external *PCI Express* (PCIe) *Root Complex* (RC), or other sources. The ITS also manages LPIs during core power management.

The GIC-600 supports up to 16 ITS blocks per chip.

See *2.3 ITS* on page 2-34.

For more information about the ITS, see the *Arm® GICv3 and GICv4 Software Overview*.

**MSI-64 Encapsulator**

The MSI-64 Encapsulator is a small block that combines the *DeviceID* (DID), required by writes to the GITS_TRANSLATER register, into a single memory access.

See *2.4 MSI-64 Encapsulator* on page 2-39.

**SPI Collator**

The GIC-600 supports up to 960 SPIs that are spread across the system. The SPI Collator enables SPIs to be converted into messages remotely from the Distributor. This enables hierarchical clock-gating of the Distributor and the use of other more aggressive low-power states.

See *2.5 SPI Collator* on page 2-42.

**Wake Request**

The Wake Request contains all the architecturally defined **wake_request** signals for each core on the chip. It is a separate block that can be positioned remotely from the Distributor, such as next to a system control processor if necessary.

See *2.6 Wake Request* on page 2-44.

**GIC interconnect**

The GIC interconnect is a set of components that can be used for routing the AXI4-Stream interfaces between the different blocks.

See *2.7 Interconnect* on page 2-45.

**Top level**

The top level has no specific interfaces but combines the interfaces of other blocks within the clock or power domain to reduce the number of domain bridges. The GIC-600 build scripts enable you to build the GIC from a single combined block or a set of individual blocks that are interconnected using your own transport layer.

See *Figure 2-8 GIC top-level structure options* on page 2-47.

These blocks can be combined in different ways:

- In systems where there is an available free-flowing transport layer in place, existing buses can be used to route the GIC traffic.
- The GIC-600 includes a narrow, 16-bit, AXI4-Stream interconnect that can be used for routing internal traffic.

The following figure shows a GIC-600 with a free-flowing interconnect in an example system.

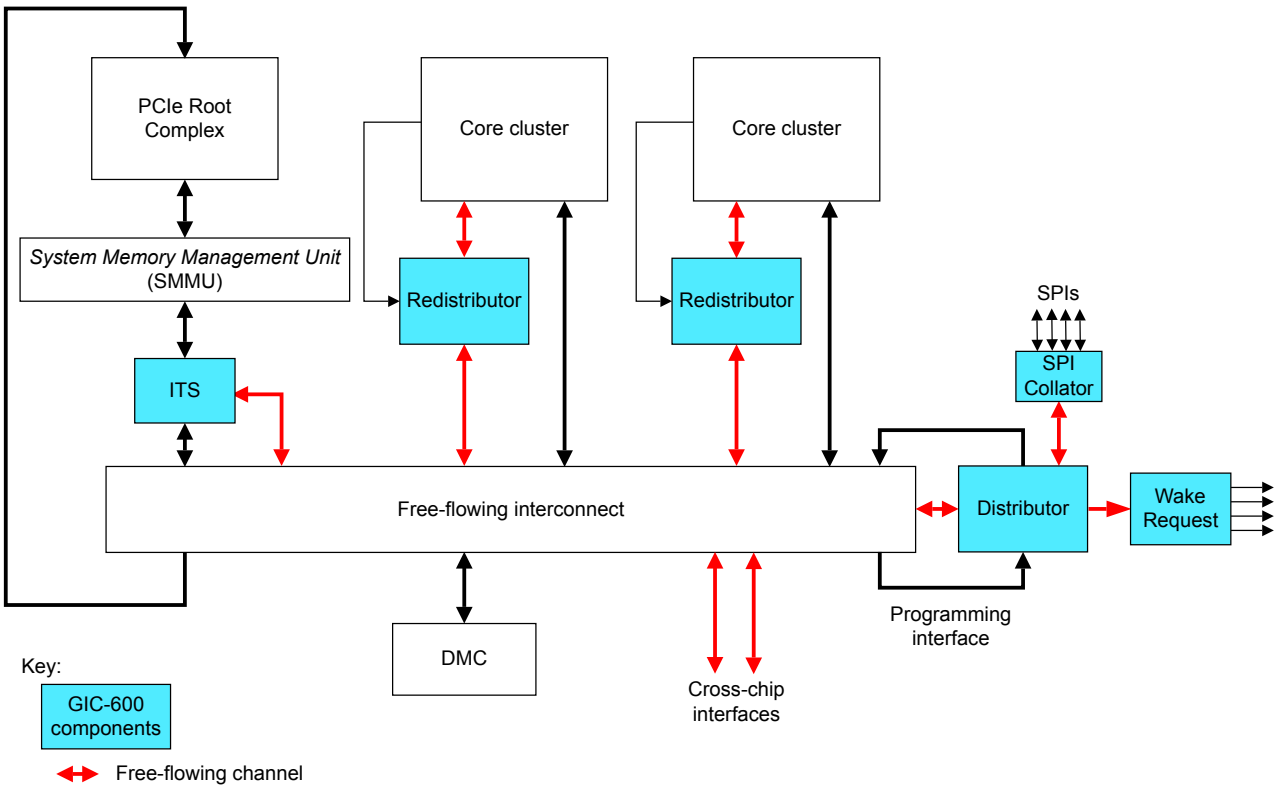


**Figure 1-1 GIC-600 with free-flowing interconnect in an example system**

**Note**

A free-flowing channel is clear to transmit a transaction that arrives at its destination without any non-transient dependencies on other transactions.

The following figure shows a GIC-600 with interconnect in an example system.

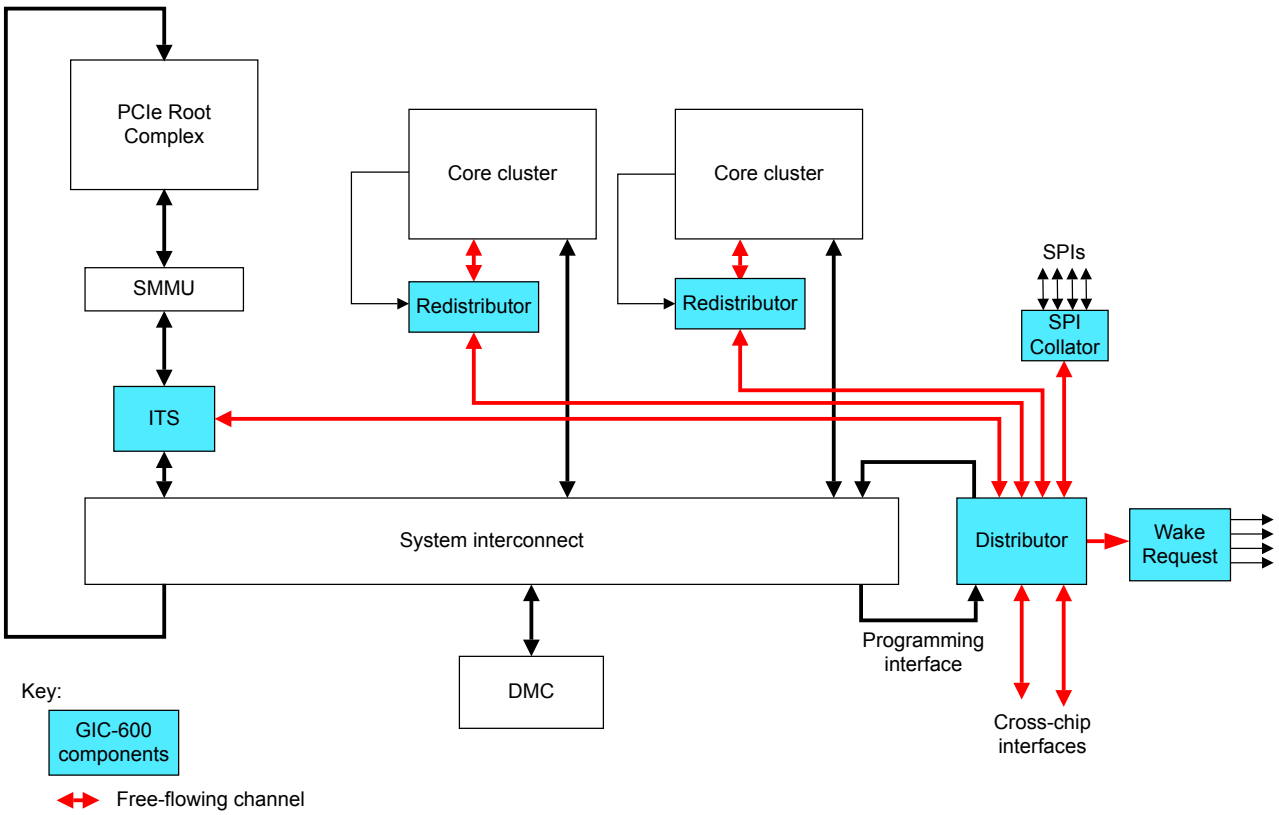


**Figure 1-2 GIC-600 with interconnect in an example system**

**Note**

Cross-chip interfaces enable communication between cores in a multichip configuration. Cross-chip is supported in product version r1p2 onwards.

The following figure shows a monolithic GIC-600 with interconnect in an example system.

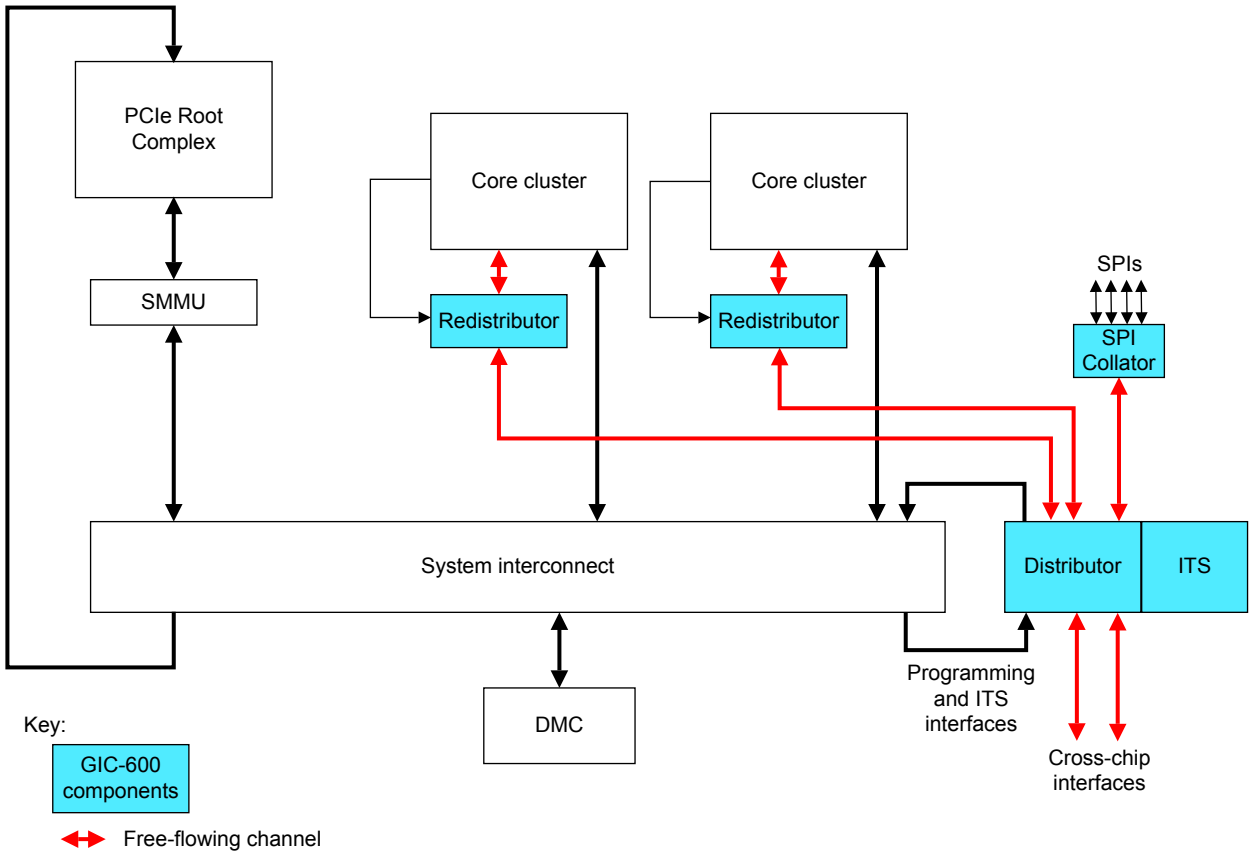


**Figure 1-3 Monolithic GIC-600 with interconnect in an example system**

**Note**

If the GIC supports LPIs, there must be free-flowing access to main memory. This requirement is irrespective of the interconnect that is used for routing the AXI4-Stream interfaces. For more information, see the *Arm® CoreLink™ GIC-600 Generic Interrupt Controller Configuration and Integration Manual*.

The GIC-600 supports cores that implement only the ARMv8.0-A architecture, and later versions such as v8.2-A. The cores must also support the GIC CPU interface with the standard GIC AXI4-Stream protocol interface. The GIC-600 implements version 3.0 of the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

## 1.3 Compliance

The GIC-600 interfaces are compliant with Arm specifications and protocols.

The GIC-600 is compliant with:

- The AMBA AXI4-Stream protocol. See the *Arm® AMBA® AXI and ACE Protocol Specification*.
- Version 3.0 of the Arm GIC architecture specification. See the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

The GIC Stream protocol is based on the following specifications:

- The AMBA AXI4-Stream protocol. See the *Arm® AMBA® 4 AXI4-Stream Protocol Specification*.
- The GIC Stream Protocol Interface. See the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

***Related information***

*Arm® AMBA® AXI and ACE Protocol Specification*

*Arm® Generic Interrupt Controller Architecture Specification, version 3.0 and version 4.0*

## 1.4 Features

The GIC-600 provides interrupt services and masking, registers and programming, interrupt grouping, security, performance monitoring, and error correction.

### Interrupt services and masking:

- Support for the following interrupt types:
  - Up to 56000 LPIs. A peripheral generates these interrupts by writing to a memory-mapped register in the GIC-600. See *2.1.7 Distributor configuration* on page 2-28.
  - Up to 960 SPIs in groups of 32. See *2.1.7 Distributor configuration* on page 2-28.
  - Up to 16 PPIs that are independent for each core and can be programmed to support either edge-triggered or level-sensitive interrupts. See *2.2.6 Redistributor configuration* on page 2-32.
  - Up to 16 SGIs that are generated through the GIC CPU interface of a core.
- Up to 16 ITS modules that provide device isolation and ID translation for message-based interrupts and enable virtual machines to program devices directly.
- Interrupt masking and prioritization with 32 priority levels, five bits per interrupt.

### Registers and programming:

- Flexible affinity routing, using the *Multiprocessor Identification Register* (MPIDR) addresses, including support for all four affinity levels.
- Single ACE-Lite slave port on each chip for programming of all *GIC Distributor* (GICD) registers, GIC *Interrupt Translation Service* (GITS) registers, and *GIC Redistributor* (GICR) registers. Each ITS has an optional ACE-Lite slave port for programming the GITS_TRANSLATER register.
- Coherent view of SPI register data across multiple chips.

### Security:

- A global *Disable Security* (DS) bit. This bit enables support for systems without security.
- The following interrupt groups allow interrupts to target different Exception levels:
  - Group 0.
  - Non-secure Group 1.
  - Secure Group 1.

  See *3.8 Security* on page 3-62 for more information about security and groupings.

**Note**

For more information about Exception levels, see the *Arm® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

### Performance monitoring:

- *Performance Monitoring Unit* (PMU) counters with snapshot functionality.

### Error correction:

- ARMv8.2 *Reliability Accessibility Serviceability* (RAS) architecture-compliant error reporting for:
  - Software access errors.
  - ITS command and translation errors.
  - *Error Correcting Code* (ECC) errors.

## 1.5 Test features

The GIC-600 provides *Design for Test* (DFT) signals for test mode.

***Related reference***

*A.1 Common control signals* on page Appx-A-185

## 1.6 Product documentation

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

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

The GIC-600 documentation is as follows:

**Technical Reference Manual**

The TRM describes the functionality and the effects of functional options on the behavior of the GIC-600. It is required at all stages of the design flow. The choices that are made in the design flow can mean that some behaviors that the TRM describes are not relevant. If you are programming the GIC-600, contact:

- The implementer to determine:
  - The build configuration of the implementation.
  - What integration, if any, was performed before implementing the GIC-600.
- The integrator to determine the signal configuration of the device that you use.

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

**Configuration and Integration Manual**

The CIM describes:

- The available build configuration options.
- How to configure the *Register Transfer Level* (RTL) with the build configuration options.
- How to integrate the GIC-600 into a SoC.
- How to implement the GIC-600 into your design.
- The processes to validate the configured design.

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

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

## 1.7 Product revisions

This section describes the differences in functionality between product revisions.

**r0p0** First release.

**r0p0-r0p1** Functional changes are:
- Bug fixes.

**r0p1-r0p2** Functional changes are:
- Bug fixes.

**r0p2-r0p3** Functional changes are:
- Bug fixes.

**r0p3-r1p2** Functional changes are:
- Multichip functionality.

**r1p2-r1p3** Functional changes are:
- Bug fixes.

# Chapter 2
# Components and configuration

Read this for a description of the major components of the GIC-600.

It contains the following sections:


- *2.1 Distributor* on page 2-24.
- *2.2 Redistributor* on page 2-30.
- *2.3 ITS* on page 2-34.
- *2.4 MSI-64 Encapsulator* on page 2-39.
- *2.5 SPI Collator* on page 2-42.
- *2.6 Wake Request* on page 2-44.
- *2.7 Interconnect* on page 2-45.
- *2.8 Hierarchy* on page 2-46.

## 2.1 Distributor

The Distributor is the main communication point between all GIC-600 blocks. It performs SPI management and LPI caching, and all communications with other blocks and chips.

The following figure shows the Distributor and its interfaces.

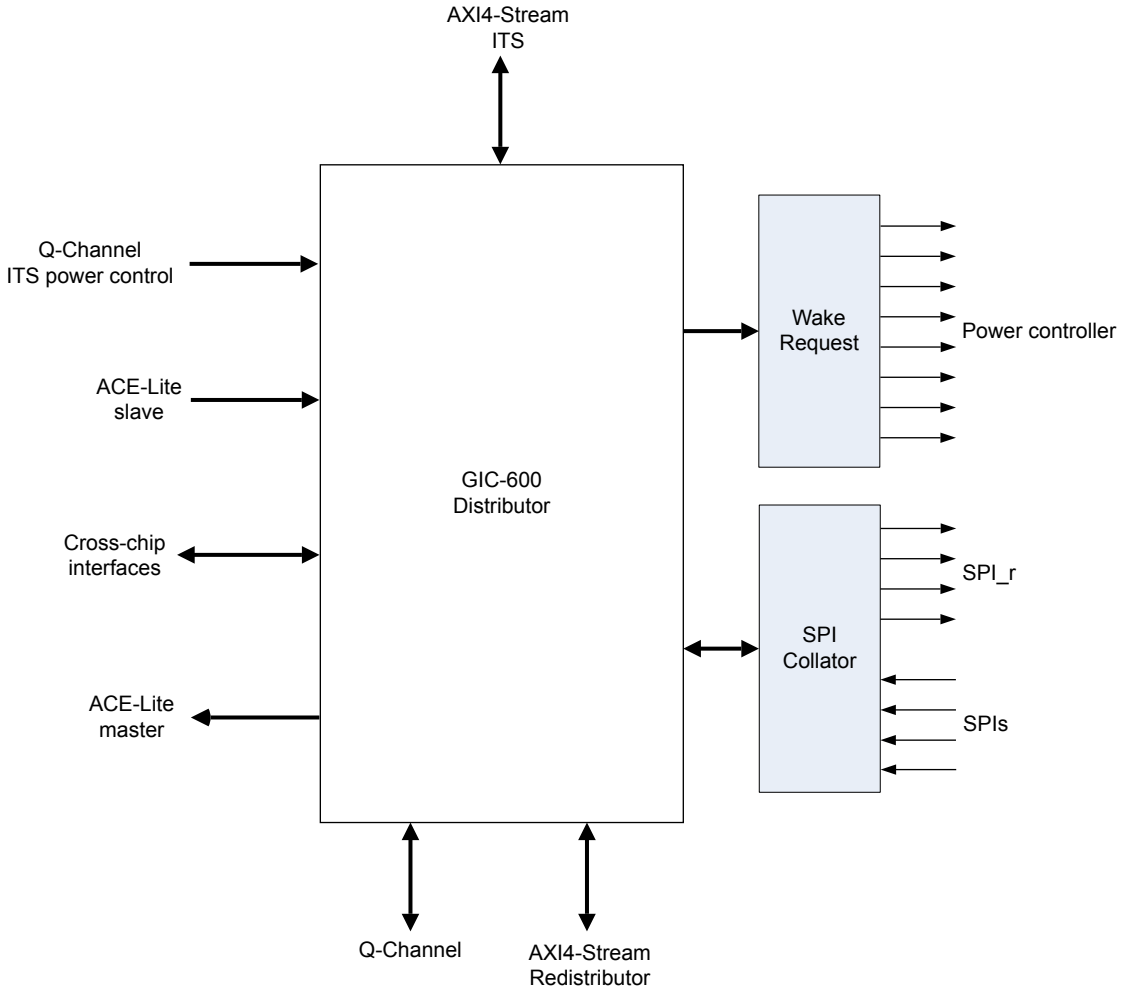


**Figure 2-1 GIC-600 Distributor**

The Distributor is the main hub of the GIC and it implements most of the GICv3 architecture including:

- Programming, forwarding, and prioritization of SPIs, see *3.1.3 SPIs* on page 3-49.
- Caching and forwarding of LPIs, see *3.1.4 LPIs* on page 3-50.
- SGI routing and forwarding.
- Register programming of all registers apart from GITS_TRANSLATER.
- Power control of cores and Redistributor blocks.

This section contains the following subsections:

- *2.1.1 Distributor AXI4-Stream interfaces* on page 2-25.
- *2.1.2 Distributor ACE-Lite slave interface* on page 2-25.
- *2.1.3 Distributor ACE-Lite master interface* on page 2-26.
- *2.1.4 Distributor Q-Channels* on page 2-27.
- *2.1.5 P-Channel* on page 2-27.
- *2.1.6 Distributor miscellaneous signals* on page 2-28.
- *2.1.7 Distributor configuration* on page 2-28.

### 2.1.1 Distributor AXI4-Stream interfaces

The GIC-600 uses AXI4-Stream interfaces to communicate between blocks.

These interfaces are fully credited.

**Note**

- **ic<xy>tready xy** can be **cd**, **dc**, **pd**, **dp**, **id**, **di**, **rd**, **dr**, or **dw**.
- Packets must not be reordered between endpoints, for example, between the Distributor and a single Redistributor block, irrespective of the interconnect that is used. Packets must never be interleaved.

For information about AXI4-Stream signals, see the *Arm® AMBA® 4 AXI4-Stream Protocol Specification*.

The following table lists the AXI4-Stream input interfaces.

**Table 2-1 AXI4-Stream input interface descriptions**

| Bus | Destination | Width | ic<xy>dtid |
|---|---|---|---|
| ICID | ITS to Distributor | 16-bit or 64-bit | ITS number |
| ICPD | Redistributor to Distributor | 16-bit, 32-bit, or 64-bit | Redistributor number |
| ICCD | SPI Collator to Distributor | 16-bit | 0 |
| ICRD | Remote Chip to Distributor | 64-bit | 0 |

The following table lists the AXI4-Stream output interfaces.

**Table 2-2 AXI4-Stream output interface descriptions**

| Bus | Destination | Width | ic<xy>dtdest |
|---|---|---|---|
| ICDI | Distributor to ITS | 16-bit or 64-bit | ITS number |
| ICDP | Distributor to Redistributor | 16-bit, 32-bit, or 64-bit | Redistributor number |
| ICDC | Distributor to SPI Collator | 16-bit | 0 |
| ICDR | Distributor to Remote Chip | 64-bit | Programmed value |
| ICDW | Distributor to Wake Request block | 16-bit | - |

Each bus has an associated **ic<xy>twakeup** signal that requests wakeup through the **qactive** signals when the Distributor, or destination block, is hierarchically clock-gated through the Q-Channel. The **ic<xy>twakeup** input signal must be driven from a cleanly registered version of the **ic<xy>tvalid** signal to prevent spurious wakeups caused by signal glitches.

For information about the Distributor Q-Channels, see *2.1.4 Distributor Q-Channels* on page 2-27.

### 2.1.2 Distributor ACE-Lite slave interface

The AMBA AXI4 ACE-Lite slave port on the GIC-600 Distributor provides access to the entire register map except for the GITS_TRANSLATER register. The interface supports either 64-bit, 128-bit, or 256-bit data widths.

The GIC-600 only accepts single beat accesses of the sizes for each register that are shown in the Programmers model, see *Chapter 4 Programmers model* on page 4-103. All other accesses are rejected and given either an OKAY or SLVERR response that is based on the GICT_ERR0CTLR.UE bit.

When the GIC-600 is a monolithic configuration, the Distributor and ITS both share an ACE-Lite slave port, and the DeviceID for the ITS translation is taken from **awuser_s[DEVICE_ID_WIDTH+2:3]**. See *2.3 ITS* on page 2-34, for more information about the ITS.

**Note**

The **a<x>user_s[2:0]** signals are not used and must be tied LOW.

The following table shows the acceptance capabilities of the Distributor ACE-Lite slave interface.

**Table 2-3 Distributor ACE-Lite slave interface acceptance capabilities**

| Attribute | Capability |
|---|---|
| Combined acceptance capability | 3 |
| Read acceptance capability | 2 |
| Read data reorder depth | 1 |
| Write acceptance capability | 2 |

The GIC-600 uses **a<x>cache_s**, **a<x>domain_s**, and **a<x>bar_s** signals to detect cache maintenance operations and barrier transactions that are responded to in a protocol-compliant manner but are otherwise ignored. The GIC-600 also ignores other cachability, shareability, and protection settings, except for security signal **a<x>prot_s[1]**.

If you are connecting to an AXI3 or AXI4 port, signals **a<x>domain>_s**, **a<x>bar_s** and, for AXI3, **a<x>len[7:4]** must all be tied LOW.

The GIC-600 has a separate **awakeup_s** signal to force the GIC to wakeup when it is hierarchically clock-gated through the Q-Channel. The **awakeup_s** signal must be connected to a cleanly registered version of **(awvalid_s | arvalid_s)** to ensure that the GIC does not request to be woken up due to incoming signal glitches.

The GIC-600 address map can have several pages. The number of pages depends on your configuration. See *4.1.1 Register map pages* on page 4-104.

You must set up the system address map so that each core accesses the GICD page on its local chip at the same address. All other pages must be globally accessible, although access of pages on a remote chip by a core is expected to be rare.

In most configurations, the GIC-600 ignores address bits above ceiling[$\log_2$(page_count)] + 15. For example, a configuration that uses 11 pages ignores address bits above 19, and any address bits of the form `0xXXXXX00000` is accepted to access the GICD page of the memory map. However, in monolithic configurations, where the Distributor and ITS share the ACE-Lite slave port, there are two address tie-offs that specify the full page address of the GICD and GITS_TRANSLATER pages. The page address comprises address bits[x:16]. For example, if the GICD page is at 32-bit address `0xFFFF0000`, the tie-off is 16-bit `0xFFFF`. See *2.1.6 Distributor miscellaneous signals* on page 2-28 for information about the Distributor miscellaneous signals.

***Related reference***

*4.1.1 Register map pages* on page 4-104

### 2.1.3 Distributor ACE-Lite master interface

The GICD uses the AMBA AXI4 ACE-Lite master interface to access LPI Property and Pending tables. If LPIs are not supported, then this interface is not present.

The interface can be configured to be 64-bit, 128-bit, or 256-bit wide.

The following table shows the issuing capabilities of the Distributor ACE-Lite master interface.

**Table 2-4 Distributor ACE-Lite master interface issuing capabilities**

| Attribute | Capability | | |
|---|---|---|---|
| | Read | Write | Combined |
| 256-bit aligned read and writes to any Pending table | 3 | 3 | 3 |
| 8-bit read and writes to any Pending table | 1 | 1 | 1 |
| 256-bit aligned reads to the Property table | 1 | 0 | 1 |
| 8-bit reads to the Property table | 4 | 0 | 4 |

Each transaction uses a unique transaction ID, and properties come from either the GICR_PROPBASER or GICR_PENDBASER registers according to the destination. There is one copy of the attribute fields for all GICR_PROPBASER registers and another for all GICR_PENDBASER registers, so software must program these registers to a consistent value in all Redistributors.

The ACE-Lite master port cannot issue barriers or *Cache Maintenance Operations* (CMOs). However, it can issue shareable, *ReadOnce* and *WriteUnique*, transactions if programmed to do so.

See *3.12 Memory access and attributes* on page 3-69 for more information.

The **a<x>user_m** signal outputs the GICR_TYPER.ProcessorNumber of the core that is associated with each transaction, but it can be ignored and it is not necessary to route it anywhere else.

**Note**

If the Distributor and ITS both share the same ACE-Lite master port, the issuing capabilities are cumulative.

### 2.1.4 Distributor Q-Channels

There is a single Q-Channel for clock gating the GIC-600 Distributor. The Q-Channel interface denies access if the Distributor block is busy processing interrupts.

The Distributor also has a separate Q-Channel that enables power control for each configured ITS. The Q-Channel reads GITS_CTLR.Quiescent to check whether the ITS is fully disabled. If the Quiescent bit is set, the Q-Channel **qacceptn_its** signal is asserted, and the GIC guarantees that the bus to the relevant ITS is idle in both directions and that the ITS can be powered down. To perform wake-on-LPI functionality, you can use GITS_FCTLR.PWE to disable the bus while the ITS is still active and able to translate interrupts. If the bus is disabled, the system must re-enable the bus based on the status of the ITS **qactive** signal, that is, when **qactive_its** is not asserted.

**Note**

The **qreqn*** signals are synchronized internally, and can be driven asynchronously. See *A.2 Power control signals* on page Appx-A-187.

***Related information***

*AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces*

### 2.1.5 P-Channel

The P-Channel is used for power control of the GIC-600 Distributor.

The P-Channel is present only in multichip configurations. It is used to safely isolate the Distributor from other chips to allow the save and restore of its register states.

***Related concepts***

*3.6 Power management* on page 3-58

### 2.1.6 Distributor miscellaneous signals

The Distributor generates or processes several signals, such as tie-offs, interrupts, and handshakes.

The following table shows the Distributor miscellaneous signals.

**Table 2-5 Distributor miscellaneous signals**

| Signal | Direction | Description |
|---|---|---|
| **chip_id** | Input | Tie off this signal to identify the chip in the system. Only present if there is more than one chip in the system. |
| **fault_int** | Output | These fault handling and error reporting interrupts are defined in *Arm® Reliability, Availability, and Serviceability (RAS) Architecture Extension*. The GIC-600 can deliver these interrupts internally but the outputs are provided for any other device such as a system control processor that does not receive normal interrupts from the GIC.<br>See *3.16 Reliability, Accessibility, and Serviceability* on page 3-75. |
| **err_int** | Output | |
| **pmu_int** | Output | The PMU counter overflow interrupt. This interrupt can be routed internally but is provided as an external output to trigger an external agent to service the GIC, for example, to read out the PMU counter snapshot registers.<br>See *3.15 Performance Monitoring Unit* on page 3-73. |
| **sample_req**, | Input | This 4-phase handshake provides a hardware mechanism to snapshot the PMU counters and has the same effect as writing to the GICP_CAPR register. |
| **sample_ack** | Output | |
| **gict_allow_ns**, | Input | From reset, these tie-off signals control whether Non-secure software can access the GICT RAS, and the GICP PMU, pages. Secure software can override the values at any time. |
| **gicp_allow_ns** | Input | |
| **gicd_page_offset** | Input | This tie-off signal is used to set the page address bits[x:16] of the GICD page. Only present in monolithic configurations. |

### 2.1.7 Distributor configuration

You can configure several options that relate to the operation of the Distributor block.

**Table 2-6 Configurable options for the Distributor**

| Feature | Range of options |
|---|---|
| Number of chips | 1-16 |
| Affinity level that is used for chip selection | 2, 3 |
| Affinity0 width | 0 |
| Affinity1 width | 0-8 |
| Affinity2 width | 0-8 |
| Affinity3 width | 0-8 |
| LPI support | True, False |
| LPI cache size (entries / 2) | 8, 16, 32, 64, 128, 256, 512 |
| Number of ITS | 0-16 |
| Number of Redistributors on chip | 1-64 |
| Number of message-based SPIs permitted in system | 32-960, in blocks of 32 |

**Table 2-6 Configurable options for the Distributor (continued)**

| Feature | Range of options |
|---|---|
| Number of SPI wires on chip for wire-based SPIs | 0-960 |
| Security support | Options include:<br>• Security support programmable. Resets to support security.<br>• Security support always present.<br>• Security support not present.<br>**Note**<br>See *Security model* in the *Arm® GICv3 and GICv4 Software Overview* for information about the implications of setting Security support to not present. |

See the *Arm® CoreLink™ GIC-600 Generic Interrupt Controller Configuration and Integration Manual* for more information.

***Related information***

*Arm® GICv3 and GICv4 Software Overview*

## 2.2 Redistributor

The Redistributor is responsible for PPIs and SGIs that are associated with its related cluster or group of cores. A Redistributor is also referred to as a PPI block.

The following figure shows the Redistributor block.

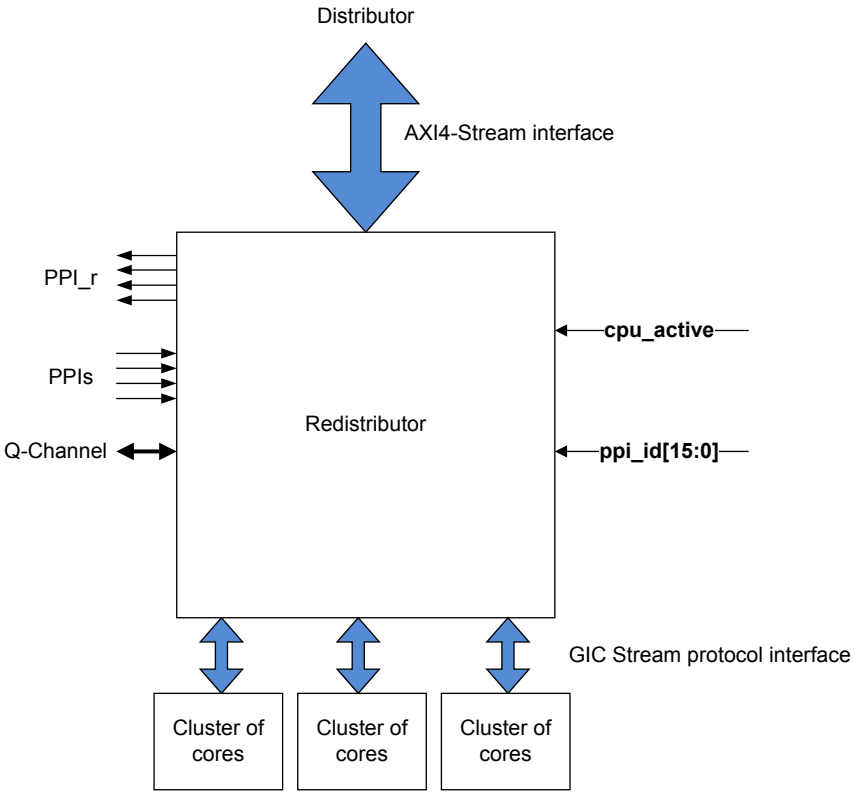


**Figure 2-2 GIC-600 Redistributor**

The Redistributor performs the following functions:

- Maintaining the SGI and PPI programming.
- Monitoring, and if necessary, synchronizing the PPI wires.
- Prioritizing SGIs, PPIs, and any other interrupts that are sent from the Distributor, and forwarding them to the core.
- Maintaining the GIC Stream protocol and communicating with the cluster.

There can be multiple Redistributors in a configuration and they can be sized to match your system. For example, if you have two clusters of eight cores, then you can have one Redistributor positioned next to each cluster. You can use a Redistributor for each cluster to reduce the PPI wiring and enable the Redistributor to be powered down with the cores for extra power savings. Alternatively, for a small system, combining all cores into one Redistributor block might be the best solution. See *Configuration options* in the *Arm® CoreLink™ GIC-600 Generic Interrupt Controller Configuration and Integration Manual* for more information.

**Note**

The Redistributor (GICR) registers are programmed through the Distributor ACE-Lite slave port. The Distributor also contains the architectural LPI functionality.

This section contains the following subsections:

- *2.2.1 Redistributor AXI4-Stream interface* on page 2-31.
- *2.2.2 Redistributor GIC Stream protocol interface* on page 2-31.

- *2.2.3 Redistributor Q-Channel* on page 2-31.
- *2.2.4 Redistributor PPI signals* on page 2-31.
- *2.2.5 Redistributor miscellaneous input signals* on page 2-32.
- *2.2.6 Redistributor configuration* on page 2-32.

### 2.2.1 Redistributor AXI4-Stream interface

Each Redistributor has an upstream and downstream AXI4-Stream port for communicating with the Distributor. This interface is either 16-bit or 64-bit wide and uses a fully credited protocol.

See *1.1 About the GIC-600* on page 1-12.

### 2.2.2 Redistributor GIC Stream protocol interface

The GIC-600 uses the GIC Stream protocol interface to send interrupts to the core and receive notifications when the core activates interrupts. The GIC Stream protocol interface has a pair of 16-bit wide AXI4-Stream interfaces, one upstream interface, and one downstream interface.

The GIC Stream protocol interface, also referred to as the GIC Stream interface, uses the GIC Stream protocol to pass interrupts and responses to the CPU interface inside each core.

**Table 2-7 GIC Stream protocol interface signals**

| Signal name | Description |
|---|---|
| **iri** | Prefix which identifies the names of the downstream interface signals. These signals are sent by the GIC Stream master. On this interface, the Redistributor is the master and the CPU interface is the slave. |
| **icc** | Prefix which identifies the names of the upstream interface signals. These signals are sent by the GIC Stream slave. On this interface, the CPU interface is the master and the Redistributor is the slave. |
| **iritdest** | The GIC Stream master interface uses this signal to direct packets to one core within the cluster. |
| **icctid** | The GIC Stream slave interface uses this signal to determine which core within the cluster sent a packet. |

Both the **iritdest** and **icctid** can support up to 16 cores that use packed binary encoding, as opposed to one-hot encoding. They can also be divided down using an AXI4-Stream crossbar to support clusters of an arbitrary number of cores from 1-16.

The necessary crossbar is generated as part of the render process based on the number of GIC-Stream buses that are specified for each Redistributor.

### 2.2.3 Redistributor Q-Channel

The Redistributor has a single Q-Channel input that is used to ensure that the Redistributor can be safely clock-gated hierarchically.

If the Redistributor is busy, actively processing interrupts or sending messages up or downstream, the Q-Channel denies a quiescence request, **qreqn**, by asserting the **qdeny** signal. For more information, see the *AMBA® Low Power Interface Specification, Arm® Q-Channel and P-Channel Interfaces*.

**Note**

The **qreqn** input is synchronized inside the Redistributor.

The **qactive** signal is connected to the PPI wires directly, and must be considered as an asynchronous output.

***Related reference***

*A.2 Power control signals* on page Appx-A-187

### 2.2.4 Redistributor PPI signals

GIC-600 supports 8, 12, or 16 PPIs, and synchronized output return wires, for each core. The number of PPIs and return wires must be the same for all cores sharing a Redistributor.

Level-sensitive PPI signals are active-LOW by default, as with previous Arm GIC implementations. However, individual PPI signals can be inverted and synchronized using parameters `GIC600_<config_name>_PPI<ppi_id>_<cpu_number>_<ppi_number>_<INV/SYNC>`.

Every wire has a corresponding wire from after the synchronizer or capture flop. These can be used to create pulse extenders for edge-triggered interrupts that cross clock domains.

**Note**

If you plan to use edge-triggered PPIs and the Q-Channel to clock gate the Redistributor hierarchically, you must use pulse extenders to ensure that interrupts are not missed while the clock is restarted.

For information about the purpose of each PPI used by the core in your system, refer to the Technical Reference Manual for the core.

### 2.2.5 Redistributor miscellaneous input signals

The Redistributor receives signals that identify the status of each core. It also has a tie-off signal that provides the Redistributor with a unique identifier.

**Table 2-8 Redistributor miscellaneous input signals**

| Signal | Direction | Description |
|---|---|---|
| **cpu_active** | Input | This signal indicates if the core is active and not in a low-power state such as retention. The GIC can decide to target only active cores for 1 of N SPIs. See *3.15 Performance Monitoring Unit* on page 3-73.<br>**Note**<br>**cpu_active** is not synchronized into the Redistributor. If **cpu_active** is driven from a different domain, it must be synchronized externally. |
| **ppi_id[15:0]** | Input | This tie-off signal provides the Redistributor with a unique identifier that is used primarily to ensure that the GIC is correctly integrated into the system. |

***Related concepts***

*3.6 Power management* on page 3-58

### 2.2.6 Redistributor configuration

You can configure several options that relate to the operation of the Redistributor block.

**Table 2-9 Configurable options for the Redistributor**

| Feature | Range of options |
|---|---|
| Number of cores downstream | 1-64 |
| PPIs per core | 8, 12, 16 |
| ECC support[a] | True, False |
| Bus data width | 16 or 32 |
| GIC Stream bus structure | Flexible buses and domains |

See the *Arm® CoreLink™ GIC-600 Generic Interrupt Controller Configuration and Integration Manual* for more information.

***Related information***

*Arm® GICv3 and GICv4 Software Overview*

[a] See *3.16 Reliability, Accessibility, and Serviceability* on page 3-75 for more information.

***Related concepts***

*3.1 Interrupt types* on page 3-49

## 2.3 ITS

The ITS provides a software mechanism for translating message-based interrupts into LPIs. The ITS is supported optionally in configurations that support LPIs.

A peripheral generates an LPI by writing to the GITS_TRANSLATER in the ITS. The write provides the ITS with the following information:

- *EventID* (VID). A value written to GITS_TRANSLATER. The EventID identifies which interrupt the peripheral is sending. Each interrupt source is identified by an *Interrupt Identifier* (INTID). The EventID might be the same as the INTID, or it might be translated by the ITS into the INTID.
- *DeviceID* (DID). The DeviceID is a unique identifier that identifies the peripheral.

The following figure shows the ITS block.

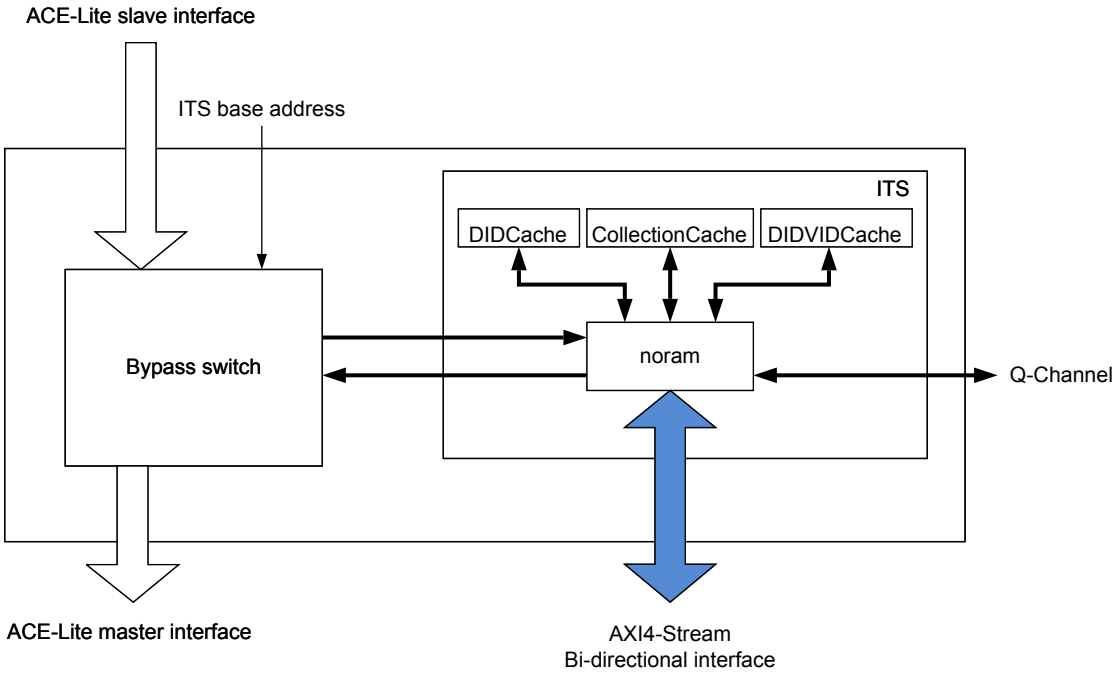


**Figure 2-3 ITS block**

The ITS is an implementation of the GICv3 Interrupt Translation Service as described in the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*. The ITS translates MSI requests to the required LPI and target. It also has a set of commands for managing LPIs for core power management and load balancing.

A main use of the ITS is the translation of MSI/MSIx messages from a PCIe *Root Complex* (RC). To complete the translation, the ITS must be supplied with a DeviceID that is derived from the PCIe RequestorID. To reduce the distance that the DeviceID is transferred and to enable better compartmentalization between RCs, the ITS is best placed next to the RC. To ease integration, the ITS has an optional bypass switch as shown in the ITS block diagram. If the bypass switch is not configured, the ACE-Lite slave and master ports connect to the ITS directly. See *2.3.1 ITS ACE-Lite slave interface* on page 2-35 and *2.3.2 ITS ACE-Lite master interface* on page 2-36.

**Note**

In accordance with PCIe dependency rules, read responses on a PCIe Root Complex slave port must be ordered against completion of posted writes on a Root Complex master port. If the connected interconnect does not guarantee free-flowing accesses of reads to memory under all circumstances, then you must not use the configuration shown in *Figure 2-3 ITS block* on page 2-34. This also applies to the CoreLink CMN-600 Coherent Mesh Network if the I/O coherent Requesting Node (RN-I) is able to access the same I/O Home Node (HN-I) that provides access to the PCIe Root Complex slave port. If the

ITS is configured without a bypass switch, then a bypass switch can still be used to provide ITS access to memory through a different interconnect port, without merging the master ports.

---

See *3.10 Interrupt translation service (ITS)* on page 3-65 for more information.

The following figure provides an example of the ITS integration process.

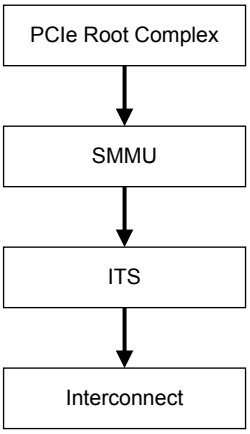


**Figure 2-4 ITS integration**

An ITS can be placed anywhere in the system so that it is seen by devices that want to send MSIs. However, the system is responsible for ensuring that the DeviceID reaching each ITS is not spoofed by rogue software using either **a<x>user** signals or MSI-64. See *2.4 MSI-64 Encapsulator* on page 2-39.

If the ITS is placed downstream of an ACE interconnect, care must be taken to avoid system deadlock. See *Chapter 3, Key integration points* in the *Arm® CoreLink™ GIC-600 Generic Interrupt Controller Configuration and Integration Manual*.

See *3.10 Interrupt translation service (ITS)* on page 3-65 for more information about each inner block.

This section contains the following subsections:

- *2.3.1 ITS ACE-Lite slave interface* on page 2-35.
- *2.3.2 ITS ACE-Lite master interface* on page 2-36.
- *2.3.3 ITS AXI4-Stream interface* on page 2-37.
- *2.3.4 ITS Q-Channel* on page 2-37.
- *2.3.5 ITS miscellaneous signals* on page 2-38.
- *2.3.6 ITS configuration* on page 2-38.

### 2.3.1 ITS ACE-Lite slave interface

The ITS AMBA AXI4 ACE-Lite slave interface has a configurable width of 64 bits, 128 bits, or 256 bits. The slave, master, and address data widths must match.

The ITS ACE-Lite slave port contains only the GITS_TRANSLATER register. See the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0* for more information.

If the bypass switch configuration option is selected, the port accepts all ACE-Lite traffic, and filters accesses to the ITS based on an address match set by the ITS base address tie-off **its_page_offset [ADDR_WIDTH:16]**. Without the bypass switch, the upper bits of the address, 16 and above, are ignored, and the system address decoders must ensure that only relevant ITS writes arrive at the ITS.

The ACE-Lite slave interface ignores all **a<x>snoop**, **a<x>cache**, **a<x>domain**, and **a<x>prot** information other than to filter CMOs and Barriers to ensure that it replies in a protocol-compliant manner.

To generate an LPI, the ITS requires the DeviceID of the issuing master. For PCIe, the DeviceID is derived from the RequestorID.

The GIC-600 supports two different methods for deriving the DeviceID:

- When using the MSI-64 config parameter, the write to GITS_TRANSLATER is converted to 64-bit accesses at an unmapped system address and the DeviceID is transferred in the upper 32 bits of the access. In this case, only burst length 1, 64-bit ACE-Lite writes are accepted.
- When not using MSI-64, the DeviceID is transported on the **AWUSER** bus with the address (AW) phase of the register access. In this case, burst length 1, 32-bit or 16-bit writes are accepted.

The DeviceID must be transferred using a method that cannot be spoofed by malicious software.

**Note**

These two modes cannot be mixed on a single ITS.

If the bypass switch is configured, it includes a transaction tracker that ensures PCIe ordering requirements are met. There are two options that are based on the `full_bypass_tracker` tag:

0 = A simple scheme is used, which ensures that all previous transactions sent downstream have completed before forwarding an MSI to the ITS, and conversely, that the ITS has accepted all MSIs before continuing to send traffic downstream.

1 = A more complex scheme, which allows continuous downstream traffic including interleaved MSIs, unless the buffer slots become full. There are two buffers, `bypass_max_outstanding`, which specifies the number of concurrent downstream transactions allowed and `bypass_interrupt_count`, which specifies the number of concurrent MSIs that can be waiting for their prerequisite transactions to complete.

**Note**

The ITS slave port contains only write-only registers, therefore the read channel always uses the simple transaction tracker.

If the bypass switch is configured, the slave and master ports must both have the same data width and the same address width.

If the Distributor and ITS both share the ACE-Lite slave port, the port properties match those of the Distributor ACE-Lite slave port, which are described in *2.1.2 Distributor ACE-Lite slave interface* on page 2-25.

The following table shows the acceptance capabilities of the ITS ACE-Lite slave interface.

**Table 2-10 ITS ACE-Lite slave interface acceptance capabilities**

| Attribute | With bypass switch | Without bypass switch |
|---|---|---|
| Combined acceptance capability | Read acceptance capability + Write acceptance capability | 3 |
| Read acceptance capability | 128 | 1 |
| Read data reorder depth | 128 | 1 |
| Write acceptance capability | 128 or `bypass_max_outstanding` | 2 |

The ITS ACE-Lite slave interface has an associated **awakeup** signal. To ensure that incoming traffic wakes the ITS correctly when it is clock gated hierarchically through the Q-Channel, **awakeup** must be driven from a registered version of **awvalid** and **arvalid**. To prevent spurious wake events, ensure that the **awakeup** signal is registered cleanly.

### 2.3.2 ITS ACE-Lite master interface

The ITS AMBA AXI4 ACE-Lite master interface has a configurable width of 64 bits, 128 bits, or 256 bits. If the bypass switch is not included, the ID width is 4 bits, otherwise the ID width is one more than the ID width of the corresponding input channel.

The ACE-Lite master port issues accesses to the ITS private tables and Command queue. If the bypass switch is configured, the port also forwards transactions from the slave interface. The ACE-Lite bus can issue I/O coherent transactions, therefore you can place these tables in shared memory if required. Placing the Command queue in shared memory avoids having to flush the cache before executing ITS commands.

**Note**

- When heavily loaded, the ITS creates a necessary dependency between writes on its slave port and reads on its master port. You must ensure that any writes that back up to the slave port do not prevent the free-flow of both reads and writes to the memory.
- In an ACE system, you must ensure that the write channel from any core cache that could be snooped is not blocked by accesses to the ITS slave port. If the write channel is blocked, and the snoop is prevented from completing its task, a potential deadlock can result.

Arm recommends that if you place the ITS downstream of an ACE interconnect, then you must not place tables in shareable memory.

The ITS can issue the following transaction types:

- 256-bit aligned read to the Command queue.
- 64-bit aligned read and write to the Device table.
- 32-bit aligned read and write to the *Interrupt Translation Table* (ITT).
- 16-bit aligned read and write to the Collection table.
- If the bypass switch is configured, any bypassed transactions from the slave port.

ITS issued transactions output the DeviceID on the **a<x>user_** signals. The DeviceID is used for information and does not have to be routed anywhere if it is not required. If the bypass switch is included, ITS issued transactions are identified by a value of 0 on **a<x>id[0]**.

**Note**

The ITS issues only one outstanding transaction per ID. This gives a maximum of one outstanding write and five outstanding reads, excluding any transactions from the slave port. If this port is combined with the Distributor ACE-Lite master port, some of these properties are changed. See *Figure 2-8 GIC top-level structure options* on page 2-47 for more information.

***Related information***

*Arm® GICv3 and GICv4 Software Overview*

### 2.3.3 ITS AXI4-Stream interface

The ITS AXI4-Stream interface is a bi-directional AXI4-Stream interface of either 16-bit or 64-bit width for communication between the ITS and the GIC Distributor components on the same chip.

Arm expects a typical distributed system to be 16 bits wide. When a pre-existing wide interconnect is used, the 64-bit option allows messages to be efficiently packed.

The interface is fully credited so all messages can be accepted without dependency on any other ports.

### 2.3.4 ITS Q-Channel

The ITS has a Q-Channel interface which controls requests from an external clock gating source.

If the ITS is busy, the Q-Channel interface asserts the **qdeny** signal to deny an external request to gate its clock. When an external request occurs, the interface requests a wakeup by asserting **qactive**.

The **qreqn** input is synchronized to the ITS. See *A.2 Power control signals* on page Appx-A-187.

***Related reference***

*A.2 Power control signals* on page Appx-A-187

### 2.3.5 ITS miscellaneous signals

The ITS generates or processes several signals, such as an ID tie-off, and the ITS page offset.

**Table 2-11 ITS miscellaneous signals**

| Signal | Direction | Description |
|---|---|---|
| **its_page_offset [ADDR_WIDTH-16:0]** | Input | Modifies the address map to ensure only writes to the correct location trigger MSI requests. Only present when the bypass switch is configured.<br>Specifies the 64K page address that includes the GITS_TRANSLATER register address, and is matched against **axaddr[ADDR_WIDTH-16:1]**. |
| **its_id** | Input | This is an ID tie-off. It must be tied to the **ic<x>dtdest** value used to read the ITS on the AXI4-Stream interface. This ID value feeds into the GITS_CFGID register and is used to check that the GIC system is correctly interconnected. If top-level stitching is used, which creates a hierarchical level from the other components, this signal is not visible. |
| **its_transr_page_offset** | Input | This tie-off signal is used to set the page address of the GITS_TRANSLATER register. Only present in monolithic configurations. |

### 2.3.6 ITS configuration

You can configure several options that relate to the operation of the ITS block.

**Table 2-12 Configurable options for the ITS**

| Feature | Range of options |
|---|---|
| DeviceID[b] | 1-16 |
| EventID size[b] | 1-32 |
| Bypass_ports | 1 or 0 |
| ACE-Lite data width | 64, 128, or 256 |
| MSI-64 support | True or False |
| ACE-Lite read ID width | 1-32 |
| ACE-Lite write ID width | 1-32 |
| ECC support[c] | True or False |
| DID cache size | 2, 4, 8, 16, 32, 64, or 128 |
| DID and VID cache size | 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, or 2048 |
| Collection cache size | 2, 4, 8, 16, 32, 64, 128, 256, or 512 |
| Domain[d] | Any legal domain identifier |

***Related information***

*Arm® GICv3 and GICv4 Software Overview*

b See *2.3 ITS* on page 2-34 for more information.
c See *3.16 Reliability, Accessibility, and Serviceability* on page 3-75 for more information.
d See *Figure 2-8 GIC top-level structure options* on page 2-47.

## 2.4 MSI-64 Encapsulator

The MSI-64 Encapsulator reduces system wiring by combining the DeviceID onto the Data bus for writes to the GITS_TRANSLATER register.

The following figure shows an overview of the MSI-64 Encapsulator process.

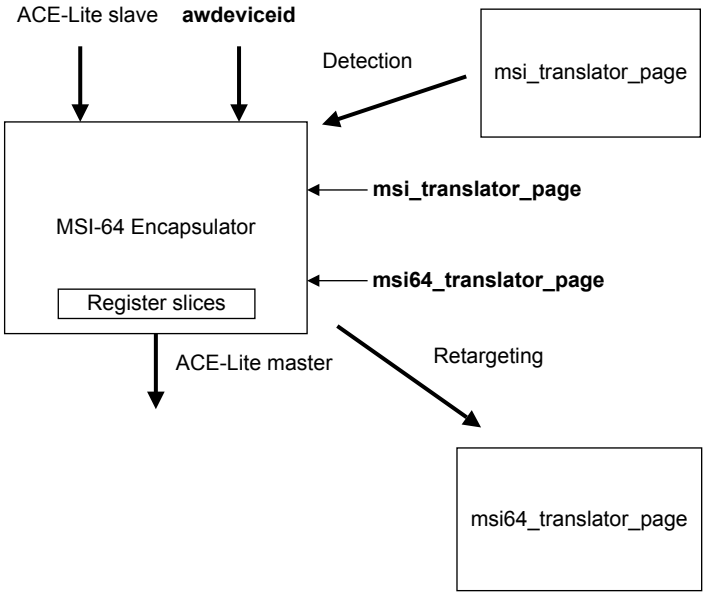


**Figure 2-5 MSI-64 Encapsulator**

The MSI-64 Encapsulator detects translations that are targeted at the target page address of the GITS_TRANSLATER register, set by the **msi_translator_page** tie-off. It then converts accesses to 64-bit writes with the **awdeviceid** in the upper 32 bits of the data and retargets them to the **msi64_translator_page**. This avoids having to use wires to transfer a DeviceID to the GITS_TRANSLATER register for translation.

See *3.13 MSI-64* on page 3-71 for more information.

This section contains the following subsections:

- *2.4.1 MSI-64 ACE-Lite interfaces* on page 2-39.
- *2.4.2 MSI-64 miscellaneous signals* on page 2-40.
- *2.4.3 MSI-64 Encapsulator configuration* on page 2-40.

### 2.4.1 MSI-64 ACE-Lite interfaces

The MSI-64 Encapsulator has an ACE-Lite slave interface and an ACE-Lite master interface.

**MSI-64 ACE-Lite slave interface with awdeviceid**
This interface is a full ACE-Lite slave port with an extra **awdeviceid** input signal, which is valid, and must remain stable with **awvalid**.

**MSI-64 ACE-Lite master interface**
This interface is a full ACE-Lite master port.

The following table shows the transaction acceptance capabilities of both slave and master ports.

**Table 2-13 Transaction acceptance**

| Transaction type | Maximum number of transactions allowed |
|---|---|
| Read | 128 |
| Write | 128 |
| Combined | 256 |

Any leading **wdata** is registered and held until the **awaddr** signal arrives. These signals are described in *A.5 ACE interface signals* on page Appx-A-190.

**Note**

- The MSI-64 Encapsulator requires a data bus that has a width of 64 bits or greater.
- The ACE-Lite master port never issues more than two addresses before signal **wlast** is asserted.

### 2.4.2 MSI-64 miscellaneous signals

The MSI-64 receives target address signals for the GITS_TRANSLATER register, and an ACE-Lite sideband signal.

**Table 2-14 MSI-64 miscellaneous signals**

| Signal | Direction | Description |
|---|---|---|
| **msi_translator_page** | Input | The target page address of the GITS_TRANSLATER register. The MSI-64 Encapsulator does not support a **msi_transalator_page** value of 0. |
| **msi64_translator_page** | Input | The target address of the 64-bit GITS_TRANSLATER register. |
| **awdeviceid** | Input | The ACE-Lite AW sideband signal that reports the DeviceID for writes to GITS_TRANSLATER. The value is ignored for non-MSI writes. |

### 2.4.3 MSI-64 Encapsulator configuration

The MSI-64 Encapsulator does not have any configurable options at design time. However, if this block is generated in your RTL design, it has several parameter options that you can configure.

The MSI-64 Encapsulator is generated as part of any GIC configuration that includes an MSI-64 enabled ITS.

The following table shows the parameter options for the MSI-64 Encapsulator that you can configure at build time.

**Table 2-15 Configurable options for the MSI-64 Encapsulator**

| RTL parameter | Function | Range of options |
|---|---|---|
| `DATA_WIDTH` | Specifies the width of data signals **rdata** and **wdata** | 64, 128, 256 |
| `ADDR_WIDTH` | Specifies the width of address signals **araddr** and **awaddr** | 17-48 |
| `AWUSER_WIDTH` | Specifies the width of signal **awuser** | 1-128 |
| `ARUSER_WIDTH` | Specifies the width of signal **aruser** | 1-128 |
| `RUSER_WIDTH` | Specifies the width of signal **ruser** | 1-128 |
| `WUSER_WIDTH` | Specifies the width of signal **wuser** | 1-128 |
| `BUSER_WIDTH` | Specifies the width of signal **buser** | 1-128 |

**Table 2-15 Configurable options for the MSI-64 Encapsulator (continued)**

| RTL parameter | Function | Range of options |
|---|---|---|
| DEVICEID_WIDTH | Specifies the width of the DeviceID. | 1-20 |
| WID_WIDTH | Specifies the width of signal **wid** | 1-32 |
| RID_WIDTH | Specifies the width of signal **rid** | 1-32 |
| FWD_REG_TYPE | Register slice type on forward AW, AR, and W channels. | 0 = None<br>1 = Reverse<br>2 = Forward<br>3 = Full |
| REV_REG_TYPE | Register slice type on B and R channels | 0 = None<br>1 = Reverse<br>2 = Forward<br>3 = Full |

## 2.5 SPI Collator

The SPI Collator converts SPI wires into messages to be sent to the Distributor.

The following figure shows the SPI Collator block.

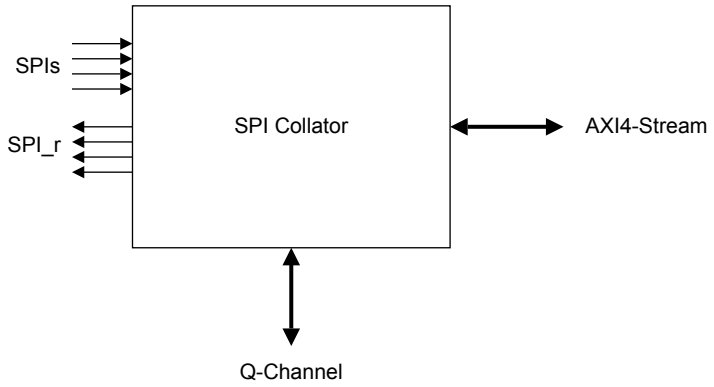


**Figure 2-6 SPI Collator**

Individual SPIs can be synchronized into the SPI Collator, or the SPI Collator can be placed in the same clock domain as the interrupt sources and the messages that are synchronized into the Distributor.

Placing the SPI Collator in a clock domain that is always on and is remote from the GIC Distributor enables more aggressive power saving because the GIC Distributor can be clock gated hierarchically.

This section contains the following subsections:

- *2.5.1 SPI Collator AXI4-Stream interface* on page 2-42.
- *2.5.2 SPI Collator wires* on page 2-42.
- *2.5.3 SPI Collator power Q-Channel* on page 2-42.
- *2.5.4 SPI Collator configuration* on page 2-43.
- *2.5.5 SPI Collator clock Q-Channel* on page 2-43.

### 2.5.1 SPI Collator AXI4-Stream interface

The AXI4-Stream interface enables communication between the SPI Collator and the Distributor.

The AXI4-Stream ports apply only transient backpressure to the AXI4-Stream interface, which enables packets to be routed over any free-flowing interconnect.

### 2.5.2 SPI Collator wires

The SPI Collator wires can be extended to create other functions.

By default, the asserted level of a SPI is active-HIGH, as with previous Arm GIC implementations. However, each SPI can be either inverted or synchronized, or both, using the parameters `gic600_<config_name>_SPI_INV[n]` and `gic600_<config_name>_SPI_SYNC[n]`, where:

- `SPI_INV[n]` == 1 = inverter enabled.
- `SPI_SYNC[n]` == 1 = synchronizer enabled.
- `[n]` = SPI_ID - 32.

Each SPI Collator wire has corresponding wires after the synchronizer or capture flop that can be used to create pulse extenders for edge-triggered interrupts that cross clock domains.

### 2.5.3 SPI Collator power Q-Channel

The SPI Collator has a power Q-Channel interface that accepts requests from an external source, such as the system power controller.

When signal **qactive_col** is LOW, it indicates that all SPIs to the SPI Collator are in their normal state of either 0 (edge-triggered, or active HIGH), or 1 (edge-triggered, or active LOW), and therefore all messages are sent to the Distributor.

If **qactive_col** is HIGH, the SPI Collator rejects any attempt to enter a low-power mode.

If **qreqn_col** is LOW and is accepted, the SPI Collator enters low-power mode and the AXI4-Stream channels to the Distributor are flushed out to ensure that there are no messages in progress. When accepted, you can reset the SPI Collator safely without having to also reset the Distributor. You can also reset the Distributor, but you must first complete the instructions that are described in the subsections of section *3.6 Power management* on page 3-58 before the Distributor can be powered down.

**Note**

- When the SPI Collator and Distributor are both in the same domain, the power Q-Channel interface is redundant and can be tied off.
- In low-power mode, it is only safe to stop the Collator clock if all edge-triggered interrupts into the SPI Collator are pulse extended to ensure that edges are not missed.

### 2.5.4 SPI Collator configuration

You can configure several options that relate to the operation of the SPI Collator block.

**Table 2-16 Configurable options for the SPI Collator**

| Feature | Description |
|---|---|
| NUM_SPIS | The number of wires. |
| SPI_INV | A wide vector of one bit for each SPI indicating the interrupt must be inverted. |
| SPI_SYNC | A wide vector of one bit for each SPI indicating the interrupt must be synchronized. |

***Related information***

*Arm® GICv3 and GICv4 Software Overview*

### 2.5.5 SPI Collator clock Q-Channel

The SPI Collator has a clock Q-Channel interface that accepts requests from an external clock gating source, such as the system clock controller.

When signal **qactive_clk_col** is LOW, it indicates that all SPI toggles and level transitions have been passed to the Distributor, and that the SPI Collator does not require the clock.

If **qactive_clk_col** is HIGH, the SPI Collator rejects any attempt to enter a low-power mode.

If **qreqn_clk_col** is LOW and is accepted, the SPI Collator enters low-power mode and no new messages are sent to the Distributor until it enters low-power mode. If any interrupt line changes state, **qactive_clk_col** is asserted.

**Note**

In low-power mode, it is only safe to stop the Collator clock if all edge-triggered interrupts into the SPI Collator are pulse extended to ensure that edges are not missed.

## 2.6 Wake Request

The Wake Request block converts AXI4-Stream wake requests into one **wake_request** wire for each core. Each **wake_request** connects to the system power controller.

The following figure shows the Wake Request block.


**Figure 2-7 Wake Request**

The level of the asserted **wake_request** signal drops only when the Distributor leaves reset, or when the core is woken and the GICR_WAKER.ProcessorSleep bit is cleared to indicate that it is able to communicate with the GIC. The GIC supports a wake_request block reset only when the Distributor is also reset.

This section contains the following subsections:

- *2.6.1 Wake Request AXI4-Stream interface* on page 2-44.
- *2.6.2 Wake Request miscellaneous signals* on page 2-44.
- *2.6.3 Wake Request configuration* on page 2-44.

### 2.6.1 Wake Request AXI4-Stream interface

The AXI4-Stream interface enables the Wake Request block to communicate with the Distributor.

The AXI4-Stream interface does not exert back-pressure.

### 2.6.2 Wake Request miscellaneous signals

The Wake Request block generates the **wake_request[<num_cpus - 1:0>]** signal.

**Table 2-17 Wake Request miscellaneous signals**

| Signal | Description |
|---|---|
| **wake_request[<num_cpus - 1:0>]** | This output signal indicates to the power controller that an interrupt is targeting this core and that the core must be woken. When asserted, the **wake_request** is sticky unless the Distributor is put into the gated state. |

### 2.6.3 Wake Request configuration

The configuration of the Wake Request block is based on the number of cores in the system. There are no other options to configure.

***Related information***

*Arm® GICv3 and GICv4 Software Overview*

## 2.7 Interconnect

The GIC-600 uses AXI4-Stream interfaces for communication between some blocks.

These blocks are:

- Distributor to and from ITS.
- Distributor to and from Redistributors.
- Distributor to Distributor for cross-chip communications.
- Distributor to and from the SPI Collator.
- Distributor to and from the Wake Request block.

All these interfaces use fully credited schemes where all messages are guaranteed to be accepted without dependency on any other port.

Apart from the cross-chip communications, GIC-600 provides an AXI4-Stream interconnect for transporting messages. However, messages can be sent over an existing interconnect provided the interconnect is free-flowing.

### 2.7.1 Interconnect configuration

The internal interconnect is configured automatically in accordance with the number of cores and ITS blocks in the system. The configuration produces a balanced tree structure with minimum clock domain crossings.

## 2.8 Hierarchy

There are three structure options that can be selected using the `structure` configuration tag.

`wrap`: This provides the lowest level of structure, and wraps the following blocks:

- The Redistributor is wrapped with interconnect components between the Redistributor and the cores. The components wrapped at this level are shown within the blue dashed lines in the following figure. If the core is in a different clock domain, in accordance with the domain tags, then half of the ADB-400 domain bridge is included in a stitched file called `gic600_ppi_wrap_<n>_<usrcfg>.v`.
- If a bypass switch is selected as shown in *Figure 2-3 ITS block* on page 2-34, the ITS block is wrapped in a file called `gic600_its_wrap_<n>_<usrcfg>.v`.
- If the GIC is configured to share ACE-Lite ports between the ITS and GICD (configuration tag `monolithic` = 1), the ITS and GICD are stitched together in a file called `gic600_gicd_wrap_<usrcfg>.v`.

`domain`: All blocks and wrapped components that are in the same domain are stitched together in a file called `gic600_domain_<name>_<usrcfg>.v` and includes ADB-400 domain bridges and collated low-power interfaces. Blocks and components at this level are shown within the red dashed lines in the following figure.

`full`: All domains are stitched together to create a single top-level GIC-600 file called `gic600_<usrcfg>.v`.

The following figure shows the top-level options.

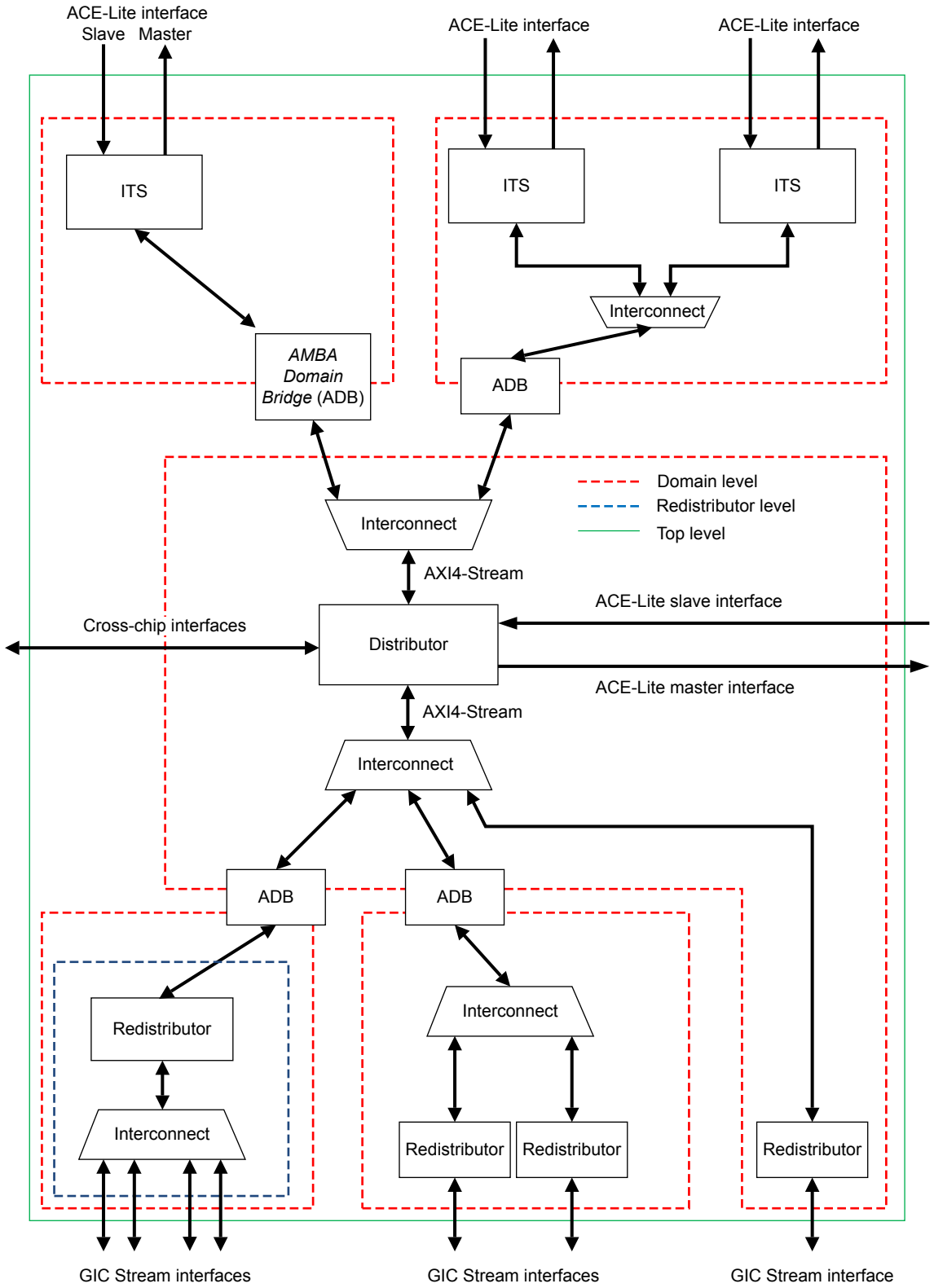


**Figure 2-8 GIC top-level structure options**

# Chapter 3
# **Operation**

Read this for an operational description of the GIC-600.

It contains the following sections:


- *3.1 Interrupt types* on page 3-49.
- *3.2 Interrupt groups* on page 3-52.
- *3.3 Physical interrupt signals (PPIs and SPIs)* on page 3-53.
- *3.4 Affinity routing and assignment* on page 3-54.
- *3.5 1 of N SPI interrupt selection* on page 3-56.
- *3.6 Power management* on page 3-58.
- *3.7 Getting started* on page 3-61.
- *3.8 Security* on page 3-62.
- *3.9 Backwards compatibility* on page 3-64.
- *3.10 Interrupt translation service (ITS)* on page 3-65.
- *3.11 LPI caching* on page 3-68.
- *3.12 Memory access and attributes* on page 3-69.
- *3.13 MSI-64* on page 3-71.
- *3.14 RAM* on page 3-72.
- *3.15 Performance Monitoring Unit* on page 3-73.
- *3.16 Reliability, Accessibility, and Serviceability* on page 3-75.
- *3.17 Multichip operation* on page 3-97.

## 3.1 Interrupt types

The GIC-600 manages SPIs, SGIs, PPIs, and LPIs.

This section contains the following subsections:


- *3.1.1 SGIs* on page 3-49.
- *3.1.2 PPIs* on page 3-49.
- *3.1.3 SPIs* on page 3-49.
- *3.1.4 LPIs* on page 3-50.
- *3.1.5 Choosing between LPIs and SPIs* on page 3-50.


### 3.1.1 SGIs

Software Generated Interrupts are inter-processor interrupts, that is, interrupts generated from one core and sent to other cores.

Each core in the system processes an SGI independently of the other cores. The priority of an SGI, and other settings, are also independent for each core.

SGIs are generated by writing to system registers in the CPU interface of the core that generates the interrupt. SGI signals are edge triggered.

Up to 16 SGIs can be recorded for each target core, where each SGI has a different INTID in the range ID0-ID15.

### 3.1.2 PPIs

A Private Peripheral Interrupt identifies an interrupt source, such as a timer, that is private to the core, and which is independent of the same source for another core. PPIs are typically used for peripherals that are tightly coupled to a particular core.

Interrupts that are connected to the PPI inputs associated with one core, are only sent to that core. Each core processes a PPI independently of other cores. The settings of a PPI are also independent for each core.

A PPI is unique to one core. However, the PPIs to other cores can have the same INTID. Up to 16 PPIs can be recorded for each target core, where each PPI has a different INTID in the range ID16-ID31.

PPI signals are active-LOW level-sensitive by default. However, you can set a PPI signal to be either level-sensitive or edge-triggered using GICR_ICFGR1, see the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

The GIC-600 provides an option, through parameters, to include one or both a synchronizer and inverter on each PPI interrupt wire. See *2.2.4 Redistributor PPI signals* on page 2-31 for more information.

For information about the purpose of each PPI used by the processor core in your system, refer to the processor Technical Reference Manual.

### 3.1.3 SPIs

A Shared Peripheral Interrupt is generated by a peripheral that is accessible across the whole system, such as a USB receiver, and which can be routed to several cores. 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 a SPI on one core activates the SPI for all cores. That is, the GIC-600 allows at most one core to activate a SPI. The settings for each SPI are also shared between all cores.

SPIs are generated either by wire inputs or by writes to the ACE-Lite slave programming interface. The GIC-600 can support up to 960 SPIs corresponding to the external **spi** signal on the SPI Collator. The number of SPIs available depends on the implemented configuration. The permitted values are ID32-ID960, 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. The GIC-600 provides an option, through a parameter, to include one or both a synchronizer and inverter on each SPI interrupt wire.

The GIC-600 uses the SPI Collator to convert wire-based interrupts into messages to reduce system wiring, and to allow more aggressive clock gating of the GIC to reduce power consumption. See *2.5 SPI Collator* on page 2-42 for more information.

SPIs are programmed through the GICD register address space, which is spread coherently across all configured chips to provide a single view to the *Operating System* (OS).

You can add a pending state to a valid SPI using GICD_SETSPI_NSR or GICD_SETSPI_SR, see the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

### 3.1.4 LPIs

*Locality-specific Peripheral Interrupts* (LPIs) are always message-based, and can be from a peripheral, or from a PCIe root complex.

An LPI targets only one core. LPIs are generated when the peripheral writes to the ITS. The ITS contains the registers to control the generation and maintenance of LPIs. The ITS provides INTID translation, allowing peripherals to be owned directly by a virtual machine if an SMMU is also present for those peripherals.

**Note**

- The ITS enables interrupts to be translated to the ID space of the hypervisor instead of directly to a virtual machine.
- Instead of using an ITS, registers can be used to configure the GIC-600 to generate and control LPIs. For more information, see *GICR_SETLPIR register* in the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

### 3.1.5 Choosing between LPIs and SPIs

Message-based interrupts can be either LPIs or SPIs.

The decision to use an LPI or SPI for an interrupt can be made by software, and depends on whether there are spare SPIs and if the GIC-600 has ITS support. This can be achieved by either making the peripheral write to a different GIC-600 address, or by changing the address translation for the interrupt write in the SMMU. Changing only the SMMU is possible because the registers for Non-secure message-based interrupts, GICD_SETSPI_NSR and GITS_TRANSLATER, or GICR_SETLPIR for configurations without LPI support, are at the same address offset in different pages.

The following factors can help you to decide which interrupt type is most appropriate:

- Only the ITS provides INTID translation, therefore LPIs are preferable for peripherals that are owned by a virtual machine. This is because the hypervisor can let the virtual machine program the peripheral directly, and the ITS convert the IDs of interrupts 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 are able to target all cores, which means that the GIC-600 attempts to automatically balance the interrupt load to cores that are active but not handling other interrupts.
- The GIC-600 can provide a greater number of LPIs than SPIs.
- You might decide not to include LPI support in a small system where the features of the ITS are not required and there are few message-based interrupts.
- SPIs usually have a better worst-case interrupt latency than LPIs. This is because SPIs have all their settings stored internally to the GIC-600, 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.

***Related information***

*Arm® GICv3 and GICv4 Software Overview*

## 3.2 Interrupt groups

The GIC-600 configures the interrupts that it receives into one of three groups. Each group determines the security status of an interrupt and how it is routed.

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

These registers 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 security state for interrupts in that group, depending on the Exception level of the core.
- 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 the GIC-600 is set to security disabled, the meaning and number of interrupt groups is affected.

You can specify GIC-600 security by setting the configuration parameter `ds_value` to either 0, 1 or P, where 0 = Security enabled (fixed), 1 = Security disabled (fixed), P = Security is programmable by software during the boot sequence using GICD_CTLR.DS.

See the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0* for more information.

***Related information***

*Arm® GICv3 and GICv4 Software Overview*

## 3.3 Physical interrupt signals (PPIs and SPIs)

The GIC-600 supports two types of physical interrupt signal.

The two types of physical interrupt signal are:

**Level-sensitive**
: The interrupt is pending while the interrupt input is asserted. As with previous Arm GICs, PPIs are active-LOW, whereas SPIs are active-HIGH by default. However, you can change these default settings, see *3.1 Interrupt types* on page 3-49 for more information.

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

To set the correct settings for the system, you must program the GICD_ICFGRn and GICR_ICFGR1 registers.

The GIC-600 provides optional synchronizers on every interrupt wire input and also return signals, to enable pulse extenders when sending edge-triggered interrupts across domain boundaries, see *2.5.2 SPI Collator wires* on page 2-42.

***Related concepts***

*2.2.4 Redistributor PPI signals* on page 2-31

*2.5.2 SPI Collator wires* on page 2-42

***Related information***

*Arm® GICv3 and GICv4 Software Overview*

*Arm® Generic Interrupt Controller Architecture Specification, version 3.0 and version 4.0*

## 3.4 Affinity routing and assignment

The GIC-600 uses affinity routing, a hierarchical scheme, to identify connected cores and for routing interrupts to specific cores.

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

- For 32-bit ARMv8 processors, the MPIDR defines three levels of affinity, with an implicit affinity level 3 value of 0.
- For 64-bit ARMv8 processors, the MPIDR defines four levels of affinity.

**Note**

The GIC-600 regards each hardware thread of a processor that supports multiple hardware threads as a single independent core.

The affinity of a core is represented by four 8-bitfields using dot-decimal notation, <Aff3>.<Aff2>.<Aff1>.<Aff0>, where Affn is a value for affinity level n. An example of an identification for a specific core would be 0.255.0.15.

The affinity scheme matches the format of the MPIDR_EL1 register in ARMv8-A. System designers must ensure that the ID reported by the core of the MPIDR_EL1 register matches how the core is connected to the interrupt controller.

The GIC-600 allows fully flexible allocation of MPIDR. However, it has two built-in default assignments that are based on the `aff0_thread` configuration parameter, see the *Arm® CoreLink™ GIC-600 Generic Interrupt Controller Configuration and Integration Manual*.

- When `aff0_thread` == 1, the four fields are mapped to 0.<cluster>.<core>.<thread>.
- When `aff0_thread` == 0, the four fields are mapped to 0.0.<cluster>.<core>.

The following figure shows the affinity hierarchical structure.

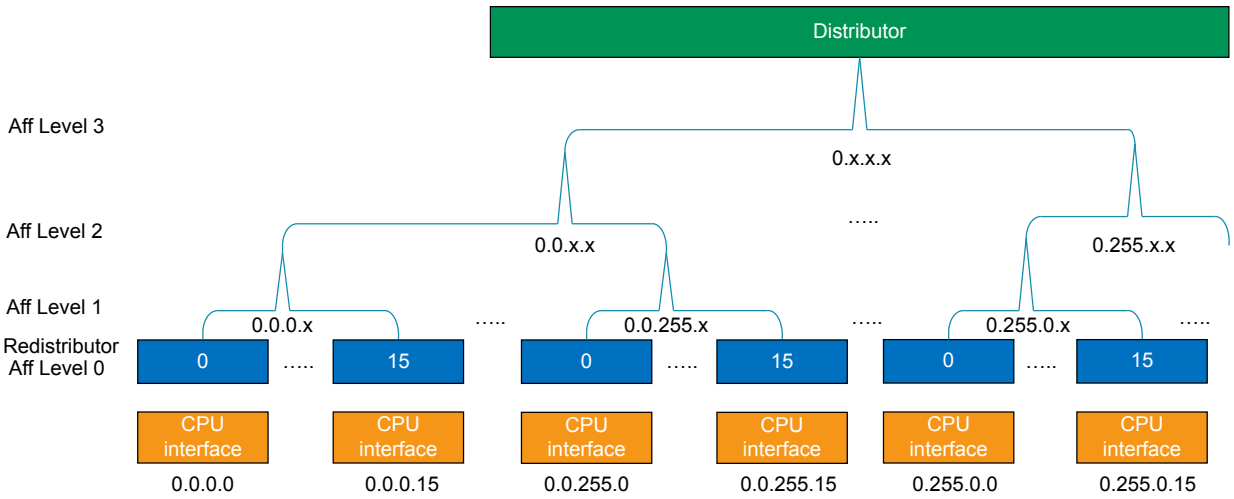


**Figure 3-1 Affinity routing**

There can be up to 256 nodes at level 3, with each node able to host 256 child level 2 nodes. Similarly each level 2 node can host 256 level 1 nodes. However, level 1 nodes can only host 16 child level 0 nodes.

For more information about affinity routing, see the *Arm® GICv3 and GICv4 Software Overview*, and the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

***Related information***

*Arm® Generic Interrupt Controller Architecture Specification, version 3.0 and version 4.0*

## 3.5 1 of N SPI interrupt selection

The GIC-600 supports 1 of N selection of SPI interrupts.

When the relevant GICD_IROUTERn.Interrupt_Routing_Mode == 1, the GIC selects the appropriate core for a SPI.

When GICD_IROUTERn.Interrupt_Routing_Mode == 0, the SPI is routed to the core specified by the remaining fields of GICD_IROUTERn.

The selections that the GIC-600 makes can be controlled or influenced by several 1 of N features:

**cpu_active**

A **cpu_active** signal is an input to a Redistributor that corresponds to a particular core. It indicates to the GIC that a core is in a transparent low-power state, such as retention, and that it must be selected as a target for a SPI if there are no other options possible.
Typically, a power controller or power control logic generates the **cpu_active** signal. If this signal is not available in the system, the input must be tied HIGH.

——— **Note** ———

The **cpu_active** provides an indication only, it cannot stop selection of the core or stop the GIC sending messages to the core.

———————————

**GICR_CTLR.DPGxx (Disabled Processor Group)**

Setting a DPG bit prevents 1 of N interrupts of a particular group being sent to that core. Any interrupts that have not reached a core at the time of the change are recalled and reprioritized by the GIC. For information about the DPG bits, see *GICR_CTLR, Redistributor Control Register* in the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

**Processor and GICD Group enables and GICR_WAKER.ProcessorSleep**

A 1 of N interrupt is not sent to a core if the core is asleep as indicated by GICR_WAKER.ProcessorSleep, or the interrupt group is disabled by either the processor, or the GICD_CTLR group.

**Interrupt class**

This is an implementation-defined feature that the GIC-600 provides. Each core can be assigned to either class 0 or class 1 by writing to the relevant GICR_CLASS register. A SPI, programmed as 1 of N, by GICD_IROUTERn.Interrupt_Routing_Mode, can be programmed to target either class 0, class 1, or both classes by the GICD_ICLARn register. By default, all 1 of N SPIs can go to both classes, so the interrupt class feature is disabled by default. The system can use this partitioning for any purpose, for example in a big.LITTLE™ system, all the big cores can be in class 1 and little cores in class 0, allowing 1 of N SPIs to be partitioned according to the amount of processing they require.

**GICD_CTLR.E1NWF**

The GICD_CTLR register E1NWF bit controls whether the GIC-600 wakes a core if there are no other possible targets for a 1 of N SPI.

The GIC tries to wake the minimum of cores possible and only wakes a core if there is no other possible target awake that is able to accept the 1 of N interrupt. To do this, the GIC uses the GICR.DPG and GICR_CLASS.Class bits to determine if any core is awake that can accept the interrupt. If a suitable core is not awake, the GIC then wakes a core.

Arm strongly recommends that if you use GICD_CTLR.E1NWF, you must also set the DPGx bits of register GICR_CTLR to specify whether a core is likely to accept a particular interrupt group in a timely manner. The GIC does not continue to wake cores until one is found. The GIC-600 uses two passes to try to find the best place for a 1 of N interrupt, by using a round-robin arbiter between:

- Any core that has **cpu_active** set, is fully enabled for the interrupt, and has no other pending interrupts.
- Any core that is fully enabled for the interrupt and has no interrupts of a higher priority than the 1 of N interrupt.

If neither option is available to the 1 of N, the interrupt is assigned to any legal target and regularly re-evaluated to ensure that it is not excluded from other SPIs of the same priority.

## 3.6 Power management

The GIC-600 can be powered down by the system power controller, and supports the cores that it services that are also being powered down by the power controller. The GICR_WAKER and the IMPLEMENTATION-DEFINED GICR_PWRR registers provide bits to control functions that are associated with power management.

This section contains the following subsections:

- *3.6.1 Redistributor power management* on page 3-58.
- *3.6.2 Processor core power management* on page 3-58.
- *3.6.3 Other power management* on page 3-59.

### 3.6.1 Redistributor power management

At reset, the Redistributors are considered to be powered down. To power up the Redistributors, software must use the GICR_PWRR register.

**Note**

This requirement is true for all GIC-600 configurations.

The GICR_PWRR register can control Redistributor power management either by operating through the core, or through the Redistributor.

If operating through the core, each core must program its GICR_PWRR.RDPD = 0 and GICR_PWRR.RDAG = 0 to ensure that the Redistributor powers up. Alternatively, a single core can power up the Redistributor for all cores that connect to the same Redistributor by writing GICR_PWRR.RDPD = 0 and GICR_PWRR.RDAG = 1.

You can use GICR_PWRR.RDG to identify which core shares a Redistributor.

The powerup and powerdown sequences are shown in the following pseudocode:

```
Power off (setting RDPD to 1):

// Check group not transitioning.
repeat
until (GICR_PWRR.RDGPD == GICR_PWRR.RDGPO)

// Write to power the CPU off.
GICR_PWRR.RDPD = 1;


Power on (setting RDPD to 0):

repeat
  // Check group not transitioning.
  repeat
  until (GICR_PWRR.RDGPD == GICR_PWRR.RDGPO)

  // Write to power the CPU on.
  GICR_PWRR.RDPD = 0;

  // Check access, if RDPD == 0 then powered on.
until (GICR_PWRR.RDPD == 0)
```

**Note**

GICR_PWRR must be accessed using the GICR address space that relates to the core being powered on or off.

### 3.6.2 Processor core power management

The GIC architecture defines the programming sequence to safely power down a core that is connected to the GIC-600.

The powerdown programming sequence uses the GICR_WAKER.ProcessorSleep bit. When all cores within a cluster are powered down using the architectural sequence, you can power gate the GIC Stream interface for that cluster.

Before a core is powered down, you must set the GICR_WAKER.ProcessorSleep bit to 1. The core must then poll the GICR_WAKER.ChildrenAsleep bit to ensure that there are no outstanding transactions on the GIC Stream interface of the core.

To ensure that there are no interrupts during the powerdown of the core, in a typical powerdown sequence you must:

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

——— **Note** ———

The core powerdown sequence that you use must match the core powerdown sequence that is described in the Technical Reference Manual for your processor.

———————————

When a core is powered down and the GICR_WAKER.ProcessorSleep bit is set to 1, if the GIC-600 receives an interrupt that targets only that core, it attempts to wake the core by asserting the **wake_request** signal that corresponds to that core. The **wake_request** signal is asserted by the Wake Request block and must be connected to the system power controller.

You must not set the GICR_WAKER.ProcessorSleep bit to 1 unless the core enters a power state where the GIC-600 uses the power controller to wake the core instead of the GIC Stream interface. For example, with Cortex®-A53 and Cortex-A57, if the core enters a low-power state that is based on the *Wait For Interrupt* (`WFI`) or *Wait For Event* (`WFE`) instructions, such as retention, you must not set the GICR_WAKER.ProcessorSleep bit to 1.

Interrupts can cause the core to leave the low-power state, entered by executing a `WFI` or `WFE` instruction, as defined in the *Arm® Architecture Reference Manual ARMv8, for ARMv8-A architecture 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 low-power states. In such a system, software has more control of the conditions under which cores leave low-power 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 bit for that core is set, the **wake_request** signal is asserted for that core.

———————————

See the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0* for information about power management, and about wakeup signals and their relation to the core outputs.

### 3.6.3 Other power management

The GIC-600 can be powered up and powered down using non-architectural protocols.

When powering down the GIC-600, software must preserve the state of the GIC-600, except for any LPI pending interrupts that are preserved in pending tables, as defined in the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

You can preserve the LPI pending bits by using an IMPLEMENTATION-DEFINED powerdown sequence, which ensures that the memory pointed to by each GICR_PENDBASER contains the updated pending information for the LPIs. The IMPLEMENTATION-DEFINED powerdown sequence must:

1. Complete the powerdown sequence for all cores.
2. Set GICR_WAKER.Sleep to 1.
3. Poll GICR_WAKER until GICR_WAKER.Quiescent is set.

**Note**

- GICR_WAKER.Sleep can only be set to 1 when:
  - All Redistributors have GICR_WAKER.ProcessorSleep == 1.
  - All Redistributors have GICR_WAKER.ChildrenAsleep == 1.
- GICR_WAKER.ProcessorSleep can only be set to 0 when:
  - GICR_WAKER.Sleep == 0.
  - GICR_WAKER.Quiescent == 0.
- If software decides to abort a sleep request due to an external wake request, it can do so by clearing GICR_WAKER.Sleep at any time. Software does not have to wait for GICR_WAKER.Quiescent to be set.
- There is only one GICR_WAKER.Sleep and one GICR_WAKER.Quiescent bit that can be read and written through the GICR_WAKER register of any Redistributor.

The powerdown described sequence ensures that all LPIs that are acknowledged by a write response to the write GITS_TRANSLATER are saved to the Pending tables. Any interrupt that arrives when the Sleep bit is set to 1 is ignored, and the ACE-Lite transaction completes in accordance with the ACE protocol.

Arm recommends that you disable any interrupt sources before setting GICR_WAKER.Sleep. However, if you require wake-on-interrupt behavior, the Write to GITS_TRANSLATER must be gated upstream at a location that enables software to reprogram and enable the GIC-600 without deadlock.

When the GICR_WAKER.Quiescent bit is set, it is safe to power down the GIC-600 without losing LPI pending bits. Software must still perform other steps such as the save and restore of SPI state. However, you must provide custom mechanisms to wake the GIC-600 if any interrupts arrive that must not be ignored.

When the GIC-600 next powers up, 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 speed up the initialization of the GIC-600 as follows:

1. Zero the Pending table.
2. Set the GICR_PENDBASER.PTZ bit to 1.

**Note**

GICR_PENDBASER registers can only be modified before the GICR_CTLR.Enable_LPIs bit is set, or when bits GICR_WAKER.Sleep and GICR_WAKER.Quiescent are both set.

***Related reference***

*4.4.3 Power Management Control Register, GICR_WAKER* on page 4-130

***Related information***

*Arm® GICv3 and GICv4 Software Overview*

## 3.7 Getting started

There are some basic tasks that you must complete before you can start to use the GIC-600.

The Redistributor(s) must be powered on using the GICR_PWRR register to enable the Redistributor(s) to be accessed, see *3.6.1 Redistributor power management* on page 3-58 for more information.

When the GIC-600 is powered up, it must be programmed as described in the *Arm® GICv3 and GICv4 Software Overview*.

## 3.8 Security

The GIC-600 supports the Arm TrustZone technology. Each INTID must be assigned a group and security setting.

The GIC-600 supports the three interrupt groups that are shown in the following table.

**Table 3-1 Security and groupings**

| Interrupt type | Example use |
|---|---|
| Secure Group 0 | Interrupts for EL3 (Secure firmware) |
| Secure Group 1 | Interrupts for Secure EL1 (Trusted OS) |
| Non-secure Group 1 | Interrupts for the Non-secure state (OS and the Hypervisor, or one of both) |

The following table shows the interrupt signals that are used for each interrupt group, Security state, and Exception level.

**Table 3-2 Interrupt signals, Security states, and Exception levels**

| Core Exception level and Security state | Group 0 | Group 1 | |
|---|---|---|---|
| | | Secure | Non-secure |
| Secure EL0, EL1 | FIQ | IRQ | FIQ |
| Non-secure EL0, EL1, EL2 | FIQ | FIQ | IRQ |
| EL3 | FIQ | FIQ | FIQ |

Setting the *Disable Security* (DS) bit to 1 in the GICD_CTLR register removes the security support of the GIC-600. It can be set by Secure software during the boot sequence or configured to be always set when you configure the design using the parameter `ds_value`. When the system has no concept of security, you must set GICD_CTLR.DS to allow access to important registers.

If you set GICD_CTLR.DS to 1, only a single Security state is supported. In a single Security state, register access, and the behavior and number of interrupt groups supported are affected. For more information, see *Interrupt grouping*, and *Interrupt grouping and security* in the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

**Note**

Arm recommends that you only set GICD_CTLR.DS if either your system does not support security, or the only software you run does not use security. See *Security model* in the *Arm® GICv3 and GICv4 Software Overview* for more information about the implications of setting GICD_CTLR.DS to 1.

If you run software without security awareness on a system that supports security, 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-600 from any Exception level and use two interrupt groups, Group 0 and Group 1, so that interrupts can target both the FIQ and IRQ handlers on a core.

Group 0 is always Secure in systems with security. If you decide to write security-unaware software using Group 0, it might not be portable to systems with a concept of security. Security-unaware software is most portable when written using 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.

In security aware systems, Secure software can prevent the DS bit from being written by writing to Disable Security Lock bit (GICD_SAC.DSL). Once set, only a hardware reset can clear the DSL bit.

If you know that your system is always security aware, then Arm recommends configuring the GIC-600 without DS support.

## 3.9 Backwards compatibility

The GIC-600 does not support legacy operation, that is, when GICD_CTLR.ARE_S or GICD_CTLR.NS == 0.

Therefore, SGIs and PPIs can be programmed only through the GICR register space, and SGIs are not banked by the source core.

## 3.10 Interrupt translation service (ITS)

The GIC-600 supports up to 32 ITS blocks in the system with a limit of 16 per chip. Each ITS is responsible for translating message-based interrupts from peripherals into LPIs.

Each ITS is compliant with the GICv3 architecture and is responsible for mapping translation requests with an EventID and DeviceID through to the *physical INTID* (pINTID) and Collection, a group of interrupts, and finally to the target core. The following figure shows the ITS process.

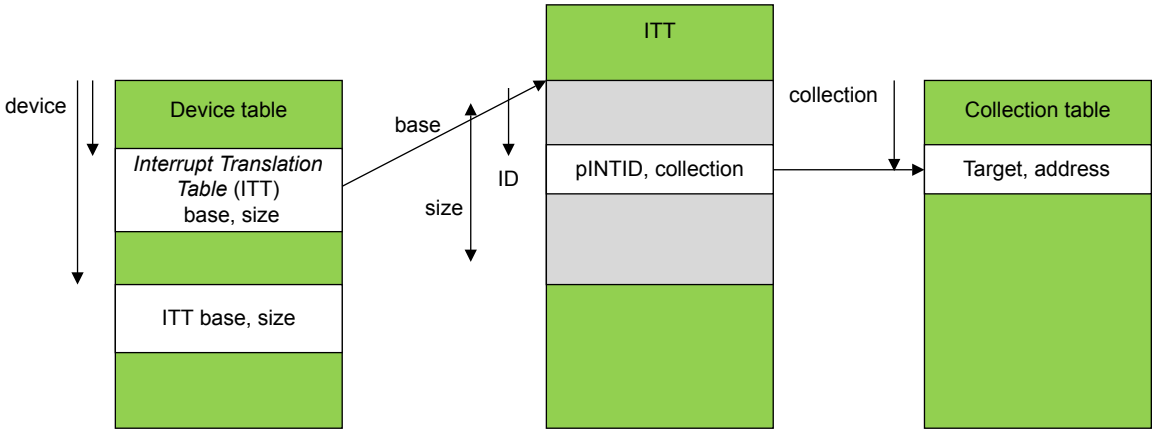


**Figure 3-2 ITS process**

To reduce memory traffic and keep interrupt latency to a minimum, GIC-600 has three two-way set associative caches in each ITS:

- DeviceID to ITT base address.
- DeviceID and EventID to collection.
- Collection to target core.

In small configurations, these caches might be too small to be worth the overhead of implementing them as SRAM. If ECC protection is not required for a cache implemented as an array of flops, and to reduce RAM area, you can remove ECC from each RAM individually, see the *Arm® CoreLink™ GIC-600 Generic Interrupt Controller Configuration and Integration Manual* for more information.

It is common for the DeviceID to be a non-contiguous number that is derived from the PCIe RequestorID. To ensure that this does not result in a sparse DeviceID table and wasted memory, the GIC-600 supports indirect Device tables (GITS_BASERn.Indirect = 1) where the first-level table points at subtables that can be allocated at runtime. See the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0* for more details.

The GIC-600 uses memory-backed collections only, which means that before the ITS is enabled by writing to GITS_CTLR.Enabled, memory must be allocated for the Device table, the Collection table, and the ITS Command queue. Inline with the architecture, these tables must be pre-cleared to 0 by software, apart from pointers to cleared level-two Device tables, unless the tables were previously populated by the GIC-600.

The GIC-600 ITS supports all GICv3 commands as described in the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

GITS_TYPER.PTA is 0 for all configurations, which means that all references to processor cores in ITS commands are implemented through the GICR_TYPER.ProcessorNumber field.

Command and translation errors are reported through the RAS registers. See *3.16 Reliability, Accessibility, and Serviceability* on page 3-75.

For details on how to program and use the ITS, see the *GICv3 and GICv4 Software Overview*.

This section contains the following subsections:

- *3.10.1 ITS cache control, locking, and test* on page 3-66.
- *3.10.2 ITS commands and errors* on page 3-66.

### 3.10.1 ITS cache control, locking, and test

The GIC-600 can lock certain interrupts to the ITS caches.

The ITS caches are described in *3.10 Interrupt translation service (ITS)* on page 3-65.

If an LPI or translation is missed in a cache, several memory reads can be required to obtain the data necessary from memory. This can result in a range of latency that might not be acceptable for some LPIs.

The GIC-600 can lock certain translations into the ITS caches, with the following guarantee:

- Interrupts that are locked in ITS caches always hit and never require any translation.

The ITS caches are automatically managed and invalidated as necessary when the GITS_BASERn registers are updated. Therefore, software intervention is not required. However, to aid debug and integration testing, you can force invalidation of the appropriate cache by setting the relevant bit in the GITS_FCTLR register.

A forced invalidation of the Event cache abandons all locked entries.

The GITS_OPR and GITS_OPSR registers control cache locking, which is DEVICE_ID, EVENT_ID, and the correct GITS_OPR.LOCK_TYPE (ITS = 2). The GIC attempts to perform the lock, and reports in GITS_OPSR. If the lock succeeds, GITS_OPSR.REQUEST_COMPLETE == 1 and GITS_OPSR.REQUEST_PASS == 1.

Each cache set is two-way set associative. Only one entry can be locked in each cache set. Any attempt to lock both ways in a set, reports as failed in GITS_OPSR. You can also use the GITS_OPSR register to unlock entries that are locked.

The GITS_OPSR register has two test features:

- Trial: Tests the mapping by writing a DeviceID and EventID to GITS_OPR with GITS_OPR.LOCK_TYPE = 1 (TRIAL). This causes the ITS to translate the supplied DeviceID and, or EventID pair, and report the generated target and *physical ID* (PID) in GITS_OPSR. It also reports if the translation fails, GITS_OPSR.REQUEST_PASS == 0, or if it hit a locked entry, GITS_OPSR.ENTRY_LOCKED. The interrupt is not set to pending.
- Track: Can be used to detect the arrival of a certain EventID and, or DeviceID pair, which is reported by setting GITS_OPSR.REQUEST_COMPLETE.

While any GITS_OPR operation, other than Track, is in progress, the GITS_OPSR.REQUEST_IN_PROGRESS bit is set and no further updates are accepted by GITS_OPR until the previous operation completes. To ensure that the operation is accepted, Arm recommends that the GITS_OPR value is read after writing. You can abort Track operation by writing GITS_OPR.LOCK_TYPE == Track_abort.

### 3.10.2 ITS commands and errors

Each ITS detects a wide range of command errors and translation errors, and reports them in ARMv8.2 RAS architecture-compliant error records.

The ITS record error syndromes comprise four groups that each have separate enables in the GITS_FCTLR register. The following table shows the ITS record error syndrome groups.

**Table 3-3 ITS record error syndrome groups**

| Group | Control |
|---|---|
| ACE-Lite slave Write translation errors. Only when the ITS has a separate ACE-Lite slave port. | GITS_FCTLR.AEE (Access Error Enable) |
| Translation errors on incoming writes to GITS_TRANSLATER. | GITS_FCTLR.UEE (Unmapped Error Enable) |
| Errors during commands. | GITS_FCTLR.CEE (Command Error Enable) |
| Other errors, such as, memory system, or memory allocation errors. | None |

See *ITS command and translation error records 13+* on page 3-88 for information about all the detected syndromes.

ITS commands must be written by software before they are executed.

The ITS Command queue operates a stall mechanism on any error, irrespective of the GITS_FCTLR.CEE value. To execute commands, software writes to a Command queue in memory and then updates the GITS_CWRITER.Offset to indicate that there are commands to run. See *3.7 Getting started* on page 3-61 for more information. Normally, the GITS_CREADR.Offset increments until it matches the GITS_CWRITER.Offset, wrapping as necessary, to indicate that the Command queue has completed.

If an error occurs, GITS_CREADR.Stalled is set, which indicates that processing has stopped and software intervention is required. If GITS_FCTLR.CEE is set, at least one error is reported in the relevant error record to aid software debug. You can correct the command identified by GITS_CREADR and resume the Command queue by writing to GITS_CWRITER.Retry. If the command is no longer required, you must rewrite it as a `SYNC` command before you resume.

To determine when Command queue execution has completed, you can use either one of two methods:

- Polling GITS_CREADR.Offset until it matches GITS_CWRITER.Offset.
- Putting an `INT` command in the queue and waiting for that interrupt to arrive.

For the second method, Arm recommends that you enable GITS_FCTLR.CEE and that you configure the fault_handling, or error_recovery interrupt to be delivered to a core that can resolve Command queue issues. See *3.16 Reliability, Accessibility, and Serviceability* on page 3-75 for more information.

## 3.11 LPI caching

If LPI support is configured, the GIC-600 supports a single LPI cache per chip.

The LPI cache is two-way set associative based on the lowest bits of the LPI INTID, and stores LPI properties from the LPI Property table. The relevant set is checked for valid properties as each LPI arrives in the system.

The cache is fully associative for pending LPIs, which means that the LPI system fills almost all lines in the cache before sending anything to the Pending tables. The GIC-600 is not optimized for collating LPIs that have the same INTID. However the system is designed to reorder and sort the cache over time. In some circumstances, this can cause duplicated interrupts to not be collated efficiently. However, the reduced use of the Pending table, results in better latency bounds under load.

This method of caching means that priorities are associated with an incoming LPI and remain with it until it is serviced. Changes in the LPI Property table are not accepted by the GIC until the relevant `INV` and `SYNC` commands are executed through an ITS, GICR_INVLPIR or GICR_INVALLR.

The GIC-600 considers priority and enable when choosing data to retain in the cache. However, pending interrupts always take priority over interrupts that are not pending, so there is no guarantee that the highest priority interrupt data always remains stored in the cache.

***Related reference***

*2.1.7 Distributor configuration* on page 2-28

## 3.12 Memory access and attributes

The LPI and ITS transactions are located in memory tables whose locations are defined in registers that specify their base address, size, and access attributes.

Arm recommends that all tables are placed in Normal memory. All ITS tables are private, and after allocation, are accessed only by the GIC. However, the LPI Property table and ITS Command queue are written to by cores, and read by the GIC.

The following table shows the **a<x>cache** and **a<x>domain** mappings for the memory transactions that the GIC generates.

**Table 3-4 Memory access registers**

| Access type | Register | Mapping control bit[e] |
|---|---|---|
| LPI Property table | GICR_PROPBASER | GICD_FCTLR.DCC |
| LPI Pending table | GICR_PENDBASER | |
| ITS Device table | GITS_BASER0 | GITS_FCTLR.DCC |
| ITS Translation table | GITS_BASER0 | |
| ITS Collection table | GITS_BASER1 | |
| ITS Command queue | GITS_CBASER | |

The main cacheability value is derived from the *BASER*.OuterCache field, unless it is zero, in which case the cacheability value is a value that is shown in the following table.

**Table 3-5 Cacheability values**

| Main cacheability value (*BASER*.OuterCache field) | Other cacheability value (*BASER*.InnerCache field) | arcache | awcache | arcache (DCC = 1) | awcache (DCC = 1) |
|---|---|---|---|---|---|
| `0b000`, Device-nGnRnE | - | `0b0010` | `0b0010` | `0b0010` | `0b0010` |
| `0b001`, Normal Non-cacheable | Match | `0b0011` | `0b0011` | `0b0011` | `0b0011` |
| `0b001`, Normal Non-cacheable | No match | `0b0011` | `0b0011` | `0b0011` | `0b0011` |
| `0b010`, Normal Cacheable RA Write-Through | Match | `0b0011` | `0b0011` | `0b1110` | `0b0110` |
| `0b010`, Normal Cacheable RA Write-Through | No match | `0b0011` | `0b0011` | `0b1110` | `0b0110` |
| `0b011`, Normal Cacheable RA Write-Back | Match | `0b1111` | `0b0111` | `0b1111` | `0b0111` |
| `0b011`, Normal Cacheable RA Write-Back | No match | `0b0011` | `0b0011` | `0b1111` | `0b0111` |
| `0b100`, Normal Cacheable WA Write-Through | Match | `0b0011` | `0b0011` | `0b1010` | `0b1110` |
| `0b100`, Normal Cacheable WA Write-Through | No match | `0b0011` | `0b0011` | `0b1010` | `0b1110` |
| `0b101`, Normal Cacheable WA Write-Back | Match | `0b1011` | `0b1111` | `0b1011` | `0b1111` |
| `0b101`, Normal Cacheable WA Write-Back | No match | `0b0011` | `0b0011` | `0b1011` | `0b1111` |
| `0b110`, Normal Cacheable WA RA Write-Through | Match | `0b0011` | `0b0011` | `0b1110` | `0b1110` |

[e] The mappings are designed for the ARMv8 and ARMv8.2 generation of cores. However, setting this bit converts the GIC-600 to full featured mapping.

**Table 3-5 Cacheability values (continued)**

| Main cacheability value (*BASER*.OuterCache field) | Other cacheability value (*BASER*.InnerCache field) | arcache | awcache | arcache (DCC = 1) | awcache (DCC = 1) |
|---|---|---|---|---|---|
| `0b110`, Normal Cacheable WA RA Write-Through | No match | `0b0011` | `0b0011` | `0b1110` | `0b1110` |
| `0b111`, Normal Cacheable WA RA Write-Back | Match | `0b1111` | `0b1111` | `0b1111` | `0b1111` |
| `0b111`, Normal Cacheable WA RA Write-Back | No match | `0b0011` | `0b0011` | `0b1111` | `0b1111` |

Signal **a<x>domain** is driven according to the *BASER*.Shareability field unless the resultant cacheability is Device, or Non-cacheable, in which case it becomes `0b11`, system Shareable in accordance with the *Arm® AMBA® AXI and ACE Protocol Specification*.

## 3.13 MSI-64

The MSI-64 Encapsulator can be used to combine the DeviceID into single memory access writes to the GITS_TRANSLATER register in the ITS.

The ITS translates DeviceID/EventID pairs into LPI physical INTIDs.

A normal MSI/MSI64 write contains the EventID in the lower 16 bits or 32 bits of data. However, the DeviceID must be transported using a different method. The DeviceID is often derived directly from a PCIe RequestorID or *System Memory Management Unit* (SMMU) StreamID.

The GIC-600 ITS supports two mechanisms:

**awuser_*_s**
: The DeviceID arrives on sideband user signals. The system integrator must ensure that rogue software cannot directly or indirectly, perform an access to the GITS_TRANSLATER register with a DeviceID that matches a real device.

**MSI-64**
: When configured to support MSI-64, the ITS expects the DeviceID to be in the upper 32 bits of a 64-bit write to the GITS_TRANSLATER register.
To prevent rogue software accessing the GITS_TRANSLATER register and spoofing any device, Arm recommends that the GITS_TRANSLATER register is moved to an arbitrary page that is protected by the Hypervisor.
The GIC-600 uses two methods to support this:

- The MSI-64 Encapsulator modifies the page address of accesses to the architectural GITS_TRANSLATER address, set by the **msi_translator_page** tie-off, to the system-defined page set by **msi64_translator_page**.
- When the ITS shares an ACE-Lite slave port, a separate page address tie-off **gits_transr_page_offset**, allows the GITS_TRANSLATER register page to be moved to anywhere in the address map to match the **msi64_translator_page** value that is independent of the GICD address map reset.

  ——— **Note** ———

  The **msi64_translator_page** and **gits_translater_page**, or one of either, must not be on top of any other GIC register page.

  ———————————

  To ensure that this method of mapping is hidden from software, all accesses to the GITS_TRANSLATER register must pass through an Encapsulator, or similar embedded functionality. See *2.4 MSI-64 Encapsulator* on page 2-39 for more information about the MSI-64 Encapsulator.

## 3.14 RAM

The GIC-600 uses multiple RAMs to store a range of states for all types of interrupt.

In typical operation, the RAMs are transparent to software.

A RAM is protected from errors using an ECC with *Single Error Correction and Double Error Detection* (SECDED). If single or double errors are detected, they are reported in the software visible error records, see *3.16 Reliability, Accessibility, and Serviceability* on page 3-75 for more information.

For all ECC schemes that are used in the GIC-600, the correction code is 0 when all data in the RAM is 0.

## 3.15 Performance Monitoring Unit

The GIC-600 contains a PMU for counting key GIC events from both the Distributor and any configured ITS blocks on the same chip.

——— **Note** ———

Redistributor events are not tracked by the PMU. The delivery of PPI and SGI interrupts can be counted by recording calls to the core interrupt service routine.

———————————

The GIC events are described in *Table 4-60 EVENT field encoding* on page 4-173.

The PMU has five counters with snapshot capability and overflow interrupt.

Secure and Non-secure interrupts are counted together and therefore Non-secure software cannot, by default, access the GICP (PMU) register space. However, Secure software can decide to allow access. This can be done by programming the GICD_SAC.GICPNS bit, or by integrating the GIC with the **gicp_allow_ns** tie-off set HIGH.

——— **Note** ———

If GICD_CTLR.DS == 1, the GICP register space is accessible to all software.

———————————

### Count configuration

Each PMU counter can be programmed individually to count a range of events.

To configure a counter:

1. Program the counter GICP_EVCNTRn to a known value. This could be 0 to count events, or a higher number to trigger an overflow after a known number of events.
2. Program the associated GICP_EVTYPERn to count the required event.
3. Program the required filter type for the event by programming GICP_FRn.
4. Enable the counter by programming the corresponding bit in GICP_CNTENSET0.
5. Repeat the previous steps for all counters that are required.
6. Enable the global count enable in GICP_CR.E.

——— **Note** ———

PMU registers, other than enables, do not have resets and must be programmed before use.

———————————

### Overflow interrupt

The overflow interrupt can be enabled on a per counter basis by enabling the relevant bit of GICP_INTENSET0, where bit[0] enables GICP_EVCNTR0, bit[1] enables GICP_EVCNTR1, and so on. Similarly, the overflow interrupt enable can be disabled by corresponding writes to GICP_INTENCLR0.

When enabled, the interrupt activates at any of these events:

- A write to a GICP_OVSSET0 for any counter.
- An overflow on any enabled counter.

The GICP_OVSSET0 and GICP_OVSCLR0 can be used for save and restore operations and for testing the correct integration of the **pmu_int** interrupt.

The **pmu_int** can be used to trigger external logic, for example, to trigger a read of the captured data.

Alternatively, by programming a valid SPI ID into the GICP_IRQCR.SPIID field, the **pmu_int** SPI is delivered internally in accordance with normal SPI programming.

### Snapshot

Each PMU counter GICP_EVCNTRn has a corresponding GICP_SVRn snapshot register. On a snapshot event, all five counters are copied to their backup registers so that all consistent data is copied out over a longer period.

The snapshot events are:

- A handshake on the four phase **sample_req**/**sample_ack** external handshake.
- A write of 1 to the GICP_CAPR register CAPTURE bit.
- An overflow of an enabled counter when GICP_EVTYPERn.OVFCAP is set.

**Note**

There is only one set of snapshot registers, therefore data is replaced in multiple capture events.

## 3.16 Reliability, Accessibility, and Serviceability

The GIC-600 uses a range of RAS features for all RAMs, which include SECDED, ECC, and Scrub, software and bus error reporting.

The GIC makes all necessary information available to software through v8.2 RAS architecture-compliant register space.

This section contains the following subsections:


- *3.16.1 Non-secure access* on page 3-75.
- *3.16.2 Scrub* on page 3-75.
- *3.16.3 Error record classification* on page 3-75.
- *3.16.4 ECC error reporting and recovery* on page 3-75.
- *3.16.5 Error recovery and fault handling interrupts* on page 3-76.
- *3.16.6 Error handling records* on page 3-77.
- *3.16.7 Bus errors* on page 3-95.


### 3.16.1 Non-secure access

You can control whether Non-secure software has access to the RAS architecture-compliant register space by using GICD_SAC.GICTNS. Its reset value is set by the **gict_allow_ns** tie-off signal.

In the case of an error, and if the GICD_CTLR.DS == 0, all SPIs, PPIs, and SGIs, resort to a Secure group. Therefore, interrupt programming is not revealed to the Non-secure side.

### 3.16.2 Scrub

The GIC-600 holds significant programming and interrupt states in RAM, which is protected by SECDED and ECC.

However, the contents of some RAM is expected to be static over long periods of time, and there is a potential for errors to accumulate if a particular address is not accessed after a period. To prevent this a scrub system is used, whereby software can trigger periodically, a low-priority scrub through GITS_FCTLR.SIP, GICR_FCTLR.SIP, and GICD_FCTLR.SIP. This process triggers a check and if necessary, a Write-Back, of all valid RAM entries. Any errors that are found during a scrub are also reported in the relevant RAS error record.

### 3.16.3 Error record classification

The GIC reports errors in ARMv8.2 RAS architecture-compliant error records, which are accessible through the ACE-Lite slave programming interface.

There are four classes of error records:

- Correctable ECC errors.
- Uncorrectable ECC errors.
- ITS command and translation errors.
- Software access errors.

The error records have a separate reset so that they can be read after a main GIC reset to determine any problems.

### 3.16.4 ECC error reporting and recovery

When an ECC error is detected, the GIC-600 attempts to contain the error and ensure it cannot propagate further.

The following table shows the GIC behavior when errors are detected in each RAM.

**Table 3-6 ECC error reporting**

| RAM | Action in response to an Uncorrectable Error |
|---|---|
| ITS caches | All ITS caches are memory that is backed. The contents are reloaded from memory. However, if entries are locked in the errored cache line, the lock is lost. Software can use the GITS_OPSR register to determine if all expected locked entries are still in place. |
| SPI | The SPI is flagged as being in error and the error is reported through the GICD_IERRRn register. The corrupted RAM contents can be read until the error is cleared by writing GICD_IERRRn. SPIs that are in the error state can also be determined by reading the GICD_IERRRn register. This SPI is not reused until it is reprogrammed and re-enabled. |
| LPI | All information from the RAM entry is reported. Software can determine the set of interrupts that might have errors, based on the reported ID, to check priority, and to target information.<br>**Note**<br>Repeated double errors in the LPI cache cause an overflow of the error record, which means subsequent information is lost. Arm recommends that a high priority SPI is used to trigger a core to clear the error record as fast as possible. |
| Redistributor RAM | In the Redistributor, only group and priority are maintained in the RAM. If an error occurs, this information becomes UNKNOWN for four interrupts. Pending and Active states are maintained but the enable is cleared so that the interrupt is not forwarded.<br>You can determine the interrupts that are in error by reading the GICR_IERRVR register.<br>**Note**<br>Because the group is UNKNOWN, it is assumed to be Secure, and therefore interrupt deactivates can be ignored. Software must consider this as part of the recovery sequence.<br>It is also possible for a GenerateSGI packet to become corrupted. In this case, the GenerateSGI is reported as bad.<br>For more information about Pending and Active PPI states, see the *Arm® GICv3 and GICv4 Software Overview*. |
| SGI | The SGI RAM holds group and *Non-Secure Access Control* (NSACR) information for all cores. It is used to enable wakeup of the Redistributor as required. If an error occurs in the RAM, then all SGIs for that core are considered to be Secure. This prevents Non-secure masters from raising Secure interrupts incorrectly. |

### 3.16.5 Error recovery and fault handling interrupts

You can assign a recorded correctable ECC error to the fault_handling interrupt by setting GICT_ERR<n>CTLR.CFI.

All correctable ECC errors have error counters, therefore, the interrupt only fires when the counter in the associated GICT_ERR<c>MISC0 register overflows. You can preset the counter to any value by writing to GICT_ERR<c>MISC0.Count. For example, to fire an interrupt on any correctable error, write `0xFF`, or to fire an interrupt on every second correctable error, write `0xFE`.

You can assign a recorded uncorrectable ECC error either to the fault-handling interrupt, **fault_int**, by setting GICT_ERR<n>CTLR.FI, or to the error recovery interrupt, **err_int**, by setting GICT_ERR<n>CTLR.UI. The interrupt fires on every uncorrectable interrupt occurrence irrespective of the counter value.

You can route interrupts **fault_int** and **err_int** out as interrupt wires for situations where error recovery is handled by a core that does not receive interrupts directly from the GIC, such as a central system control processor. Alternatively, you can drive each interrupt internally by programming the associated GICT_ERRIRQCR<n> register.

Each GICT_ERRIRQCR<n> register contains an ID field that must be programmed to 0 if internal routing is not required, or if internal routing is required, to a legally supported SPI ID. If the programmed

ID value is less than 32, out of range, or for multichip configurations, not owned on chip, the register updates to 0 and no internal delivery occurs.

Arm recommends that if the **err_int** and **fault_int** are internally routed, the target interrupts must not have SPI Collator wires, or if they are present, are tied off. This prevents software checking for the same ID at multiple destinations.

The **err_int** and **fault_int** do not have direct test enable registers. You can test connectivity using error record 0 and triggering an error, such as an illegal AXI access to a non-existent register.

### 3.16.6 Error handling records

The GIC-600 has several error records. The range of error handling records that are available depends on the configuration of the GIC-600.

The GIC-600 error handling records are listed in the following table.

**Table 3-7 Error handling records**

| Record | Description |
|---|---|
| 0 | Uncorrected software error in the Distributor.<br>*Table 3-8 Software errors, record 0* on page 3-78 |
| 1 | Corrected SPI RAM error.<br>*Table 3-9 SPI RAM errors, records 1-2* on page 3-85 |
| 2 | Uncorrected SPI RAM error.<br>*Table 3-9 SPI RAM errors, records 1-2* on page 3-85 |
| 3 | Corrected SGI RAM error.<br>*Table 3-10 SGI RAM errors, records 3-4* on page 3-86 |
| 4 | Uncorrected SGI RAM error.<br>*Table 3-10 SGI RAM errors, records 3-4* on page 3-86 |
| 5 | Reserved. |
| 6 | Reserved. |
| 7 | Corrected PPI RAM error.<br>*Table 3-11 PPI RAM errors, records 7-8* on page 3-87 |
| 8 | Uncorrected PPI RAM error.<br>*Table 3-11 PPI RAM errors, records 7-8* on page 3-87 |
| 9 | Corrected LPI RAM error. Not present if there is no LPI support.<br>*Table 3-12 LPI RAM errors, records 9-10* on page 3-87 |
| 10 | Uncorrected LPI RAM error. Not present if there is no LPI support.<br>*Table 3-12 LPI RAM errors, records 9-10* on page 3-87 |
| 11 | Corrected error from ITS RAM. Not present if an ITS is not present.<br>*Table 3-13 ITS RAM errors, records 11-12* on page 3-88 |

**Table 3-7 Error handling records (continued)**

| Record | Description |
|---|---|
| 12 | Uncorrected error from ITS RAM. Not present if an ITS is not present.<br>*Table 3-13 ITS RAM errors, records 11-12* on page 3-88 |
| 13+ | Uncorrected software error in ITS. One record per ITS on the chip. Not present if an ITS is not present.<br>*Table 3-15 ITS command and translation errors, records 13+* on page 3-89 |

The details, events, and recovery sequences of each record are described in *Software error record 0* on page 3-78 to *ITS command and translation error records 13+* on page 3-88.

### Software error record 0

Software error record 0 records software errors that are uncorrectable.

Record 0 contains software programming errors from a wide range of sources within the GIC-600. In general, these errors are contained. For uncorrected errors, the information that is provided gives enough information to enable recovery without significant loss of functionality.

Arm recommends that record 0 is connected to a high priority interrupt. This prevents the record from overflowing if it receives more errors than it is able to process with the possible loss of information required for recovery. See *3.16.5 Error recovery and fault handling interrupts* on page 3-76 for more information.

The following table describes the syndromes that are recorded in record 0, the reported information, and recovery instructions.

**Table 3-8 Software errors, record 0**

| GICT_ERR<n>STATUS.IERR (Syndrome) | GICT_ERR<n>STATUS.SERR | GICT_ERR<n>MISC0.Data<br>Description (other bits RES0)[f] | Recovery, Prevention |
|---|---|---|---|
| `0x0`, SYN_ACE_BAD<br>Illegal ACE-Lite Slave Access. | `0xE` | AccessRnW, bit[12]<br>AccessSparse, bit[11]<br>AccessSize, bits[10:8]<br>AccessLength, bits[7:0] | Repeat illegal access, with appropriate size and properties.<br>Full access address is given in GICT_ERR0ADDR. |
| `0x1`, SYN_PPI_PWRDWN<br>Attempt to access a powered down Redistributor. | `0xF` | Redistributor, bits[24:16]<br>Core, bits[8:0] | Ensure that the Redistributor is powered up before accessing. See *4.4.5 Power Register, GICR_PWRR* on page 4-132.<br>Attempt was made by the core reported in MISC0. |

[f] Always packed from 0 (lowest = 0).

**Table 3-8 Software errors, record 0 (continued)**

| GICT_ERR<n>STATUS.IERR (Syndrome) | GICT_ERR<n>STATUS.SERR | GICT_ERR<n>MISC0.Data<br>Description (other bits RES0)[f] | Recovery, Prevention |
|---|---|---|---|
| `0x2`, SYN_PPI_PWRCHANGE<br>Attempt to power down Redistributor rejected. | `0xF` | Redistributor, bits[24:16]<br>Core, bits[8:0] | Ensure that the core accessing the register, or all cores with the same GICR_PWRR.RDG if GICR_PWRR.RDAG is set, has completed the GICR_WAKER.ProcessorSleep handshake. |
| `0x3`, SYN_GICR_ARE<br>Attempt to access GICR or GICD registers in mode that cannot work. | `0xF` | Core, bits[8:0] | Repeat the access to the specified core accessing the correct register space. That is, if ARE_S and ARE_NS == 1 then PPI and SGI registers must be accessed through the GICRx instead of GICD register space. |
| `0x4`, SYN_PROPBASE_ACC<br>Attempt to reprogram PROPBASE registers to a value that is not accepted because another value is already in use. | `0xF` | Core, bits[8:0] | GICR_PROPBASER is shared between all cores on a chip. Once any GICR_CTLR.Enable_LPIs bit is set, the value is locked and cannot be updated unless a complete GICR_WAKER.Sleep handshake is complete.<br>See *A.2 Power control signals* on page Appx-A-187. |
| `0x5`, SYN_PENDBASE_ACC<br>Attempt to reprogram PENDBASE registers to a value that is not accepted because another value is already in use. | `0xF` | Core, bits[8:0] | Once any GICR_CTLR.Enable_LPIs bit is set, the Shareability, InnerCache, and OuterCache fields are locked for the whole chip. They can only be changed by completing the GICR_WAKER.Sleep handshake.<br>See *A.2 Power control signals* on page Appx-A-187. Otherwise, repeat the register access using the current global values. |

[f] Always packed from 0 (lowest = 0).

**Table 3-8 Software errors, record 0 (continued)**

| GICT_ERR<n>STATUS.IERR (Syndrome) | GICT_ERR<n>STATUS.SERR | GICT_ERR<n>MISC0.Data<br>Description (other bits RES0)[f] | Recovery, Prevention |
|---|---|---|---|
| 0x6, SYN_LPI_CLR<br>Attempt to reprogram ENABLE_LPI when not enabled and not asleep. | 0xF | Core, bits[8:0] | Arm recommends that you do not clear the Enable_LPIs bit. Instead, interrupts must be unmapped using an ITS. If you must clear, then you must flush the LPI cache using the GICR_WAKER.Sleep handshake.<br>See *A.2 Power control signals* on page Appx-A-187. |
| 0x7, SYN_WAKER_CHANGE<br>Attempt to change GICR_WAKER abandoned due to handshake rules. | 0xF | Core, bits[8:0] | GICR_WAKER.ProcessorSleep and GICR_WAKER.ChildrenAsleep form a four-phase handshake. The attempt to change state must be repeated when the previous transition has completed. |
| 0x8, SYN_SLEEP_FAIL<br>Attempt to put GIC to sleep failed as cores are not fully asleep. | 0xF | Core, bits[8:0] | All cores must be asleep, using the GICR_WAKER.ProcessorSleep handshake, before you flush the LPI cache using GICR_WAKER.Sleep. |
| 0x9, SYN_PGE_ON_QUIESCE<br>Core put to sleep before its Group enables were cleared. | 0xF | Core, bits[8:0] | The core must disable its group enables before it toggles the GICR_WAKER.ProcessorSleep handshake, otherwise, the GIC clears its record of the group enables, causing a mismatch between the GIC and the core. |
| 0xA, SYN_GICD_CTLR<br>Attempt to update GICD_CTLR was prevented due to *Register Write Pending* (RWP) or Group enable restrictions. | 0xF | Data, bits[7:0] | Software must wait for GICD_CTLR.RWP to be 0 before repeating the GICD_CTLR Write. The data represents the target value. |

[f] Always packed from 0 (lowest = 0).

**Table 3-8 Software errors, record 0 (continued)**

| GICT_ERR<n>STATUS.IERR (Syndrome) | GICT_ERR<n>STATUS.SERR | GICT_ERR<n>MISC0.Data<br>Description (other bits RES0)[f] | Recovery, Prevention |
|---|---|---|---|
| `0x10`, SYN_SGI_NO_TGT<br>SGI sent with no valid destinations. | `0xE` | Core, bits[8:0] | If the SGI is required, software must repeat the SGI from the reported core with a valid target list.<br>If this level of RAS functionality is required, the software must track generated SGIs externally. |
| `0x11`, SYN_SGI_CORRUPTED<br>SGI corrupted without effect. | `0x6` | Core, bits[8:0] | An SGI is corrupted due to a RAM error in the PPI RAM. The RAM error details are reported separately in record 8. The GIC ignores the SGI generated from the recorded core. If you want software to recover from this, it must use an external record of the generated SGI. |
| `0x12`, SYN_GICR_CORRUPTED<br>Data was read from GICR register space that has encountered an uncorrectable error. | `0x6` | GICT_ERR0ADDR is populated | Software has tried to read corrupted data stored in SGI RAM or PPI RAM. Check records 4 and 8, and perform a recovery sequence for those interrupts. |
| `0x13`, SYN_GICD_CORRUPTED<br>Data was read from GICD register space that encountered an uncorrectable error. | `0x6` | GICT_ERR0ADDR is populated | Software has tried to read corrupted data stored in SPI RAM.<br>Check record 2 and perform a recovery sequence for those interrupts. |
| `0x14`, SYN_ITS_OFF<br>Data was read from an ITS that is powered down. | `0xF` | GICT_ERR0ADDR is populated | Ensure that the **qreqn_its<x>** power control Q-Channel is in the RUN state before accessing the relevant ITS. |
| `0x18`, SYN_SPI_BLOCK<br>Attempt to access a SPI block that is not implemented. | `0xE` | Block, bits[4:0] | No recovery required.<br>Correct the software. |
| `0x19`, SYN_SPI_OOR<br>Attempt to access a non-implemented SPI using (SET\|CLR)SPI. | `0xE` | ID, bits[9:0] | Reprogram the issuing device to send a supported SPI ID. |

[f] Always packed from 0 (lowest = 0).

**Table 3-8 Software errors, record 0 (continued)**

| GICT_ERR<n>STATUS.IERR (Syndrome) | GICT_ERR<n>STATUS.SERR | GICT_ERR<n>MISC0.Data<br>Description (other bits RES0)[f] | Recovery, Prevention |
|---|---|---|---|
| `0x1A`, SYN_SPI_NO_DEST_TGT<br>A SPI has no legal target destinations. | `0xF` | ID, bits[9:0] | Before enabling the specified SPI, reprogram the SPI to target an existing core.<br>**Note**<br>The same SPI might repeat this error several times and cause an overflow. |
| `0x1B`, SYN_SPI_NO_DEST_1OFN<br>A 1 of N SPI cannot be delivered due to bad DPG/GICR_CLASS programming. | `0xF` | ID, bits[9:0] | Ensure that there is at least one valid target for the specified 1 of N interrupt, that is, ensure that at least one core has acceptable DPG and CLASS settings to enable delivery.<br>**Note**<br>The same SPI might repeat this error several times and cause an overflow. |
| `0x1C`, SYN_COL_OOR<br>A collator message is received for a non-implemented SPI, or is larger than the number of owned SPIs in a multichip configuration. | `0xF` | ID, bits[9:0] | In a multichip configuration, ensure that there are enough owned SPIs to support all SPI wires that are used. Any unsupported interrupts must be disabled at the source. |
| `0x1D`, SYN_DEACT_IN<br>A Deactivate to a non-existent SPI, or with incorrect groups set. Deactivates to LPI and non-existent PPI are not reported. | `0xE` | None | A Deactivate occurred to a non-existent SPI, or that SPI group prevented the Deactivate occurring. Software must check the active states of SPIs. |
| `0x1E`, SYN_SPI_CHIP_OFFLINE<br>An attempt was made to send a SPI to an offline chip. | `0xF` | ID, bits[9:0] | Software must disable or retarget interrupts that are targeted at offline cores. |
| `0x28`, SYN_ITS_REG_SET_OOR<br>An attempt was made to set an *Out-of-Range* (OOR) interrupt. Only valid when GICR LPI injection registers are supported. | `0xE` | Core, bits[24:16]<br>Data, bits[15:0] | Software must reprogram the source device to only create legal LPI IDs. |

[f] Always packed from 0 (lowest = 0).

**Table 3-8 Software errors, record 0 (continued)**

| GICT_ERR<n>STATUS.IERR (Syndrome) | GICT_ERR<n>STATUS.SERR | GICT_ERR<n>MISC0.Data<br>Description (other bits RES0)[f] | Recovery, Prevention |
|---|---|---|---|
| 0x29, SYN_ITS_REG_CLR _OOR<br>An attempt was made to clear an OOR interrupt. Only valid when GICR LPI injection registers are supported. | 0xE | Core, bits[24:16]<br>Data, bits[15:0] | Software must not attempt to clear non-existent LPIs. |
| 0x2A, SYN_ITS_REG_INV_OOR<br>An attempt was made to invalidate an OOR interrupt. Only valid when GICR LPI injection registers are supported. | 0xE | Core, bits[24:16]<br>Data, bits[15:0] | Software must not attempt to clear non-existent LPIs. |
| 0x2B, SYN_ITS_REG_SET_ENB<br>An attempt was made to set an interrupt when LPIs are not enabled. Only valid when GICR LPI injection registers are supported. | 0xF | Core, bits[24:16]<br>Data, bits[15:0] | Software must follow architectural steps to enable LPIs on the specified core before enabling the core to send interrupts. |
| 0x2C, SYN_ITS_REG_CLR _ENB<br>An attempt was made to clear an interrupt when LPIs are not enabled. Only valid when GICR LPI injection registers are supported. | 0xF | Core, bits[24:16]<br>Data, bits[15:0] | Software must not try to clear LPIs on a core that does not have LPIs enabled using GICR_CTLR.Enable_LPIs. |
| 0x2D, SYN_ITS_REG_INV_ENB<br>An attempt was made to invalidate an interrupt when LPIs are not enabled. Only valid when GICR LPI injection registers are supported. | 0xF | Core, bits[24:16]<br>Data, bits[15:0] | Software must not try to invalidate LPIs on a core that does not have LPIs enabled using GICR_CTLR.Enable_LPIs. |
| 0x40, SYN_LPI_PROP_READ_FAIL<br>An attempt was made to read properties for a single interrupt, where an error response was received with the data. | 0x12 | Target, bits[31:16]<br>ID, bits[15:0] | Software must reprogram the LPI Property table for the specified ID with error-free data and then issue an INV command through the ITS. If an overflow occurred, an INVALL command must be issued to all cores. |
| 0x41, SYN_PT_PROP_READ_FAIL<br>An attempt was made to read properties for a block of interrupts, where an error response was received with the data. | 0x12 | Target, bits[31:16]<br>ID, bits[15:0] | Software must reprogram the LPI Property table for the specified ID with error-free data and then issue an INV command through the ITS. If an overflow occurred, an INVALL command must be issued to all cores. |

[f] Always packed from 0 (lowest = 0).

**Table 3-8 Software errors, record 0 (continued)**

| GICT_ERR<n>STATUS.IERR (Syndrome) | GICT_ERR<n>STATUS.SERR | GICT_ERR<n>MISC0.Data<br>Description (other bits RES0)[f] | Recovery, Prevention |
|---|---|---|---|
| 0x42, SYN_PT_COARSE_MAP_READ_FAIL<br>An attempt was made to read the coarse map for a target, where an error response was received with the data. | 0x12 | Target, bits[31:16] | No recovery is necessary because the GIC assumes that the coarse map is full. |
| 0x43, SYN_PT_COARSE_MAP_WRITE_FAIL<br>An attempt was made to write the coarse map for a target, with an error received with the Write response. | 0x12 | Target, bits[31:16] | The GIC attempts to continue, however this error indicates issues with the memory system, and operation might be UNPREDICTABLE. |
| 0x44, SYN_PT_TABLE_READ_FAIL<br>An attempt was made to read a block of interrupts from a Pending table, where an error response was received with the data. | 0x12 | Target, bits[31:16]<br>ID, bits[15:0] | Software must determine the reason for the pending error Read fail. The GIC uses the data supplied, however, it is possible for the LPI interrupt to be lost around the specified LPI. |
| 0x45, SYN_PT_TABLE_WRITE_FAIL<br>An attempt was made to write back a block of interrupts from a Pending table, with an error received with the write response. | 0x12 | Target, bits[31:16]<br>ID, bits[15:0] | The GIC tries to continue, however, this error indicates issues with the memory system, and operation might be UNPREDICTABLE. |
| 0x46, SYN_PT_SUB_TABLE_READ_FAIL<br>An attempt was made to read a sub-block of interrupts from a Pending table, where an error response was received with the data. | 0x12 | Target, bits[31:16]<br>ID, bits[15:0] | Software must determine the reason for the pending error read fail. The GIC uses the data supplied, however, it is possible for the LPI interrupt to be lost around the specified LPI. |
| 0x47, SYN_PT_TABLE_WRITE_FAIL_BYTE<br>An attempt was made to write back a sub-block of interrupts from a Pending table, with an error received with the write response. | 0x12 | Target, bits[31:16]<br>ID, bits[15:0] | The GIC tries to continue, however, this error indicates issues with the memory system, and operation might be UNPREDICTABLE. |

### SPI RAM error records 1-2

SPI RAM error record 1 records RAM ECC errors that are correctable. SPI RAM error record 2 records RAM ECC errors that are uncorrectable.

SPI RAM error records 1-2 are present if SPI RAM ECC is configured.

[f] Always packed from 0 (lowest = 0).

The GIC-600 has two SPI RAM, SPI0 and SPI1 that contain the programming for SPIs. SPI0 contains SPIs that have even-numbered IDs, and SPI1 contains SPIs that have odd-numbered IDs.

If a correctable error is detected in SPI RAM, it is corrected and the error is reported in error record 1. See *3.16.5 Error recovery and fault handling interrupts* on page 3-76 for information about the error counters and interrupt generation options.

Correctable errors do not require software to take any action within the GIC. However, software can choose to track error locations in case a RAM row or column can be repaired, if the RAM has repair capability.

The GICT_ERR1MISC0 reports data for SPI error records 1-2 shown in the following table.

**Table 3-9 SPI RAM errors, records 1-2**

| Record | GICT_ERR1MISC0.Data |
|---|---|
| 1 = Correctable | Bit location, ID, bits[$\log_2$(SPIs)+] |
| 2 = Uncorrectable | ID, bits[$\log_2$(SPIs) - 1:0] |

The RAM address can be determined from the ID >> 1. ID[0] specifies the SPI RAM number.

If a SPI has an uncorrectable error, GICD_IERRRn identifies the SPI. While in this error state, the interrupt reverts to a disabled, Secure group 0, edge-triggered SPI, and Non-secure access is controlled by GICD_FCTLR.NSACR. This enables Secure software to control whether Non-secure accesses can set the interrupt to pending while in the errored state.

For uncorrectable errors, software is required to perform the following recovery sequence:

1. Read the error record to determine if an uncorrectable error has occurred.
2. Clear the error record to enable future errors to be tracked.
3. Read all GICD_IERRRn registers to identify SPIs that have errors. The GICD_IERRRn registers must be read from the Secure side.[g]
4. If required, read out any of the current programmed states. This includes programmed data that is corrupted and generates an error, unless GICT_ERR0CTRL.UE is disabled. Arm recommends that intended programming is stored in memory so that this step is not required.
5. Write to GICD_ICENABLERn to disable all interrupts that have errors.
6. Write 1 to GICD_IERRRn to clear all interrupts that have errors and revert GICD_IGROUPRn, GICD_IGRPMODRn, GICD_ICFGRn and GICD_NSACR to their default values.
7. Read GICD_IERRRn to ensure that the error has cleared. If the error has not cleared, clear all the GICD_CTLR group enables to force all SPIs to be returned to their owner chips. Repeat the write to GICD_IERRRn when GICD_CTLR.RWP has returned to 0. When the error clear is accepted, the group enables can be re-enabled.
8. Reprogram the interrupt to the intended settings.
9. If the interrupt is reprogrammed to be level-sensitive, write to GICD_ICPENDRn to ensure that any edge-sensitive pending bits are cleared.
10. If the interrupt is edge-triggered, Arm recommends that software checks the device, if possible, in case an edge is lost.
11. Ensure that the active bit is set correctly depending on whether it is being processed. Clear the active bit using GICD_ICACTIVE to ensure that the interrupt is delivered when it is set to pending in the future. However, if the interrupt is being processed in a core, the interrupt might be delivered again before it is deactivated.
12. Re-enable the reprogrammed interrupts by writing to GICD_ISENABLER.
13. Recheck the error record to ensure that no more errors are reported, if necessary, repeat step *2*.

[g] If the error record reports only one error, the block that contains the error can be determined using the ID in the GICT_ERR2MISC0 register, by calculating the block number as 1 + (ID / 32). However, in the case of an overflow, all GICD_IERRRn registers must be checked.

### SGI RAM error records 3-4

SGI RAM error record 3 records RAM ECC errors that are correctable. SGI RAM error record 4 records RAM ECC errors that are uncorrectable.

SGI RAM error records 3-4 are present if SGI RAM ECC is configured.

The Distributor records a subset of the SGI programming, and stores this information in the SGI RAM, to ensure that it can make the correct routing decisions for SGIs.

If a correctable error is detected in SGI RAM, the error is corrected and the error is reported in error record 3. See *3.16.5 Error recovery and fault handling interrupts* on page 3-76 for information about the error counters and interrupt generation options.

Correctable errors do not require software to take an action within the GIC. However, the GIC can choose to track error locations in case it can repair a RAM row or column.

The GICT_ERR<n>MISC0 reports data for SGI error records 3-4 shown in the following table.

**Table 3-10 SGI RAM errors, records 3-4**

| Record | GICT_ERR<n>MISC0.Data |
|---|---|
| 3 = Correctable | Bit location, $\log_2$(width).<br>Address, bits[(ceiling(cores / 16) × 16) − 1:0]. |
| 4 = Uncorrectable | Address, bits[(ceiling(cores / 16) × 16) − 1:0]. |

The RAM stores information for the same SGI for up to 16 cores on a single row.

GICR_SGIDR contains default values for GICR_IGROUPR0, GICR_IGRPMODR0, and GICR_NSACR for each SGI.

When SGI programming that is in error, is taken from GICR_SGIDR, the corrupted SGI number is given by address × 16 on cores (address - (address × 16)) to (address - (address × 16)) + 15.

For uncorrectable errors that occur in either the PPI or SGI RAM, software is required to perform the following recovery sequence:

1. Read the error record to determine if an uncorrectable error has occurred.
2. Clear the error record to enable future errors to be tracked.
3. Read all GICR_IERRVR registers to identify SGIs and PPIs that have errors. The GICR_IERRVR registers must be read from the Secure side.
4. If required, read out any of the current programmed states.[h] This includes programmed data that is corrupted and generates an error, unless GICT_ERR0CTRL.UE is disabled. Arm recommends that intended programming is stored in memory so that this step is not required.
5. Write to GICR_ICENABLER0, to disable all interrupts that have errors.
6. Write 1 to the interrupts that have errors in the relevant GICR_IERRVR register to clear the interrupts that have errors and revert GICR_IGROUPR0, GICR_IGRPMODR0, and GICR_NSACR to their default values. The values of PPIs are not changed.
7. Reprogram the interrupt to the intended settings.
8. Re-enable the reprogrammed interrupts by writing to the relevant GICR_ISENABLER0.
9. Recheck the error record to ensure that no more errors are reported, if necessary, repeat step *2*.

### PPI RAM error records 7-8

PPI RAM error record 7 records RAM ECC errors that are correctable. PPI RAM error record 8 records RAM ECC errors that are uncorrectable.

PPI RAM error records 7-8 are present if PPI RAM ECC is configured.

[h] The GICR_NSACR is overwritten when an error occurs, therefore, the pre-error value cannot be read back at this stage.

Error records 7-8 record the errors from PPI RAM that contain GICR_IGROUPR0, GICR_IGRPMODR0, and GICR_IPRIORITYRn information for PPIs and SGIs. PPI RAM also contains a buffer that stores generated SGIs when backpressure occurs.

The GICT_ERR<n>MISC0 reports data for PPI error records 7-8 shown in the following table.

**Table 3-11 PPI RAM errors, records 7-8**

| Record | GICT_ERR<n>MISC0.Data |
|---|---|
| 7 = Correctable | PPI block, bits[18+].<br>Bit location, bits[17:12].<br>Offset, bits[11:8].<br>SGI/Int, bit[7].<br>Core, bits[6:0]. |
| 8 = Uncorrectable | PPI block, bits[12+].<br>Offset, bits[11:8].<br>SGI/Int, bit[7].<br>Core, bits[6:0]. |

For uncorrectable errors, software must perform the recovery sequence that is described in *SGI RAM error records 3-4* on page 3-86.

### LPI RAM error records 9-10

LPI RAM error record 9 records RAM ECC errors that are correctable. LPI RAM error record 10 records RAM ECC errors that are uncorrectable. Each error generates an LPI interrupt.

LPI RAM error records 9-10 are present if LPI RAM ECC is configured.

Error records 9-10 record errors from the main LPI cache.

The GICT_ERR<n>MISC0 reports data for LPI error records 9-10 shown in the following table.

**Table 3-12 LPI RAM errors, records 9-10**

| Record | GICT_ERR<n>MISC0.Data |
|---|---|
| 9 = Correctable | Bit location, bits[15+].<br>Reserved, bit[14].<br>Pending[i], bits[13:12]<br>Reserved, bits[11:10].<br>Address, bits[9:0]. |
| 10 = Uncorrectable | Pending, bits[13:12].<br>Reserved, bits[11:10].<br>Address, bits[9:0]. |

When an uncorrectable error occurs, the data shown in the table is stored and the GICT_ERR10MISC1 register is updated to contain the RAM contents of the corrupted line. The line in RAM is dropped, and any pending interrupts that it might contain are lost.

[i] Pending bits[13:12] indicate if there were pending interrupts in the cache at the time of the corruption.

If required, software can use the data in the GICT_ERR10MISC1 register to check several interrupt sources, such as the corrupted INTID. This ID is never more than two bits away from the recorded ID.

### ITS RAM error records 11-12

ITS RAM error record 11 records ITS RAM ECC errors that are correctable. ITS RAM error record 12 records ITS RAM ECC errors that are uncorrectable.

ITS RAM error records 11-12 are present if an ITS is configured.

Error records 11-12 record the errors from ITS RAM.

All ITS tables are memory backed allowing uncorrectable errors to be read from RAM again without software intervention. These records are used for tracking RAM errors and for possible RAM maintenance.

The GICT_ERR<n>MISC0 reports data for ITS RAM error records 11-12 shown in the following table.

**Table 3-13 ITS RAM errors, records 11-12**

| Record | GICT_ERR<n>MISC0.Data |
|---|---|
| 11 = Correctable | Bit location, bits[(x + 15)+].<br>Address, bits[(x + 14)+].<br>RAM, bits[x + 2:x].<br>ITS, bits[x - 1:0].<br>$x = \log_2(ITS)$. |
| 12 = Uncorrectable | Address, bits[(x + 3)+].<br>RAM, bits[x + 2:x].<br>ITS, bits[x - 1:0].<br>$x = \log_2(ITS)$. |

GICT_ERR<n>MISC0 gives information relating to the corrupted ITS, RAM, and RAM address. The bit location of a correctable error is also given. The ITS RAM encoding is shown in the following table.

**Table 3-14 ITS RAM encoding**

| RAM | Record 11 | Record 12 |
|---|---|---|
| 0 | None | None |
| 1 | Device cache | Device cache |
| 2 | Collection cache | Collection cache |
| 3 | Event cache | Event cache |
| 4 | - | Reserved |
| 5 | - | Reserved |
| 6 | - | Reserved |
| 7 | - | Event cache, locked |

### ITS command and translation error records 13+

The ITS command and translation error records 13+ record uncorrectable command and translation errors from each configured ITS.

The ITS command and translation error records capture software events so that the operation of software can be tracked.

The ITS command and translation error records capture software events so that the operation of software can be tracked. The software command errors that are captured are uncorrectable errors only, which require software to correct the command to restart.

The GICT_ERR<n>STATUS.IERR field indicates whether an error is either related to the architecture (0) or is IMPLEMENTATION DEFINED (1). In both cases, the full 24-bit syndrome is reported in GICT_ERR<n>MISC0. Extra data is reported in GICT_ERR<n>MISC1.

The data that is captured for each ITS software syndrome is shown in the following table.

**Table 3-15 ITS command and translation errors, records 13+**

| Error mnemonic | Encoding | IERR | Stall | Mask | Description |
|---|---|---|---|---|---|
| MOVALL_TGT_OOR | 0x10E20 | 1 | 0 | - | MOVALL from a core that does not exist.<br>Command is ignored. |
| MOVALL_DST_TGT_OOR | 0x10E21 | 1 | 0 | - | MOVALL to a core that does not exist.<br>Command is ignored. |
| MOVALL_CHIP_OFFLINE_OOR | 0x10E22 | 1 | 0 | - | MOVALL to a chip that is out-of-range, or from a chip that is offline.<br>Command is ignored. |
| MOVALL_ENABLE_LPI_OFF | 0x10E23 | 1 | 0 | - | MOVALL from a core where GICR_CTLR.Enable_LPIs is 0.<br>Command is ignored. |
| MOVALL_DST_ENABLE_LPI_OFF | 0x10E24 | 1 | 0 | - | MOVALL to a core where GICR_CTLR.Enable_LPIs is 0, or to a destination chip that is offline.<br>LPIs on MOVALL source are dropped. |
| INT_PHYSICALID_OOR | 0x10326 | 1 | 0 | - | INT received with a physical ID that is beyond the range that is specified in GICR_PROPBASER.IDbits.<br>Software must correct mappings.<br>Interrupt is dropped and ID is reported in GICT_ERR<n>MISC1. |
| INT_TGT_OOR | 0x10320 | 1 | 0 | - | INT received for a core that does not exist.<br>Software must correct mappings.<br>Interrupt is dropped and TGT is reported in GICT_ERR<n>MISC1. |
| INT_CHIP_OFFLINE_OOR | 0x10322 | 1 | 0 | - | INT received for a chip that is offline.<br>Software must either correct mappings or take chip online.<br>Interrupt is dropped and TGT is reported in GICT_ERR<n>MISC1. |

**Table 3-15 ITS command and translation errors, records 13+ (continued)**

| Error mnemonic | Encoding | IERR | Stall | Mask | Description |
|---|---|---|---|---|---|
| INT_LPI_OFF | 0x10323 | 1 | 0 | - | INT received for TGT with GICR_CTLR.Enable_LPIs disabled.<br>Software must either enable LPI or correct mappings.<br>TGT is reported in GICT_ERR<n>MISC1. |
| MAPD_DEVICE_OOR | 0x10801 | 0 | 1 | CEE | A MAPD command has tried to map a device with a DeviceID that is outside the supported range, or that is beyond the memory allocated. |
| MAPD_ITTSIZE_OOR | 0x10802 | 0 | 1 | CEE | A command has tried to allocate an ITT table that is larger than the supported EventID size. |
| MAPC_COLLECTION_OOR | 0x10903 | 0 | 1 | CEE | A MAPC command has tried to map a CollectionID that is not supported. See *GITS_TYPER* on page 4-143. |
| MAPC_TGT_OOR | 0x10920 | 1 | 1/0 | CEE | A MAPC command has tried to map to a core that does not exist.<br>If the core is within the maximum range that the ITS supports, the command stalls.<br>If the command is detected in the destination Distributor, the command is ignored and the core is reported in GICT_ERR<n>MISC1.<br>**Note**<br>If the value in GICT_ERR<n>MISC1 is 0, the location of the detected error is in the ITS.<br>CEE applies to errors detected in the ITS only. |
| MAPC_LPI_OFF | 0x10923 | 1 | 0 | - | A MAPC command has tried to map a collection to a core that does not have LPIs enabled.<br>Software must correct the mapping, or it must first enable LPIs using GICR_CTLR.Enable_LPIs.<br>The core is reported in GICT_ERR<n>MISC1. |
| MAPC_CHIP_OFFLINE_OOR | 0x10922 | 1 | 0 | - | A MAPC command has targeted a core in an offline chip.<br>Software must correct the mapping or take the target chip online. |
| MAPI_DEVICE_OOR | 0x10B01 | 0 | 1 | CEE | A MAPI has tried to map a DeviceID that is not supported.<br>See GITS_BASER0[j], and for information about the supported range, see *GITS_TYPER* on page 4-143. |

[j] The *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0* describes this register.

**Table 3-15 ITS command and translation errors, records 13+ (continued)**

| Error mnemonic | Encoding | IERR | Stall | Mask | Description |
|---|---|---|---|---|---|
| MAPI_COLLECTION_OOR | 0x10B03 | 0 | 1 | CEE | A MAPI has tried to map to a collection that is not supported.<br>See GITS_BASER1[j], and for information about the supported range, see *GITS_TYPER* on page 4-143. |
| MAPI_ID_OOR | 0x10B05 | 0 | 1 | CEE | A MAPI has tried to map to an EventID size that is not supported.<br>The size that is supported is reported in GITS_TYPER, but might be reduced depending on the MAPD command for the specified DeviceID. |
| MAPI_UNMAPPED_DEVICE | 0x10B04 | 0 | 1 | CEE | A MAPI has tried to map an interrupt to a device that is not mapped. |
| MAPVI_DEVICE_OOR | 0x10A01 | 0 | 1 | CEE | A MAPVI has tried to map a device supported by the ITS that is out-of-range.<br>See GITS_BASER0[j], and for information about the supported range, see *GITS_TYPER* on page 4-143. |
| MAPVI_COLLECTION_OOR | 0x10A03 | 0 | 1 | CEE | A MAPVI has tried to map to a collection that is outside the range that the ITS supports.<br>See GITS_BASER1[j], and for information about the supported range, see *GITS_TYPER* on page 4-143. |
| MAPVI_UNMAPPED_DEVICE | 0x10A04 | 0 | 1 | CEE | A MAPVI has tried to map an interrupt to a device that is not mapped. |
| MAPVI_ID_OOR | 0x10A05 | 0 | 1 | CEE | A MAPVI has tried to use an EventID that is outside the size that the corresponding MAPD command supports. |
| MAPVI_PHYSICALID_OOR | 0x10A06 | 0 | 1 | CEE | A MAPVI is received that has a physical ID outside the range supported.<br>The supported range is >16-<8096 bits. |
| MOVI_DEVICE_OOR | 0x10101 | 0 | 1 | CEE | A MAPVI has tried to map a device that is outside the range that the ITS supports.<br>See GITS_BASER0[j], and for information about the supported range, see *GITS_TYPER* on page 4-143. |
| MOVI_COLLECTION_OOR | 0x10103 | 0 | 1 | CEE | A MOVI has tried to use a collection that is outside the range that the ITS supports.<br>See GITS_BASER1[j], and for information about the supported range, see *GITS_TYPER* on page 4-143. |
| MOVI_UNMAPPED_DEVICE | 0x10104 | 0 | 1 | CEE | A MOVI has tried to move an interrupt from a device that is not mapped. |
| MOVI_ID_OOR | 0x10105 | 0 | 1 | CEE | A MOVI has tried to use an EventID that is outside the size that the corresponding MAPD command supports. |
| MOVI_UNMAPPED_INTERRUPT | 0x10107 | 0 | 1 | CEE | A MOVI command has tried to operate on an interrupt that is not mapped. |

**Table 3-15 ITS command and translation errors, records 13+ (continued)**

| Error mnemonic | Encoding | IERR | Stall | Mask | Description |
|---|---|---|---|---|---|
| MOVI_UNMAPPED_COLLECTION | 0x10109 | 0 | 1 | CEE | A MOVI command has tried to operate on a collection that is not mapped. |
| DISCARD_DEVICE_OOR | 0x10F01 | 0 | 1 | CEE | A DISCARD has tried to use a device that is outside the range that the ITS supports.<br>See GITS_BASER0[j], and for information about the supported range, see *GITS_TYPER* on page 4-143. |
| DISCARD_UNMAPPED_DEVICE | 0x10F04 | 0 | 1 | CEE | A DISCARD has tried to drop an interrupt from a device that is not mapped. |
| DISCARD_ID_OOR | 0x10F05 | 0 | 1 | CEE | A DISCARD command has tried to use an EventID that is outside the size that the corresponding MAPD command supports. |
| DISCARD_UNMAPPED_INTERRUPT | 0x10F07 | 0 | 1 | CEE | A MOVI command has tried to operate on an interrupt that is not mapped. |
| DISCARD_ITE_INVALID | 0x10F10 | 0 | 1 | CEE | A MOVI command has tried to operate on an EventID that is not supported by the corresponding MAPD command. |
| INV_DEVICE_OOR | 0x10C01 | 0 | 1 | CEE | An INV has tried to use a device that is outside the range that the ITS supports.<br>See GITS_BASER0[j], and for information about the supported range, see *GITS_TYPER* on page 4-143. |
| INV_UNMAPPED_DEVICE | 0x10C04 | 0 | 1 | CEE | An INV has tried to invalidate an interrupt from a device that is not mapped. |
| INV_ID_OOR | 0x10C05 | 0 | 1 | CEE | An INV has tried to use an EventID that is outside the size that the corresponding MAPD command supports. |
| INV_UNMAPPED_INTERRUPT | 0x10C07 | 0 | 1 | CEE | An INV has tried to invalidate an interrupt that is not mapped. |
| INV_ITE_INVALID | 0x10C10 | 0 | 1 | CEE | An INV has tried to invalidate an interrupt with an EventID that is invalid. |
| INV_PHYSICALID_OOR | 0x10C26 | 1 | 1 | CEE | An INV has tried to invalidate an interrupt with a physical ID that is larger than the target supports.<br>See GICR_PROPBASER.IDbits[j]. |
| INV_TGT_OOR | 0x10C20 | 1 | 1 | CEE | An INV has tried to invalidate an interrupt that is mapped to an invalid target. |
| INV_LPI_OFF | 0x10C23 | 1 | 1 | CEE | An INV has tried to invalidate an interrupt that is mapped to a target that does not have LPIs enabled.<br>See GICR_CTLR.Enable_LPIs[j]. |
| INV_CHIP_OFFLINE_OOR | 0x10C22 | 1 | 1 | CEE | An INV has tried to invalidate an interrupt that is mapped to a chip that is offline. |
| INVALL_COLLECTION_OOR | 0x10D03 | 0 | 1 | CEE | An INVALL has tried to invalidate an OOR collection. See *GITS_TYPER* on page 4-143. |

**Table 3-15 ITS command and translation errors, records 13+ (continued)**

| Error mnemonic | Encoding | IERR | Stall | Mask | Description |
|---|---|---|---|---|---|
| INVALL_UNMAPPED_COLLECTION | 0x10D09 | 0 | 1 | CEE | An INVALL has tried to invalidate a collection that is not mapped. |
| INVALL_TGT_OOR | 0x10D20 | 1 | 1 | CEE | An INVALL has been sent to an illegal target. |
| INVALL_LPI_OFF | 0x10D23 | 1 | 1 | CEE | An INVALL has been sent to a target that has LPIs turned off. |
| INVALL_CHIP_OFFLINE_OOR | 0x10D22 | 1 | 1 | CEE | An INVALL has tried to invalidate an interrupt from a device that is not mapped. |
| INT_DEVICE_OOR | 0x10301 | 0 | 1 | UEE | An incoming translation has attempted to use a device that is outside the range that the ITS supports.<br>See GITS_BASER0[i], and for information about the supported range, see *GITS_TYPER* on page 4-143. |
| INT_UNMAPPED_DEVICE | 0x10304 | 0 | 1 | UEE | An incoming translation has tried to invalidate an interrupt from a device that is not mapped. |
| INT_ID_OOR | 0x10305 | 0 | 1 | UEE | An INT has tried to use an EventID that is outside the size that the corresponding MAPD command supports. |
| INT_UNMAPPED_INTERRUPT | 0x10307 | 0 | 1 | UEE | An INT command has tried to raise an interrupt that is not mapped. |
| INT_ITE_INVALID | 0x10310 | 0 | 1 | UEE | An INT command has tried to raise an interrupt with an EventID that is not supported by the corresponding MAPD command. |
| CLEAR_DEVICE_OOR | 0x10501 | 0 | 1 | CEE | A CLEAR has attempted to use a device that is outside the range that the ITS supports.<br>See GITS_BASER0[i], and for information about the supported range, see *GITS_TYPER* on page 4-143. |
| CLEAR_UNMAPPED_DEVICE | 0x10504 | 0 | 1 | CEE | A CLEAR has tried to drop an interrupt from a device that is not mapped. |
| CLEAR_ID_OOR | 0x10505 | 0 | 1 | CEE | A CLEAR has tried to drop an interrupt from an EventID that is not supported by the corresponding MAPD command. |
| CLEAR_UNMAPPED_INTERRUPT | 0x10507 | 0 | 1 | CEE | A CLEAR has attempted to drop an interrupt that is not mapped. |
| CLEAR_ITE_INVALID | 0x10510 | 0 | 1 | CEE | A CLEAR has tried to drop an interrupt from an EventID that is not supported by the corresponding MAPD command. |
| CLEAR_PHYSICALID_OOR | 0x10526 | 1 | 1 | CEE | A CLEAR has tried to drop an interrupt, which has a physical ID that is not supported by the target. |
| CLEAR_TGT_OOR | 0x10520 | 1 | 1 | CEE | A CLEAR has been sent to an illegal target. |
| CLEAR_LPI_OFF | 0x10523 | 1 | 1 | CEE | A CLEAR has been sent to a target that does not have LPIs enabled. |
| CLEAR_CHIP_OFFLINE_OOR | 0x10522 | 1 | 1 | CEE | A CLEAR has been sent to a target on a chip that is offline. |

**Table 3-15 ITS command and translation errors, records 13+ (continued)**

| Error mnemonic | Encoding | IERR | Stall | Mask | Description |
|---|---|---|---|---|---|
| OPR_ DEVICE_OOR | 0x10A01 | 1 | - | - | Software has tried an operation through GITS_OPR using a device that is outside the range that the ITS supports.<br>See GITS_BASER0[i], and for information about the supported range, see *GITS_TYPER* on page 4-143. |
| OPR_ UNMAPPED_COLLECTION | 0x10A03 | 1 | - | - | Software has tried an operation through GITS_OPR using a collection that is outside the range that the ITS supports.<br>See GITS_BASER0[i], and for information about the supported range, see *GITS_TYPER* on page 4-143. |
| OPR_ ID_OOR | 0x10A05 | 1 | - | - | Software has tried to lock an interrupt using an EventID that is larger than the specified device supports.<br>The GITS_OPSR reports a fail. |
| OPR_ UNMAPPED_DEVICE | 0x10A04 | 1 | - | - | Software has tried to lock an interrupt from a device that is not mapped through GITS_OPR.<br>The GITS_OPR reports a fail. |
| OPR_ UNMAPPED_INTERRUPT | 0x10A07 | 1 | - | - | Software has tried to lock an interrupt that is not mapped through GITS_OPR.<br>The GITS_OPSR reports a fail. |
| OPR_ SET_LOCKED | 0x10A10 | 1 | - | - | Software has tried to lock an interrupt into the cache but the set already contains a locked interrupt.<br>The GITS_OPSR reports a fail. |
| INVALID_ML_DEV_TABLE_ENTRY | 0x10B04 | 1 | 1 | CEE | Software is using a two-level Device table and the first-level table entry has not completed.<br>Software must allocate and clear a new second-level table, update the first-level entry, and repeat the command. |
| ACE_LITE_ACCESS_FAILURE | 0x10B01 | 1 | - | - | An access that the ITS issues, receives an SLVERR or DECODE error.<br>The address is given in GICT_ERR<n>MISC1. This error can occur from multiple sources.<br>Software must determine whether the Command queue is stalled, by checking GITS_OPR. If the Command queue has stalled, the command might not have occurred. |

**Table 3-15 ITS command and translation errors, records 13+ (continued)**

| Error mnemonic | Encoding | IERR | Stall | Mask | Description |
|---|---|---|---|---|---|
| ACE_LITE_TRANS_FAILURE | 0x10B03 | 1 | - | AEE | An unknown source in the system has written to the slave port with an access that is not a legal GITS_TRANSLATER access.<br>The full address of the access is given in GICT_ERR<n>MISC1.<br>If the address matches GITS_TRANSLATER, either the size, length, strobes, or access type is wrong.<br>**Note**<br>Read accesses are not tracked. |
| ACE_LITE_ADDR_OOR | 0x10B05 | 1 | - | - | ITS programming has tried to create an access to the address specified in GICT_ERR<n>MISC1 that is larger than the address space supported. |
| INVALID_COMMAND | 0x10F00 | 1 | - | CEE | An Invalid command has been detected in the Command queue.<br>Software must correct this and then resume. |

### Clearing error records

After reading a GICT_ERR<n>STATUS register, software must clear the valid register bits so that any new errors are recorded.

During this period, a new error might overwrite the syndrome for the error that was read previously. If the register is read or written, the previous error is lost.

To prevent this, most bits use a modified version of write-1-to-clear:

- Writes to the ERR<n>STATUS.UE (uncorrectable error records) or ERR<n>STATUS.CE (correctable error records) bits are ignored if ERR<n>STATUS.OF is set and is not being cleared.
- Writes to other fields in the ERR<n>STATUS register are ignored if either ERR<n>STATUS.UE or ERR<n>STATUS.CE are set and are not being cleared.

Similarly, ERR<n>MISC<x> cannot be written, except the counter fields, if the corresponding ERR<n>STATUS.MV bit is set, and ERR<n>ADDR cannot be written if ERR<n>STATUS.AV is set.

Recommended recovery sequences are described for each error record in *Software error record 0* on page 3-78 to *ITS command and translation error records 13+* on page 3-88.

## 3.16.7 Bus errors

ACE-Lite bus error syndromes such as bad transactions, and corrupted RAM data reads can be made to report an ACE-Lite *SLVERR*.

The GICT_ERR0CTLR.UE bit can be used to enable ACE-Lite bus error, *External AXI Slave Error* (*SLVERR*) for the syndromes shown in the following table.

**Table 3-16 Bus error syndromes**

| Syndrome | Description | Direction |
|---|---|---|
| SYN_ACE_BAD | ACE-Lite transactions are either bad or unrecognized | Read and Write |
| SYN_GICR_CORRUPTED | Data read from SPI RAM is corrupted | Read only |

**Table 3-16 Bus error syndromes (continued)**

| Syndrome | Description | Direction |
|---|---|---|
| SYN_GICD_CORRUPTED | Data read from SGI or PPI RAM is corrupted | Read only |
| SYN_ITS_OFF | Access to ITS attempted when powered down | Read and Write |

## 3.17 Multichip operation

You can configure the GIC-600 to support multichip operation.

This section contains the following subsections:


- *3.17.1 About multichip operation* on page 3-97.
- *3.17.2 Connecting the chips* on page 3-98.
- *3.17.3 Changing the Routing table owner* on page 3-99.
- *3.17.4 SPI ownership* on page 3-99.
- *3.17.5 Power control and P-Channel* on page 3-100.
- *3.17.6 Isolating a chip from the system* on page 3-100.
- *3.17.7 SPI operation* on page 3-101.
- *3.17.8 LPIs and the ITS* on page 3-102.


### 3.17.1 About multichip operation

Multichip operation mainly affects the behavior of SPIs.

Systems that comprise more than one chip can have several SoCs that are connected externally, or a SoC comprising several SoCs connected inside a single physical package. In all cases, each SoC is integrated with a GIC-600. A multichip system can have up to 16 chips.

PPIs do not require any specific multichip consideration because they are confined to a single core, and programming occurs through the associated GICR register space on the same chip as the core. SGIs behave the same in multichip and single chip configurations where all addressing is through MPIDR values.

To control the consistency of all chips in the configuration, the GIC-600 uses a set of registers that define the connectivity between chips. These registers are referred to as the Routing table and consist of three register types:

- Chip Registers, GICD_CHIPR<n>.
- Default Chip Register, GICD_DCHIPR.
- Chip Status Register, GICD_CHIPSR.

GICD_CHIPRn defines the Routing table. It specifies the SPIs that the chip owns, and how the chip is accessed. This register exists on each chip in the multichip configuration so that each chip has a copy of the Routing table. The register number <n> corresponds to its chip_ID.

GICD_DCHIPR specifies the current chip that is responsible for the consistency of the Routing table, and indicates when an update is in progress. A single copy of this register exists on each chip in the multichip configuration.

GICD_CHIPSR specifies details of the current status of the chip. A single copy of this register exists on each chip in the multichip configuration.

At reset, each chip in the multichip system configuration is effectively a standalone full-featured GIC. The GICD_CHIPSR register on the chip indicates this state with bit RTS == Disconnected.

For the multichip configuration to be fully coherent, all chips in the configuration must be interconnected and one chip must own the Routing table.

The sequence for connecting chips together is described in *3.17.2 Connecting the chips* on page 3-98.

When multiple chips in the configuration are connected, each set of 32 SPIs (SPI block) is owned by a specific chip, so that the SPI space between chips is partitioned.

**Note**

- SPIs that are not owned by any chip in accordance with the Routing table cannot be used.
- SPI wires on a chip can only be used for SPIs that are owned. However, message-based accesses to SPIs owned on any chip are supported.
- The Routing table can only process one operation at a time. Therefore, software must ensure that GICD_DCHIPR.PUP == 0 before commencing any operation such as writes to GICD_CHIPRx or GICD_DCHIPR.

### 3.17.2 Connecting the chips

To connect the chips in a multichip configuration requires you to follow a procedure.

The procedure for connecting the chips in a multichip configuration is as follows:

#### Procedure

1. Make sure that the values of the **chip_id** tie-off input signals to all chips are correct.
2. Make sure that all Group enables in the GICD_CTLR register are disabled and GICD_CTLR.RWP == 0.
3. Designate a chip, x to own the Routing table.

   You can designate a different chip later if necessary.
4. Set GICD_CHIPRx.ADDR to the value that each chip must use to forward messages to chip x.

   This value is driven by the AXI4-Stream input interface **icdrtdest** signal. Depending on how cross-chip messages are routed, this value can be the chip_id, or a more complex identifier.
5. Set GICD_CHIPRx.SPI_BLOCK_MIN and GICD_CHIPRx.SPI_BLOCKS to appropriate values for the SPIs that chip x owns.

   For example, if the range of interrupt ids for chip x is ID96-ID159:
   - Set SPI_BLOCK_MIN = (96 – 32) / 32 = 2.
   - Set SPI_BLOCKS = (159 – 96 + 1) / 32 = 2.
6. Set GICD_CHIPRx.SocketState = 1.
7. Read GICD_CHIPRx, to check that the writes are successful.

   The writes might fail due to security settings, an overlapping or non-existent SPI, or if another update is still in progress. If the accesses fail, then GICD_CHIPRx.SocketState == 0, indicating that the chip is offline.
8. To check that the actions of this sequence have executed correctly, read the following register fields and check that their values are as shown:
   1. GICD_CHIPSR.RTS == 2 (Consistent).
   2. GICD_DCHIPR.rt_owner == chip x.
   3. GICD_DCHIPR.PUP == 0.

   Chip x is now in the consistent state and ready to accept connections to other chips in the system configuration.

To connect more chips:

9. Set the relevant address and SPI ownership information of the next chip you want to connect to, chip y, by writing to GICD_CHIPRy.

   You can do this through any chip that is already connected, or more efficiently by writing to the chip that owns the Routing table, chip_id == rt_owner.
10. Poll GICD_DCHIPR until bit PUP == 0.

    The connection is complete when PUP == 0.
11. Read GICD_CHIPRy, to check that the write to GICD_CHIPRy is accepted.

12. Repeat steps *9* on page 3-98-*11* on page 3-98 for each chip that you want to connect to any of those already connected.

**Note**

- You must consider that data read from GICD_CHIPRn is valid only when GICD_DCHIPR.PUP == 0, otherwise the data might be updating.
- If you are connecting a new chip, the accesses must be done through a chip that is in the Consistent state and not by writing to the new chip directly.
- If you access GICD_CHIPSR while a chip is being connected, it shows RTS == Updating, register GICD_DCHIPR bit PUP is set, indicating that the Routing table is updating, so the values cannot be trusted.
- Adding or removing a chip when GICD_CTLR group enables are set is unpredictable. To check that group enables are off, software must poll GICD_CTLR.RWP.
- If you are connecting together multiple different instances of the GIC-600, the settings for the following parameters must match in all chips:
  - All affinity widths (`max_affinity_width`*).
  - Number of SPI blocks supported (`spi_blocks`).
  - LPI support type (`lpi_support`).
  - Disable Security settings (`ds_value`).
  - Total number of chips supported (`chip_count`).
  - Chip address width (`chip_addr_width`).
  - Chip affinity select level (`chip_affinity_select_level`).
  - Maximum number of cores on any single chip (`max_pe_on_chip`).
- If any chip in the system has an ITS block, parameter `its_type_support` = `full`, then direct injection LPI registers are not supported.

See the *Arm® CoreLink™ GIC-600 Generic Interrupt Controller Configuration and Integration Manual* for information on configuration parameters and their options.

### 3.17.3 Changing the Routing table owner

You can change the chip that owns the Routing table at any time.

A procedure that describes how to change the owner of the Routing table is as follows.
To change the owner of the Routing table:

1. Write to GICD_DCHIPR.rt_owner, where the value of the rt_owner is the chip_id of the new owner.
2. Poll for GICD_DCHIPR.PUP == 0.

The Routing table owner must be the last chip to be powered down.

### 3.17.4 SPI ownership

The owner of a SPI block is defined by the GICD_CHIPn registers.

You can remove SPI blocks from a chip and add them to another chip by reprogramming the relevant GICD_CHIPn registers during operation. As with all Routing table operations, GICD_DCHIPR.PUP must be polled to check completion of the operation.

Before you change the owner of a SPI block, you must ensure that the GICD_CTLR group enables have cleared, GICD_CTLR.RWP has returned to 0, and that the SPI blocks are removed from a chip before they are added to another chip.

When a SPI block is removed from, or added to, a chip, all programming that is associated with the SPI block returns to the reset state.

——— **Note** ———

You must not alter the SPI_BLOCK_MIN of an online chip because the results are unpredictable. To change SPI_BLOCK_MIN, move the chip offline by setting GICD_CHIPRn.SocketState = 0. You can now alter SPI_BLOCK_MIN and then return the chip to online.

———————————

### 3.17.5 Power control and P-Channel

You can use the P-Channel to isolate a chip from the system.

The P-Channel has three states:

- RUN (**pstate** == `0x0`). This is the normal functional mode.
- CONFIG (**pstate** == `0x9`). In this state, the GIC does not send any cross-chip messages. It accepts incoming messages but does not process them.
- OFF (**pstate** == `0xF`). In this state, the GIC does not send any cross-chip messages and does not accept any incoming messages. The **icrdtready** signal is clamped LOW to prevent accesses entering the GIC.

While in both the CONFIG and OFF states, register accesses that are normally sent to another chip are serviced locally. Therefore, the Routing table registers read the local versions instead of the copies of the Routing table owner. The same is true for SPIs that are owned remotely. Therefore, it is safe to save and restore the Distributor register values in either of these P-Channel states.

You can exit reset in either RUN or OFF states by setting the initial value of the **pstate** signal. If you have saved register values and intend to restore them, you must use the OFF state and restore the Routing table first before attempting to restore any SPI registers.

### 3.17.6 Isolating a chip from the system

You can isolate a chip from the system.

To isolate a chip from the system, use the following procedure:

1. Ensure that all cores on the chip are asleep by setting GICR_WAKER.ProcessorSleep.
2. Ensure all ITS blocks on the chip are disabled and the buses are quiesced by using the **qreqn_its<n>** Q-Channel interfaces.
3. Ensure that LPIs from other chips are not routed to this chip.
4. Attempt to enter the CONFIG state (**pstate** = `0x9`).

   If the GIC is idle and all credits are returned, it accepts the request to go into CONFIG state, otherwise it denies the request and remains in RUN state.

   ——— **Note** ———

   All SPIs must return to their own chip before a request is accepted. This means that SPIs that are enabled and pending, but targeting a core on a remote chip where the relevant CPU group is disabled, prevent transition into the CONFIG state.

   ———————————

   When in the CONFIG state, any cross-chip messages that change the internal state are held in the cross-chip interface, and all messages assert **pactive**. If **pactive** asserts while attempting to enter a lower power state, you must return to RUN (**pstate** == `0x0`).
5. When in CONFIG state, any required state can be saved.

   ——— **Note** ———

   Writing GICD_CHIPR or GICD_DCHIPR for any purpose other than to restore saved values after a hardware reset is unpredictable.

   ———————————

6. Power down the Redistributors using the GICR_PWRR registers.

7. If required, flush the LPI cache using GICR_WAKER.Sleep.

   Arm recommends that if wake-on-interrupt is required, LPIs from other chips do not target this chip while the chip is being powered down (step *3*), and must be routed back while the chip is in the OFF state.

   LPIs that arrive after sleep is set in the CONFIG state are dropped.
8. Attempt to enter the OFF state.

   **Note**

   If **pactive** is raised, return to the CONFIG state.

9. Use the Q-Channel to put the GIC into a safe mode to reset.

   **Note**

   If the SPI_Collator is in a different domain to the Distributor and only one of the domains is being reset, then the Power Q-Channel must have also accepted before the reset can occur. This might require masking interrupts outside of the GIC to ensure that all interrupt lines have reached their idle state.

   Power up is the reverse of the powerdown sequence. However, you must ensure that the Routing table is restored before other registers, else the behavior is unpredictable. Restoring values to the Routing table that are not exactly the same as those read out before a reset, can cause unpredictable behavior.

   **Note**

   Accesses to GICD_CTLR continue to be broadcast to the isolated chip, which requests wakeup.

### 3.17.7 SPI operation

When the Routing table is set up, SPIs can be programmed through any connected chip, and accesses to update stored values are routed over the cross-chip interface of the chip that owns the SPIs.

SPIs can be routed to remote chips by programming the relevant GICD_IROUTER register. Remote chips are targeted using either Affinity2 or Affinity3, and the Affinity level can be discovered using GICD_CFGID.AFSL.

If SPIs within a SPI block are sent to multiple chips, Arm recommends that you do not read or write registers GICD_ISACTIVE, GICD_ICACTIVE, GICD_ISPENDR, and GICD_ICPENDR. It is inefficient and these registers are not needed for immediate operation.

You can set interrupts to pending by writing to GICD_SETSPI_NSR, GICD_CLRSPI_NSR, GICD_SETSPI_SR and GICD_CLRSPI_SR. For efficient operation, Arm recommends that sources are programmed to write SPI IDs that are owned by their chip. Other SPI IDs are supported if these SPIs are owned somewhere in your system.

**Note**

By default, the GIC-600 does not guarantee that the pending bit has reached the point of serialization for writes to set interrupts pending. This means that there is a race between the pending bit being set and an activate being processed by the GIC after the **bresp** signal is asserted. To ensure that writes are always propagated to the point of serialization, write 1 to GICD_FCTLR.POS.

#### SPIs and the Collator

The SPI Collator wires are always connected to the lowest owned SPIs on the chip. If GICD_CHIPRn.SPI_BLOCK_MIN = 4, the SPI Collator wires to chip x drive SPI IDs that start from 160, calculated by (4 × 32) + 32 = 160.

Therefore, in a homogeneous two-chip system, each chip must not use more wires than 16 × (the number of configured SPI blocks).

#### SPI 1 of N

The GIC-600 never sends a 1 of N SPI to another chip.

### 3.17.8 LPIs and the ITS

The GIC-600 uses the value of GICR_TYPER.ProcessorNumber to route all LPIs and commands to their targets.

The GIC-600 does not use physical target addresses, therefore GITS_TYPER.PTA == 0.

The GIC-600 divides the ProcessorNumber value into two fields, Chip_ID, and Padded linear on-chip core number.

The width of the padded on-chip core number is defined by the `max_pe_on_chip` configuration parameter. This parameter sets the maximum number of cores on a single chip in the configuration. The width of the linear on-chip core number field is discoverable through GICD_CFGID.PEW.

For example, if `max_pe_on_chip` = 17, the width of the lower part of the on-chip core number field is ceiling[$\log_2(17)$] = 5. Therefore, the ProcessorNumber value of the first core on chip 1 is `0x20`, the value of the second core on chip 1 is `0x21`, the value of the first core on chip 2 is `0x40`.

The following figure shows the ProcessorNumber fields with typical values.

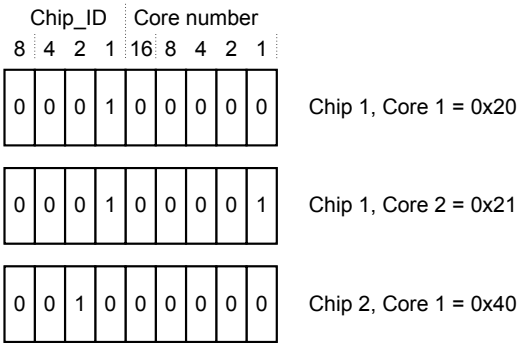


**Figure 3-3 ProcessorNumber fields**

If software attempts to access a chip that does not exist, is offline, or access a core that does not exist, the request is dropped and reported through the ITS command and translation error records 13+.

# Chapter 4
# **Programmers model**

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

It contains the following sections:


- *4.1 The GIC-600 registers* on page 4-104.
- *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.
- *4.3 Distributor registers (GICA) for message-based SPIs summary* on page 4-125.
- *4.4 Redistributor registers for control and physical LPIs summary* on page 4-126.
- *4.5 Redistributor registers for SGIs and PPIs summary* on page 4-135.
- *4.6 ITS control register summary* on page 4-141.
- *4.7 ITS translation register summary* on page 4-151.
- *4.8 GICT register summary* on page 4-152.
- *4.9 GICP register summary* on page 4-170.

## 4.1 The GIC-600 registers

All the GIC-600 registers have names that are constructed of mnemonics that indicate the logical block that the register belongs to and the register function.

The following information applies to the GIC-600 registers:

- The GIC-600 implements only memory-mapped registers.
- The GIC-600 has a single base address, except for the GITS_TRANSLATER register. The base 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 might result in a bus error that is based on GICT_ERR0CTLR.UE.
- 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 zero.
- The GIC-600 ACE-Lite slave port can be 64 bits, 128 bits, or 256 bits wide, depending on the configuration. The *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0* defines the permitted sizes of access.

  **Note**

  The GIC-600 guarantees single-copy atomicity for doubleword accesses.

- The GIC-600 supports data only in little-endian format.
- The access types for the GIC-600 are as follows:

  **RO** Read only.

  **RW** Read and write.

  **WO** Write only, reads return as UNKNOWN.

This section contains the following subsections:

- *4.1.1 Register map pages* on page 4-104.
- *4.1.2 Discovery* on page 4-105.
- *4.1.3 GIC-600 register access and banking* on page 4-105.

### 4.1.1 Register map pages

The register map is separated into several pages.

The register map pages are defined in the following table.

**Table 4-1 Register map pages**

| Offset[x:16] | Page | Description |
|---|---|---|
| 0 | GICD | GICD main page |
| 1 | GICA | GICD message-based interrupts alias |
| 2 | GICT | GIC trace and debug page |
| 3 | GICP | GIC PMU |
| 4 + (ITSnum × 2) | GITSn | ITS address page |
| 5 + (ITSnum × 2) | GITSn translate | ITS translation page |
| 4 + (2 × ITScount) + (RDnum × 2) | GICR (LPI) | GICR LPI registers |
| 5 + (2 × ITScount) + (RDnum × 2) | GICR (SGI) | GICR PPI + SGI registers |
| 4 + (2 × ITScount) + (RDcount × 2) | GICDA | Alias to GICD (page after last GICR page) |

***Related information***

*Arm® Generic Interrupt Controller Architecture Specification, version 3.0 and version 4.0*

### 4.1.2 Discovery

Arm recommends that the operating system is provided with pointers to the start of the Distributor, every ITS, and the first Redistributor page on each chip.

To verify that the pages are of GIC registers, these pointers can be checked against the discovery registers, which start at offset `0xFFD0` for each GIC page. These registers allow discovery of the architecture version and, for GIC-600, whether the page contains the Distributor, ITS, or Redistributor registers. When this information is known, additional information can be obtained from registers specific to each page.

For Redistributors, Arm recommends that you examine GICR_TYPER to determine:

- Whether the implementation has two or four pages per Redistributor that are based on the features implemented. It can be inferred that GIC-600 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 the Redistributor is for, based on affinity values.

This information allows you to iteratively search through all Redistributors in a discovery process.

The GITS_TYPER register in the GIC-600 indicates that you must program the ITS with unique ProcessorNumbers, instead of physical target addresses. The GICR_TYPER contains the unique ProcessorNumber that you must use to reference a Redistributor when programming the ITS.

**Note**

In a multichip configuration, the ProcessorNumber upper bits are derived from the **chip_id** tie-off. Therefore, make sure that the **chip_id** value is defined before the registers are read.

***Related information***

*Arm® GICv3 and GICv4 Software Overview*

### 4.1.3 GIC-600 register access and banking

The GIC-600 uses an access and banking scheme for its registers.

**Note**

For information about the register access and banking scheme, see the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

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 that provides separate Secure and Non-secure copies of the registers. A Secure access to the address accesses the Secure copy of the register. A Non-secure access to the address accesses the Non-secure copy.
- 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, which are only accessible to Secure accesses are *Read-As-Zero* and *Writes Ignored* (RAZ/WI).

***Related information***

*Arm® Generic Interrupt Controller Architecture Specification, version 3.0 and version 4.0*

## 4.2 Distributor registers (GICD/GICDA) summary

The GIC-600 Distributor functions are controlled through the Distributor registers identified with the prefix GICD. The Distributor Alias registers are identified with the prefix GICDA.

The following table lists the Distributor registers in base offset order and provides a reference to the register description that is described in either this book or the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

Address offsets are relative to the Distributor base address defined by the system memory map.

Offsets that are not shown are Reserved and RAZ/WI. Accesses to these offsets might be reported in error record 0 as a SYN_ACE_BAD access.

**Table 4-2 Distributor registers (GICD/GICDA) summary**

| Offset | Name | Type | Width | Reset | Description | Architecture defined? |
|---|---|---|---|---|---|---|
| 0x0000 | GICD_CTLR | RW | 32 | Configuration dependent | *4.2.1 Distributor Control Register, GICD_CTLR* on page 4-109 | Yes |
| 0x0004 | GICD_TYPER | RO | 32 | Configuration dependent | *4.2.2 Interrupt Controller Type Register, GICD_TYPER* on page 4-110 | Yes |
| 0x0008 | GICD_IIDR | RO | 32 | Configuration dependent | *4.2.3 Distributor Implementer Identification Register, GICD_IIDR* on page 4-111 | Yes |
| 0x000C-0x001C | - | - | - | - | Reserved | - |
| 0x0020 | GICD_FCTLR | RW | 32 | 0x0 | *4.2.4 Function Control Register, GICD_FCTLR* on page 4-112 | [k] |
| 0x0024 | GICD_SAC | RW | 32 | Tie-off dependent[l] | *4.2.5 Secure Access Control register, GICD_SAC* on page 4-114 | [k] |
| 0x0028-0x003C | - | - | - | - | Reserved | - |
| 0x0040 | GICD_SETSPI_NSR | WO | 32 | - | Non-secure SPI Set Register | Yes |
| 0x0044 | - | - | - | - | Reserved | - |
| 0x0048 | GICD_CLRSPI_NSR | WO | 32 | - | Non-secure SPI Clear Register | Yes |
| 0x004C | - | - | - | - | Reserved | - |
| 0x0050 | GICD_SETSPI_SR[mn] | WO | 32 | - | Secure SPI Set Register | Yes |
| 0x0054 | - | - | - | - | Reserved | - |
| 0x0058 | GICD_CLRSPI_SR[mn] | WO | 32 | - | Secure SPI Clear Register | Yes |
| 0x005C-0x007C | - | - | - | - | Reserved | - |
| 0x0080-0x00FC | GICD_IGROUPRn[on] | RW | 32 | 0x0 | Interrupt Group Registers | Yes |
| 0x0100-0x017C | GICD_ISENABLERn[o] | RW | 32 | 0x0 | Interrupt Set-Enable Registers | Yes |
| 0x0180-0x01FC | GICD_ICENABLERn[o] | RW | 32 | 0x0 | Interrupt Clear-Enable Registers | Yes |

[k] Microarchitecture defined.
[l] The reset values of GICD_SAC.GICTNS and GICD_SAC.GICPNS are controlled by the **gict_allow_ns** and **gicp_allow_ns** tie-off signals respectively.
[m] The existence of this register depends on the configuration of the GIC-600. If Security support is not included, then this register does not exist.
[n] This register is only accessible from a Secure access.

**Table 4-2 Distributor registers (GICD/GICDA) summary (continued)**

| Offset | Name | Type | Width | Reset | Description | Architecture defined? |
|---|---|---|---|---|---|---|
| 0x0200-0x027C | GICD_ISPENDRn[o] | RW | 32 | SPI wire dependent | Interrupt Set-Pending Registers | Yes |
| 0x0280-0x02FC | GICD_ICPENDRn[o] | RW | 32 | SPI wire dependent | Interrupt Clear-Pending Registers | Yes |
| 0x0300-0x037C | GICD_ISACTIVERn[o] | RW | 32 | 0x0 | Interrupt Set-Active Registers | Yes |
| 0x0380-0x03FC | GICD_ICACTIVERn[o] | RW | 32 | 0x0 | Interrupt Clear-Active Registers | Yes |
| 0x0400-0x07FC | GICD_IPRIORITYRn[p] | RW | 32 | Security dependent | Interrupt Priority Registers | Yes |
| 0x0800-0x0BFC | - | - | - | - | Reserved | - |
| 0x0C00-0x0CFC | GICD_ICFGRn | RW | 32 | 0x0 | Interrupt Configuration Registers | Yes |
| 0x0D00-0x0D7C | GICD_IGRPMODRn | RW | 32 | 0x0 | Interrupt Group Modifier Registers | Yes |
| 0x0D80-0x0DFC | - | - | - | - | Reserved | - |
| 0x0E00-0x0EFC | GICD_NSACRn[mq] | RW | 32 | 0x0 | Non-secure Access Control Registers | Yes |
| 0x0F00-0x60FC | - | - | - | - | Reserved | - |
| 0x6100-0x7FD8 | GICD_IROUTERn[r] | RW | 64 | 0x0080000000 | Interrupt Routing Registers.<br>See the *Arm® GICv3 and GICv4 Software Overview*.<br>**Note**<br>All SPIs are reset with Interrupt_Routing_Mode == 1. The first register is GICD_IROUTER32. | Yes |
| 0x7FDC-0xBFFC | - | - | - | - | Reserved | - |
| 0xC000 | GICD_CHIPSR | RW | 32 | P-Channel dependent | *4.2.6 Chip Status Register, GICD_CHIPSR* on page 4-115 | k |
| 0xC004 | GICD_DCHIPR | RW | 32 | 0x0 | *4.2.7 Default Chip Register, GICD_DCHIPR* on page 4-116 | k |
| 0xC008-0xC080 | GICD_CHIPRn | RW | 64 | 0x0 | *4.2.8 Chip Registers, GICD_CHIPR<n>* on page 4-117. | k |
| 0xC088-0xDFFC | - | - | - | - | Reserved | - |
| 0xE008-0xE0FC | GICD_ICLARn | RW | 32 | 0x0 | The first register is GICD_ICLAR2. *4.2.9 Interrupt Class Registers, GICD_ICLARn* on page 4-118 | k |

[o] The first one of these registers does not exist when affinity routing is enabled.
[p] The first eight of these registers do not exist when affinity routing is enabled.
[q] The first four of these registers do not exist when affinity routing is enabled.
[r] The first 32 of these registers do not exist when affinity routing is enabled.

**Table 4-2 Distributor registers (GICD/GICDA) summary (continued)**

| Offset | Name | Type | Width | Reset | Description | Architecture defined? |
|---|---|---|---|---|---|---|
| 0xE108-0xE17C | GICD_IERRRn | RW | 32 | 0x0 | The first register is GICD_IERRR1. *4.2.10 Interrupt Error Registers, GICD_IERRRn* on page 4-119 | k |
| 0xE180-0xEFFC | - | - | - | - | Reserved | - |
| 0xF000 | GICD_CFGID | RO | 64 | Configuration dependent | *4.2.11 Configuration ID Register, GICD_CFGID* on page 4-120 | k |
| 0xF008-0xFFCC | - | - | - | - | Reserved | - |
| 0xFFD0 | GICD_PIDR4 | RO | 32 | 0x44 | *4.2.12 Peripheral ID4 register, GICD_PIDR4* on page 4-121 | No |
| 0xFFD4 | GICD_PIDR5 | RO | 32 | 0x00 | Peripheral ID 5 Register | No |
| 0xFFD8 | GICD_PIDR6 | RO | 32 | 0x00 | Peripheral ID 6 Register | No |
| 0xFFDC | GICD_PIDR7 | RO | 32 | 0x00 | Peripheral ID 7 Register | No |
| 0xFFE0 | GICD_PIDR0 | RO | 32 | 0x92 | *4.2.16 Peripheral ID0 register, GICD_PIDR0* on page 4-124 | No |
| 0xFFE4 | GICD_PIDR1 | RO | 32 | 0xB4 | *4.2.15 Peripheral ID1 register, GICD_PIDR1* on page 4-123 | No |
| 0xFFE8 | GICD_PIDR2 | RO | 32 | 0x3B | *4.2.14 Peripheral ID2 register, GICD_PIDR2* on page 4-122 | No |
| 0xFFEC | GICD_PIDR3 | RO | 32 | 0x00 | *4.2.13 Peripheral ID3 register, GICD_PIDR3* on page 4-122 | No |
| 0xFFF0 | GICD_CIDR0 | RO | 32 | 0x0D | Component ID 0 Register | No |
| 0xFFF4 | GICD_CIDR1 | RO | 32 | 0xF0 | Component ID 1 Register | No |
| 0xFFF8 | GICD_CIDR2 | RO | 32 | 0x05 | Component ID 2 Register | No |
| 0xFFFC | GICD_CIDR3 | RO | 32 | 0xB1 | Component ID 3 Register | No |

This section contains the following subsections:


- *4.2.1 Distributor Control Register, GICD_CTLR* on page 4-109.
- *4.2.2 Interrupt Controller Type Register, GICD_TYPER* on page 4-110.
- *4.2.3 Distributor Implementer Identification Register, GICD_IIDR* on page 4-111.
- *4.2.4 Function Control Register, GICD_FCTLR* on page 4-112.
- *4.2.5 Secure Access Control register, GICD_SAC* on page 4-114.
- *4.2.6 Chip Status Register, GICD_CHIPSR* on page 4-115.
- *4.2.7 Default Chip Register, GICD_DCHIPR* on page 4-116.
- *4.2.8 Chip Registers, GICD_CHIPR<n>* on page 4-117.
- *4.2.9 Interrupt Class Registers, GICD_ICLARn* on page 4-118.
- *4.2.10 Interrupt Error Registers, GICD_IERRRn* on page 4-119.
- *4.2.11 Configuration ID Register, GICD_CFGID* on page 4-120.
- *4.2.12 Peripheral ID4 register, GICD_PIDR4* on page 4-121.
- *4.2.13 Peripheral ID3 register, GICD_PIDR3* on page 4-122.
- *4.2.14 Peripheral ID2 register, GICD_PIDR2* on page 4-122.
- *4.2.15 Peripheral ID1 register, GICD_PIDR1* on page 4-123.
- *4.2.16 Peripheral ID0 register, GICD_PIDR0* on page 4-124.

### 4.2.1 Distributor Control Register, GICD_CTLR

This register enables interrupts and affinity routing.

The GICD_CTLR characteristics are:

**Usage constraints** The EnableGrp* bits and the RWP bit must be 0 before the DS bit can be updated. A write that sets the DS bit must also set the EnableGrp* bits to 0.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

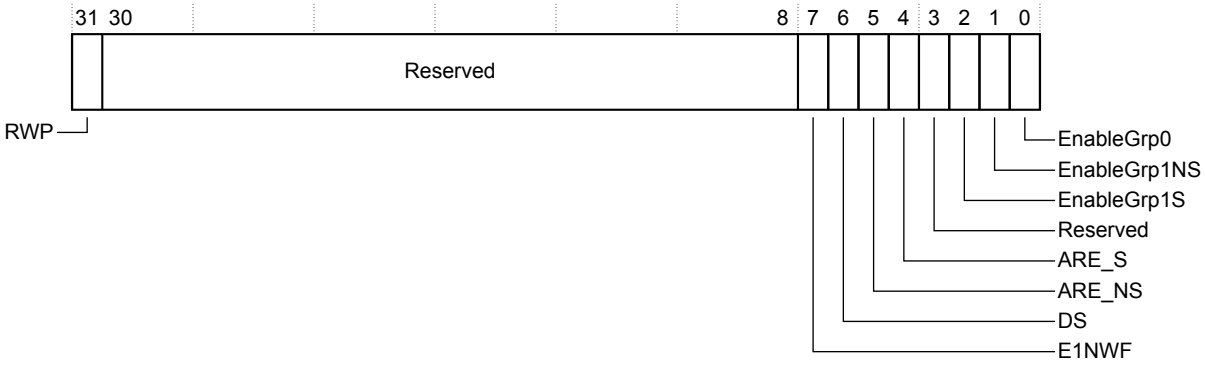


**Figure 4-1 GICD_CTLR bit assignments**

The following table shows the bit assignments.

**Table 4-3 GICD_CTLR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31] | RWP | Register Write Pending.<br>1 = Register write in progress.<br>Resets to 0. This bit is read only. |
| [30:8] | - | Reserved. |
| [7] | E1NWF | Enable 1 of N Wakeup Functionality[s].<br>Resets to 0. |
| [6] | DS | Disable Security[s].<br>Resets to `ds_value` == 1. |
| [5] | ARE_NS | Affinity Routing Enable, Non-secure state[s].<br>Resets to 1. |
| [4] | ARE_S | Affinity Routing Enable, Secure state[s].<br>Resets to 1. |
| [3] | - | Reserved. |
| [2] | EnableGrp1S | Enable Secure Group 1 interrupts[s].<br>Resets to 0. |

[s] See the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*

**Table 4-3 GICD_CTLR bit assignments (continued)**

| Bits | Name | Function |
|---|---|---|
| [1] | EnableGrp1NS | Enable Non-secure Group 1 interrupts[s].<br>Resets to 0. |
| [0] | EnableGrp0 | Enable Group 0 interrupts[s].<br>Resets to 0. |

**Note**

For information about the different Security states for this register, see *Field descriptions*, *GICD_CTLR, Distributor Control Register*, in the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

### 4.2.2 Interrupt Controller Type Register, GICD_TYPER

This register returns information about the configuration of the GIC-600. You can use this register to determine the number of Security states, the number of INTIDs, and the number of processor cores that the GIC supports.

The GICD_TYPER characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

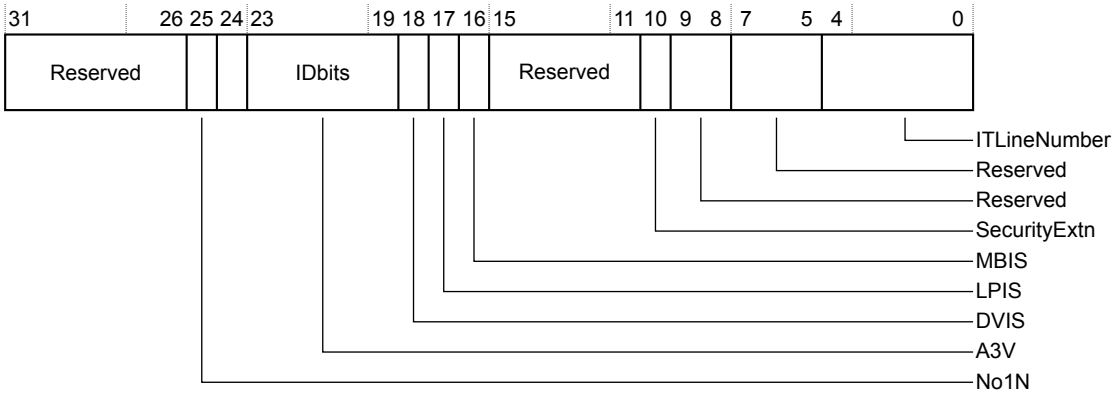


**Figure 4-2 GICD_TYPER bit assignments**

The following table shows the bit assignments.

**Table 4-4 GICD_TYPER bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:26] | - | Reserved, returns `0b000000`. |
| [25] | No1N | 1 of N SPI:<br>0 = The GIC-600 supports 1 of N SPI interrupts. |

**Table 4-4 GICD_TYPER bit assignments (continued)**

| Bits | Name | Function |
|---|---|---|
| [24] | A3V | Affinity level 3 values.<br>Depending on the configuration, returns either:<br>0 = The GIC-600 Distributor only supports zero values of Affinity level 3.<br>1 = The GIC-600 Distributor supports nonzero values of Affinity level 3. |
| [23:19] | IDbits | Interrupt identifier bits:<br>`0b01111` = The GIC-600 supports 16 interrupt identifier bits. |
| [18] | DVIS | Direct Virtual LPI injection support:<br>0 = The GIC-600 does not support Direct Virtual LPI injection.<br>See the *Arm® GICv3 and GICv4 Software Overview*. |
| [17] | LPIS | Indicates whether the implementation supports LPIs.<br>Depending on the configuration, returns either:<br>0 = LPIs are not supported.<br>1 = LPIs are supported. |
| [16] | MBIS | Message-based interrupt support:<br>1 = The GIC-600 supports message-based interrupts. |
| [15:11] | - | Reserved, returns `0b00000`. |
| [10] | SecurityExtn | Security state support.<br>Depending on the configuration, returns either:<br>0 = The GIC-600 supports only a single Security state.<br>1 = The GIC-600 supports two Security states.<br>When GICD_CTLR.DS == 1, this field is RAZ. |
| [9:8] | - | Reserved, returns `0b00`. |
| [7:5] | - | Reserved, returns `0b000`. |
| [4:0] | ITLineNumber | Number of SPIs divided by 32.<br>Returns the number of SPIs divided by 32. The permitted values for this field are 0-30 (992 SPIs maximum). |

### 4.2.3 Distributor Implementer Identification Register, GICD_IIDR

This register provides information about the implementer and revision of the Distributor.

The GICD_IIDR characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

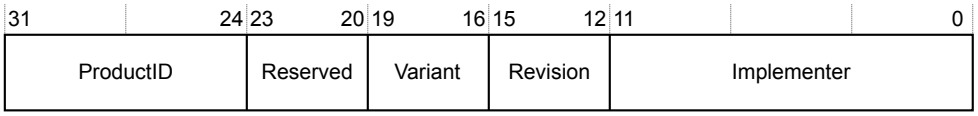


**Figure 4-3 GICD_IIDR bit assignments**

The following table shows the bit assignments.

**Table 4-5 GICD_IIDR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:24] | ProductID | Indicates the product ID:<br>`0x2` = GIC-600. |
| [23:20] | - | Reserved, RAZ. |
| [19:16] | Variant | Indicates the major revision or variant of the product for the `rmpn` identifier:<br>`0x0` = r0.<br>`0x1` = r1. |
| [15:12] | Revision | Indicates the minor revision of the product for the `rmpn` identifier:<br>`0x1` = p0.<br>`0x3` = p1.<br>`0x4` = p2.<br>`0x5` = p3. |
| [11:0] | Implementer | Identifies the implementer:<br>`0x43B` = Arm. |

### 4.2.4 Function Control Register, GICD_FCTLR

This register controls the scrubbing of all RAMs in the local Distributor. The register is not distributed and only acts on the local chip.

The GICD_FCTLR characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

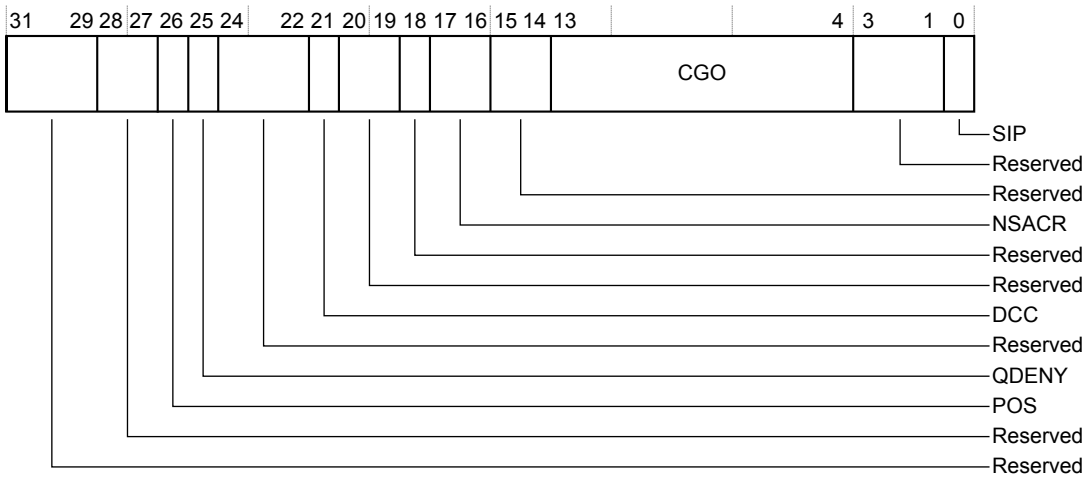


**Figure 4-4 GICD_FCTLR bit assignments**

The following table shows the bit assignments.

**Table 4-6 GICD_FCTLR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:29] | - | Reserved, returns `0b00000`. |
| [28:27] | - | Reserved.<br>RES0. |
| [26] | POS | Point Of Serialization.<br>When an interrupt is sent remotely and POS is set, it ensures that writes to GICD_SETSPI and GICD_CLRSPI propagate to remote chips before ACE-Lite sends a response. Applies only to edge-triggered interrupts.<br>1 = Propagate access to POS.<br>0 = Store locally and propagate when possible.<br>Resets to `0b0`. |
| [25] | QDENY | Q-Channel Deny.<br>Overrides the Q-Channel logic and forces the Distributor to reject powerdown requests. |
| [24:22] | - | Reserved.<br>RES0. |
| [21] | DCC | Do not Correct Cache.<br>Modifies **a<x>cache** outputs from the Distributor.<br>See *3.12 Memory access and attributes* on page 3-69. |
| [20:19] | - | Reserved.<br>RES0. |
| [18] | - | Reserved. |

**Table 4-6 GICD_FCTLR bit assignments (continued)**

| Bits | Name | Function |
|---|---|---|
| [17:16] | NSACR | Non-secure Access Control. Values are as described in the GICD_NSACR register. This is the value that is used if a SPI has an error.<br>Secure access only.<br>Resets to `0b00`. |
| [15:14] | - | Reserved, returns `0b00`. |
| [13:4] | CGO | One bit per clock gate:<br>1 = Leave clock running.<br>0 = Use full clock gating.<br>Clock gate bit assignments are specified in *Table 4-7 CGO field bit assignments* on page 4-114.<br>**Note**<br>CGO must be set if clock gates are not implemented. |
| [3:1] | - | Reserved, returns `0b000`. |
| [0] | SIP | Scrub in progress:<br>1 = Scrub in progress.<br>0 = No scrub in progress.<br>This bit is read and written by software. When a scrub is complete, the GIC clears the bit to 0. |

**Table 4-7 CGO field bit assignments**

| Value | Clock gate |
|---|---|
| `0x0` | CPU communications block |
| `0x1` | SPI registers and search |
| `0x2` | ACE-Lite slave interface |
| `0x3` | ACE-Lite master interface |
| `0x4` | LPI cache and search |
| `0x5` | SGI and GICR registers |
| `0x6` | Trace & debug |
| `0x7` | Pending table search and control |
| `0x8` | ITS communications block |

### 4.2.5 Secure Access Control register, GICD_SAC

This register allows Secure software to control Non-secure access to GIC-600 Secure features by other software.

The GICD_SAC characteristics are:

**Usage constraints** Only accessible by Secure accesses.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

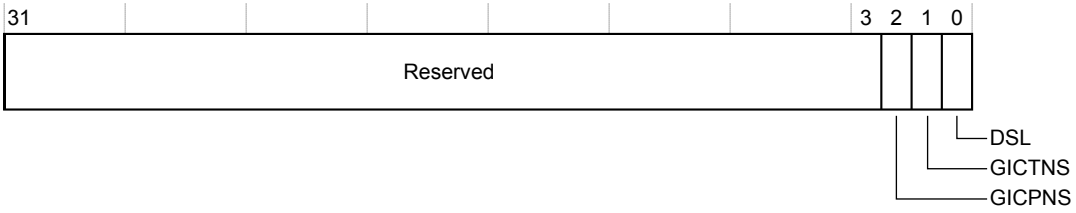


**Figure 4-5 GICD_SAC bit assignments**

The following table shows the bit assignments.

**Table 4-8 GICD_SAC bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:3] | - | Reserved, returns zero. |
| [2] | GICPNS | 1 = Allow Non-secure access to the GICP registers. This enables Non-secure access to Secure PMU data.<br>0 = Secure access only.<br>The **gicp_allow_ns** tie-off signal controls the reset value on a per-chip basis. |
| [1] | GICTNS | 1 = Allow Non-secure access to the GICT registers. This enables Non-secure access to Secure trace data.<br>0 = Secure access only.<br>The **gict_allow_ns** tie-off signal controls the reset value on a per-chip basis. |
| [0] | DSL | Disable Security Lock. *WriteOnce* (WO):<br>1 = WO bit to lock GICD_CTLR.DS to be WO at its current value.<br>0 = No effect.<br>When set to 1, this value is only returned by reset. |

### 4.2.6 Chip Status Register, GICD_CHIPSR

This register allows Secure software to access the status of the chip in a multichip configuration. A single copy of this register exists on each chip in a multichip configuration.

The GICD_CHIPSR characteristics are:

**Usage constraints** Only accessible by Secure accesses.

**Configurations** Available in all multichip configurations.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

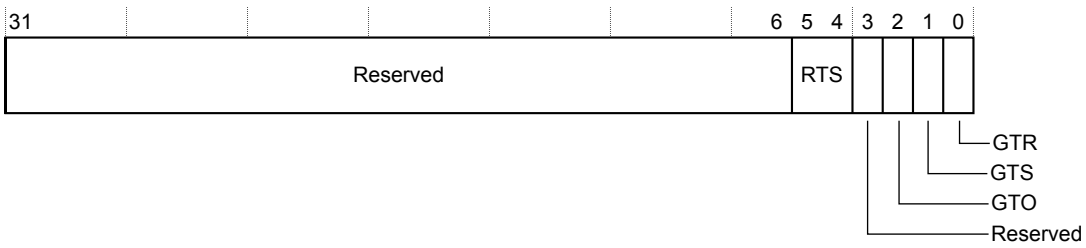


**Figure 4-6 GICD_CHIPSR bit assignments**

The following table shows the bit assignments.

**Table 4-9 GICD_CHIPSR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:6] | - | Reserved. |
| [5:4] | RTS | Route Table Status:<br>`0b00` = Disconnected.<br>`0b01` = Updating.<br>`0b10` = Consistent.<br>`0b11` = Reserved.<br>RAZ/WI = single chip configuration. |
| [3] | - | Reserved, SBZ. |
| [2] | GTO | Gating Transaction Ongoing:<br>0 = No accesses.<br>1 = Accesses ongoing.<br>This bit is RO. |
| [1] | GTS | Gating Status:<br>0 = Not gated.<br>1 = Gated.<br>This bit is RO. |
| [0] | GTR | Gating Request:<br>0 = Do not gate.<br>1 = Request to gate.<br>This bit is RO. |

### 4.2.7 Default Chip Register, GICD_DCHIPR

This register allows Secure software to access the status of a chip in a multichip system. A single copy of this register exists on each chip in a multichip configuration.

The GICD_DCHIPR characteristics are:

**Usage constraints** Only accessible by Secure accesses.

**Configurations** This register is available in all GIC-600 configurations except single chip systems.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

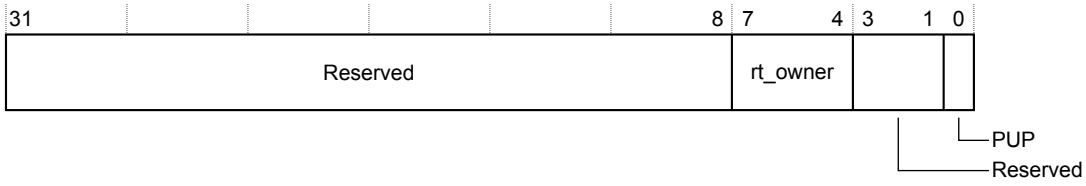


**Figure 4-7 GICD_DCHIPR bit assignments**

The following table shows the bit assignments.

**Table 4-10 GICD_DCHIPR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:8] | - | Reserved. |
| [7:4] | rt_owner | Routing table owner:<br>Value = 0-15. |
| [3:1] | - | Reserved. |
| [0] | PUP | Power Update in Progress:<br>This bit is RO.<br>0 = PUP not in progress.<br>1 = PUP in progress. |

### 4.2.8 Chip Registers, GICD_CHIPR<n>

Each register controls the configuration of the chip in a multichip system. This register exists on each chip in a multichip configuration and is identified by the chip number.

The GICD_CHIPR<n> characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations except in a single chip system.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

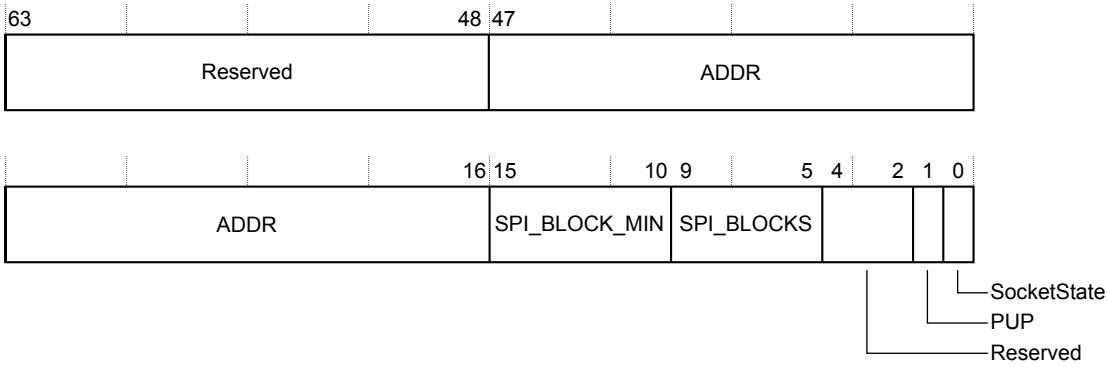


**Figure 4-8 GICD_CHIPR<n> bit assignments**

The following table shows the bit assignments.

**Table 4-11 GICD_CHIPR<n> bit assignments**

| Bits | Name | Function |
|---|---|---|
| [63:48] | - | Reserved. |
| [47:16] | ADDR | Controls the address bits[47:16]. Bits[15:0] of the address are zero. The value is driven by **icdrtdest** to route messages to the remote chip.<br>This bitfield is software RW. |

**Table 4-11 GICD_CHIPR<n> bit assignments (continued)**

| Bits | Name | Function |
|---|---|---|
| [15:10] | SPI_BLOCK_MIN | Controls the minimum number of SPIs in a group (block). The permitted values are 0-31.<br>This bitfield is software RW. |
| [9:5] | SPI_BLOCKS | Controls the number of SPI blocks. The permitted values are 0-31.<br>This bitfield is software RW. |
| [4:2] | - | Reserved. |
| [1] | PUP | This bit is RO and returns the power update status:<br>0 = Power update complete.<br>1 = Power update in progress. |
| [0] | SocketState | This bit is RW and controls the state of the chip:<br>0 = Chip is offline.<br>1 = Chip is online. |

### 4.2.9 Interrupt Class Registers, GICD_ICLARn

These registers control whether a 1 of N SPI can target a core that is assigned to class 0 or class 1 group. Each register controls 16 SPIs and the GIC-600 has 60 registers, GICD_ICLAR2-GICD_ICLAR61.

The GICD_ICLARn characteristics are:

**Usage constraints** The Distributor provides up to 60 registers to support 960 SPIs. If you configure the GIC-600 to use fewer than 960 SPIs, then it reduces the number of registers accordingly. For locations where interrupts are not implemented, the register is RAZ/WI.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

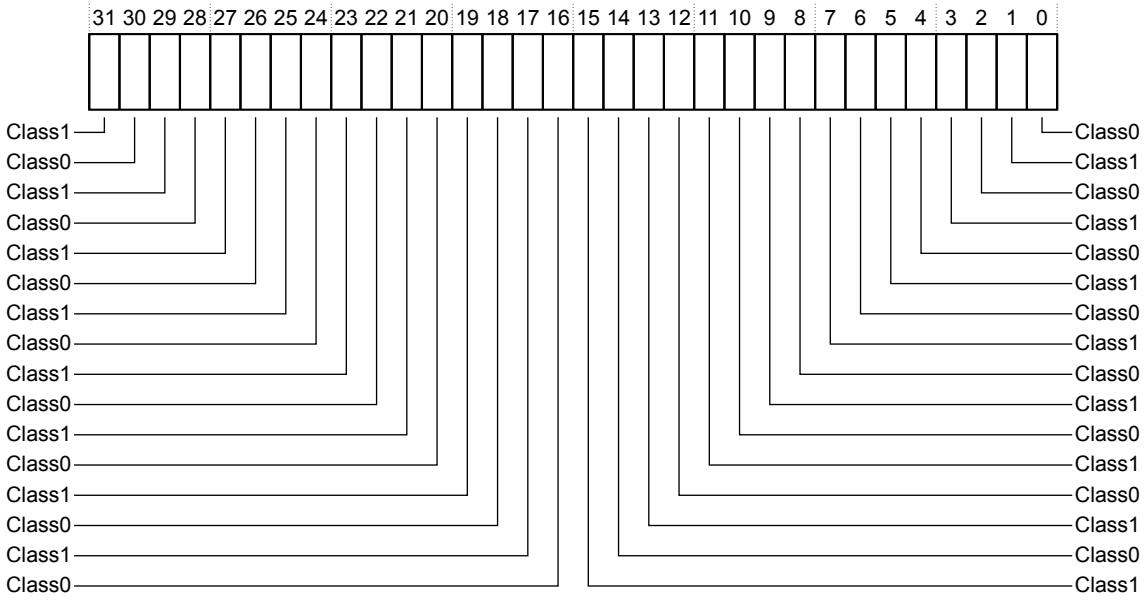


**Figure 4-9 GICD_ICLARn bit assignments**

The following table shows the bit assignments.

**Table 4-12 GICD_ICLARn bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31,29,27,25,23,21,19,17,15,13,11,9,7,5,3,1] | Class1 | Controls whether the 1 of N SPI can target a core that is assigned to class 1:<br>0 = The SPI can target a core that is assigned to class 1 group.<br>1 = The SPI cannot target a core that is assigned to class 1 group.<br>**Note**<br>The SPI that a bit refers to depends on its bit position and the base address offset of the GICD_ICLARn. |
| [30,28,26,24,22,20,18,16,14,12,10,8,6,4,2,0] | Class0 | Controls whether the 1 of N SPI can target a core that is assigned to class 0:<br>0 = The SPI can target a core that is assigned to class 0 group.<br>1 = The SPI cannot target a core that is assigned to class 0 group.<br>**Note**<br>The SPI that a bit refers to depends on its bit position and the base address offset of the GICD_ICLARn. |

### 4.2.10 Interrupt Error Registers, GICD_IERRRn

These registers indicate the error status of a SPI. Each register monitors 32 SPIs and the GIC-600 has 30 registers, GICD_IERRR1-GICD_IERRR30.

The GICD_IERRRn characteristics are:

**Usage constraints** The Distributor provides up to 30 registers to support 960 SPIs. If you configure the GIC-600 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** Available in all GIC-600 configurations.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

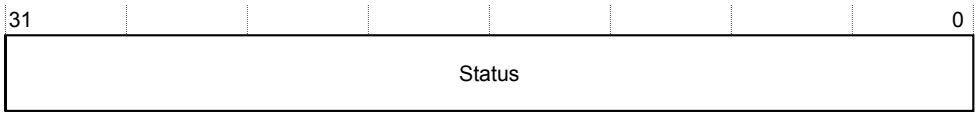


**Figure 4-10 GICD_IERRRn bit assignments**

The following table shows the bit assignments.

**Table 4-13 GICD_IERRRn bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:0] | Status | Indicates whether a SPI is in an error state:<br>0 = If read, the SPI is not in an error state and programming is valid. Write has no effect.<br>1 = If read, the SPI is in an error state and programming is not valid. Write clears the error.<br>**Note**<br>The SPI that a bit refers to depends on its bit position and the base address offset of the GICD_IERRRn. |

## 4.2.11 Configuration ID Register, GICD_CFGID

This register contains information that enables test software to determine if the GIC-600 system is compatible. There is one register per configured chip. The offset determines the chip number.

The GICD_CFGID characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

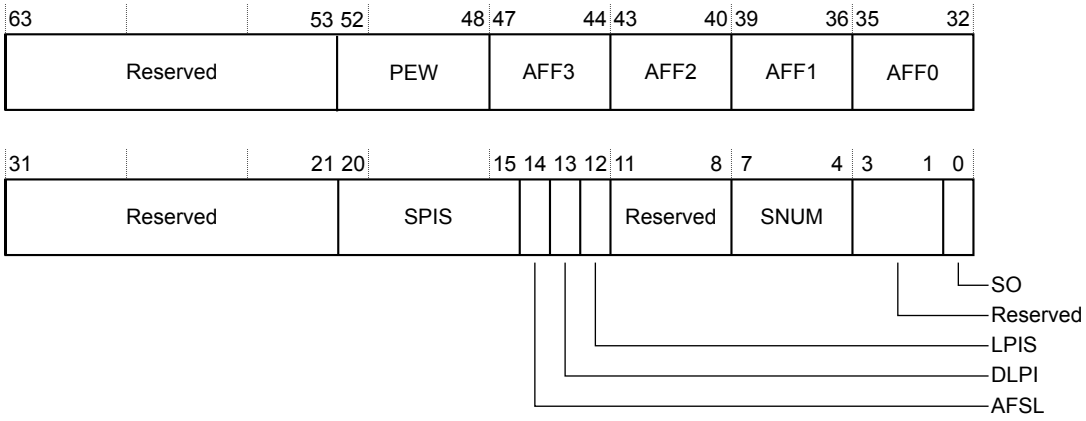


**Figure 4-11 GICD_CFGID bit assignments**

The following table shows the bit assignments.

**Table 4-14 GICD_CFGID bit assignments**

| Bits | Name | Function |
|---|---|---|
| [63:53] | - | Reserved, returns zero. |
| [52:48] | PEW | Width of lower part of on-chip core number field, ceiling[$\log_2$(max_pe_on_chip)], where `max_pe_on_chip` is the configuration option that you set, which defines the maximum number of cores on a single chip in the system.<br>This field is RES0 in versions before r1p2. |
| [47:44] | AFF3 | Returns the Affinity3 bits. |
| [43:40] | AFF2 | Returns the Affinity2 bits. |
| [39:36] | AFF1 | Returns the Affinity1 bits. |
| [35:32] | AFF0 | Returns the Affinity0 bits. |
| [31:21] | - | Reserved, returns zero. |
| [20:15] | SPIS | Number of SPI blocks supported. |
| [14] | AFSL | Chip affinity selection level. |
| [13] | DLPI | Direct LPI registers supported. |
| [12] | LPIS | LPI supported. |
| [11:8] | - | Reserved, returns zero. |
| [7:4] | SNUM | Chip number. |
| [3:1] | - | Reserved, returns zero. |
| [0] | SO | Chip offline. |

### 4.2.12 Peripheral ID4 register, GICD_PIDR4

This register returns byte[4] of the peripheral ID. The GICD_PIDR4 register is part of the set of Distributor peripheral identification registers.

The GICD_PIDR4 characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

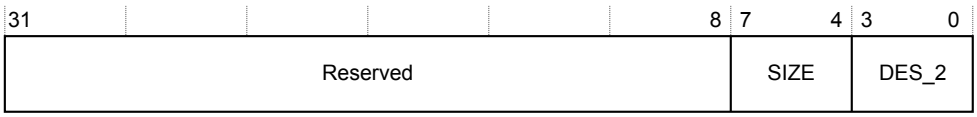


**Figure 4-12 GICD_PIDR4 bit assignments**

The following table shows the bit assignments.

**Table 4-15 GICD_PIDR4 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:8] | - | Reserved, RAZ. |
| [7:4] | SIZE | Indicates the amount of memory that the GIC-600 occupies. $Log_2$ of the number of 4KB pages from the start to the end of the GIC-600 component ID registers.<br>`0x4`. |
| [3:0] | DES_2 | Bits[10:7] of the JEDEC JEP106 identification code. Together, PIDR1.DES_0, PIDR2.DES_1, and PIDR4.DES_2 identify the component designer.<br>`0x4`. |

### 4.2.13 Peripheral ID3 register, GICD_PIDR3

This register returns byte[3] of the peripheral ID. The GICD_PIDR3 register is part of the set of Distributor peripheral identification registers.

The GICD_PIDR3 characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

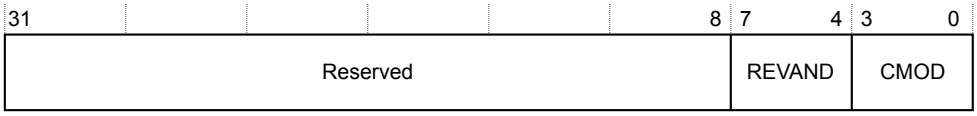


**Figure 4-13 GICD_PIDR3 bit assignments**

The following table shows the bit assignments.

**Table 4-16 GICD_PIDR3 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:8] | - | Reserved, RAZ. |
| [7:4] | REVAND | Indicates minor errata fixes specific to the revision of the component being used, for example metal fixes after implementation. `0x0` indicates that there are no errata fixes to this component.<br>`0x0`. |
| [3:0] | CMOD | Customer modified. Indicates whether the customer has modified the behavior of the component. Usually, this field is `0x0`. Customers change this value when they make authorized modifications to this component.<br>`0x0`. |

### 4.2.14 Peripheral ID2 register, GICD_PIDR2

This register returns byte[2] of the peripheral ID. The GICD_PIDR2 register is part of the set of Distributor peripheral identification registers.

The GICD_PIDR2 characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

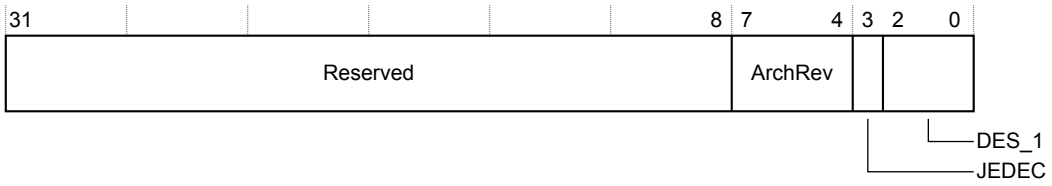


**Figure 4-14 GICD_PIDR2 bit assignments**

The following table shows the bit assignments.

**Table 4-17 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-600 complies:<br>• `0x3` = GICv3. |
| [3] | JEDEC | Indicates that a JEDEC-assigned JEP106 identity code is used. |
| [2:0] | DES_1 | Bits[6:4] of the JEP106 identity code. Bits[3:0] of the JEP106 identity code are assigned to GICD_PIDR1. |

### 4.2.15 Peripheral ID1 register, GICD_PIDR1

This register returns byte[1] of the peripheral ID. The GICD_PIDR1 register is part of the set of Distributor peripheral identification registers.

The GICD_PIDR1 characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

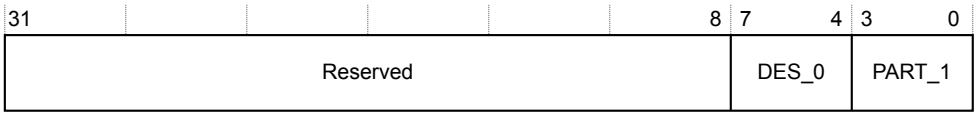


**Figure 4-15 GICD_PIDR1 bit assignments**

The following table shows the bit assignments.

**Table 4-18 GICD_PIDR1 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:8] | - | Reserved, RAZ. |
| [7:4] | DES_0 | Bits[3:0] of JEDEC JEP106 identification code. Together, PIDR1.DES_0, PIDR2.DES_1, and PIDR4.DES_2 identify the component designer.<br>`0xB`. |
| [3:0] | PART_1 | Bits[11:8] of the 12-bit part number of the component. Together, PART_0 and PART_1 field values indicate the part number of the component.<br>`0x4`. |

### 4.2.16 Peripheral ID0 register, GICD_PIDR0

This register returns byte[0] of the peripheral ID. The GICD_PIDR0 register is part of the set of Distributor peripheral identification registers.

The GICD_PIDR0 characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106.

The following figure shows the bit assignments.

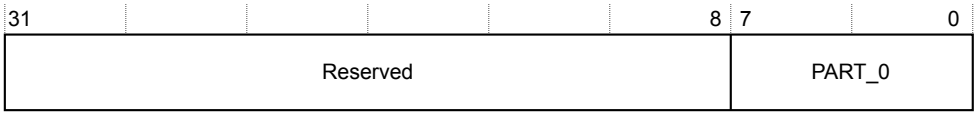


**Figure 4-16 GICD_PIDR0 bit assignments**

The following table shows the bit assignments.

**Table 4-19 GICD_PIDR0 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:8] | - | Reserved, RAZ. |
| [7:0] | PART_0 | Bits[7:0] of the 12-bit part number of the component. Together, PART_0 and PART_1 field values indicate the part number of the component.<br>`0x92`. |

## 4.3 Distributor registers (GICA) for message-based SPIs summary

The functions for the GIC-600 message-based SPIs are controlled through the Distributor registers identified with the prefix GICA.

The following table lists the message-based SPI registers. All registers are 32 bits wide.

**Table 4-20 Distributor registers (GICA) for message-based SPIs summary**

| Offset | Name | Type | Width | Reset | Description[t] | Architecture defined? |
|---|---|---|---|---|---|---|
| 0x0000-0x003C | - | - | - | - | Reserved | - |
| 0x0040 | GICA_SETSPI_NSR | WO | 32 | - | Aliased Non-secure SPI Set Register | Yes |
| 0x0044 | - | - | - | - | Reserved | - |
| 0x0048 | GICA_CLRSPI_NSR | WO | 32 | - | Aliased Non-secure SPI Clear Register | Yes |
| 0x004C | - | - | - | - | Reserved | - |
| 0x0050 | GICA_SETSPI_SR[u] | WO | 32 | - | Aliased Secure SPI Set Register[v] | Yes |
| 0x0054 | - | - | - | - | Reserved | - |
| 0x0058 | GICA_CLRSPI_SR[u] | WO | 32 | - | Aliased Secure SPI Clear Register[v] | Yes |
| 0x005C-0xFFFC | - | - | - | - | Reserved | - |

[t] For the description of the registers that are not specific to the GIC-600, see the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.
[u] The existence of this register depends on the configuration of the GIC-600. If Security support is not included, this register does not exist.
[v] This register is only accessible from a Secure access.

## 4.4 Redistributor registers for control and physical LPIs summary

The functions for the GIC-600 physical LPIs are controlled through the Redistributor registers identified with the prefix GICR. In GICv3, these registers start from the base address.

For more information about LPIs, see the *Arm® GICv3 and GICv4 Software Overview*.

For descriptions of registers that are not specific to the GIC-600, see the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

**Table 4-21 Redistributor registers for control and physical LPIs summary**

| Offset | Name | Type | Width | Reset | Description | Architecture defined? |
|---|---|---|---|---|---|---|
| 0x0000 | GICR_CTLR | RW | 32 | 0x0 | Redistributor Control Register | Yes |
| 0x0004 | GICR_IIDR | RO | 32 | Configuration dependent | *4.4.1 Redistributor Implementation Identification Register, GICR_IIDR* on page 4-127 | Yes |
| 0x0008 | GICR_TYPER | RO | 64 | Configuration dependent | *4.4.2 Interrupt Controller Type Register, GICR_TYPER* on page 4-128 | Yes |
| 0x0010 | - | - | - | - | Reserved | - |
| 0x0014 | GICR_WAKER | RW | 32 | 0x3 | *4.4.3 Power Management Control Register, GICR_WAKER* on page 4-130 | [w] |
| 0x0018-0x001C | - | - | - | - | Reserved | - |
| 0x0020 | GICR_FCTLR | RW | 32 | 0x0 | *4.4.4 Function Control Register, GICR_FCTLR* on page 4-131 | [k] |
| 0x0024 | GICR_PWRR | RW | 32 | Configuration dependent | *4.4.5 Power Register, GICR_PWRR* on page 4-132 | [k] |
| 0x0028 | GICR_CLASS | RW | 32 | 0x0 | *4.4.6 Class Register, GICR_CLASS* on page 4-133 | [k] |
| 0x002C-0x003C | - | - | - | - | Reserved | - |
| 0x0040 | GICR_SETLPIR[x] | WO | 64 | - | - | Yes |
| 0x0048 | GICR_CLRLPIR[x] | WO | 64 | - | - | Yes |
| 0x0050-0x006C | - | - | - | - | Reserved | - |
| 0x0070 | GICR_PROPBASER[y] | RW | 64 | Configuration dependent | Redistributor Properties Base Address Register | Yes |
| 0x0078 | GICR_PENDBASER[yz] | RW | 64 | Configuration dependent | Redistributor LPI Pending Table Base Address Register[aa] | Yes |
| 0x0080-0x009C | - | - | - | - | Reserved | - |

[w] Parts of this register are architecture defined and the other parts are microarchitecture defined.
[x] This register is present only when Direct LPI registers are configured.
[y] The existence of this register depends on the configuration of the GIC-600. If ITS and LPI support is not included, this register does not exist.
[z] Arm recommends that if possible, you set the GICR_PENDBASER Pending Table Zero bit to one. This reduces the power and time that is taken during initialization.
[aa] This register is only accessible from a Secure access.

**Table 4-21 Redistributor registers for control and physical LPIs summary (continued)**

| Offset | Name | Type | Width | Reset | Description | Architecture defined? |
|---|---|---|---|---|---|---|
| 0x00A0 | GICR_INVLPIR[x] | WO | 64 | - | - | Yes |
| 0x00A8-0x00AC | - | - | - | - | Reserved | - |
| 0x00B0 | GICR_INVALLR[x] | WO | 64 | - | - | Yes |
| 0x00B8-0x00BC | - | - | - | - | Reserved | - |
| 0x00C0 | GICR_SYNCR[x] | RO | 32 | 0x0 | - | Yes |
| 0x00C4-0xFFCC | - | - | - | - | Reserved | - |
| 0xFFD0 | GICR_PIDR4 | RO | 32 | 0x44 | Peripheral ID 4 Register | No |
| 0xFFD4 | GICR_PIDR5 | RO | 32 | 0x00 | Peripheral ID 5 Register | No |
| 0xFFD8 | GICR_PIDR6 | RO | 32 | 0x00 | Peripheral ID 6 Register | No |
| 0xFFDC | GICR_PIDR7 | RO | 32 | 0x00 | Peripheral ID 7 Register | No |
| 0xFFE0 | GICR_PIDR0 | RO | 32 | 0x93 | Peripheral ID 0 Register | No |
| 0xFFE4 | GICR_PIDR1 | RO | 32 | 0xB4 | Peripheral ID 1 Register | No |
| 0xFFE8 | GICR_PIDR2 | RO | 32 | 0x3B | *4.4.7 Peripheral ID2 Register, GICR_PIDR2* on page 4-134 | No |
| 0xFFEC | GICR_PIDR3 | RO | 32 | 0x00 | Peripheral ID 3 Register | No |
| 0xFFF0 | GICR_CIDR0 | RO | 32 | 0x0D | Peripheral ID 0 Register | No |
| 0xFFF4 | GICR_CIDR1 | RO | 32 | 0xF0 | Peripheral ID 1 Register | No |
| 0xFFF8 | GICR_CIDR2 | RO | 32 | 0x05 | Peripheral ID 2 Register | No |
| 0xFFFC | GICR_CIDR3 | RO | 32 | 0xB1 | Peripheral ID 3 Register | No |

This section contains the following subsections:

- *4.4.1 Redistributor Implementation Identification Register, GICR_IIDR* on page 4-127.
- *4.4.2 Interrupt Controller Type Register, GICR_TYPER* on page 4-128.
- *4.4.3 Power Management Control Register, GICR_WAKER* on page 4-130.
- *4.4.4 Function Control Register, GICR_FCTLR* on page 4-131.
- *4.4.5 Power Register, GICR_PWRR* on page 4-132.
- *4.4.6 Class Register, GICR_CLASS* on page 4-133.
- *4.4.7 Peripheral ID2 Register, GICR_PIDR2* on page 4-134.

### 4.4.1 Redistributor Implementation Identification Register, GICR_IIDR

This register provides information about the implementer and revision of the Redistributor.

The GICR_IIDR characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.4 Redistributor registers for control and physical LPIs summary* on page 4-126.

The following figure shows the bit assignments.

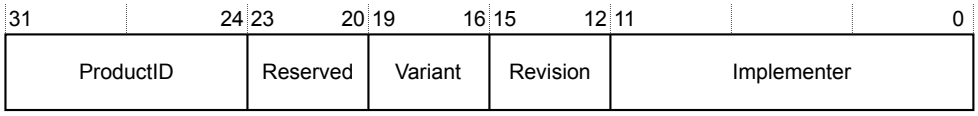


**Figure 4-17 GICR_IIDR bit assignments**

The following table shows the bit assignments.

**Table 4-22 GICR_IIDR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:24] | ProductID | Indicates the product ID:<br>`0x2` = GIC-600. |
| [23:20] | - | Reserved, RAZ |
| [19:16] | Variant | Indicates the major revision or variant of the product for the `rmpn` identifier:<br>`0x0` = r0.<br>`0x1` = r1. |
| [15:12] | Revision | Indicates the minor revision of the product for the `rmpn` identifier:<br>`0x1` = p0.<br>`0x3` = p1.<br>`0x4` = p2.<br>`0x5` = p3. |
| [11:0] | Implementer | Identifies the implementer:<br>`0x43B` = Arm. |

### 4.4.2 Interrupt Controller Type Register, GICR_TYPER

This register returns information about the configuration of the GIC-600.

The GICR_TYPER characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.4 Redistributor registers for control and physical LPIs summary* on page 4-126.

The following figure shows the bit assignments.

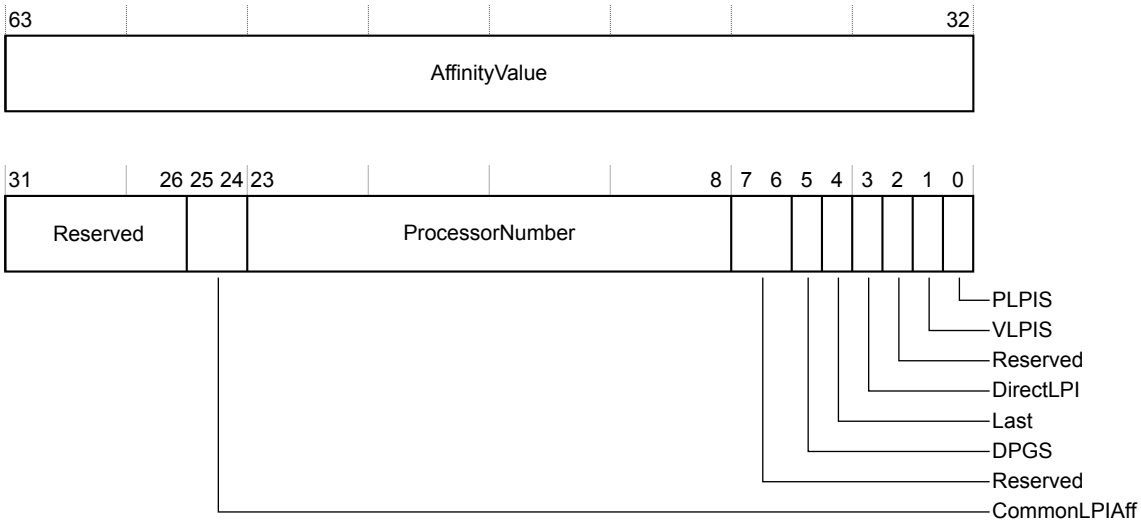


**Figure 4-18 GICR_TYPER bit assignments**

The following table shows the bit assignments.

**Table 4-23 GICR_TYPER bit assignments**

| Bits | Name | Function |
|---|---|---|
| [63:32] | AffinityValue | Affinity level value for this Redistributor:<br>AF3, bits[63:56], Affinity level 3 value.<br>AF2, bits[55:48], the Affinity level 2 value.<br>AF1, bits[47:40], the Affinity level 1 value.<br>AF0, bits[39:32], the Affinity level 0 value. |
| [31:26] | - | Reserved, returns `0b000000`. |
| [25:24] | CommonLPIAff | Returns:<br>`0b00` = Single core configuration.<br>`0b01` = If chip set by AF3.<br>`0b10` = If chip set by AF2.<br>`0b11` = Reserved. |
| [23:8] | ProcessorNumber | Returns the core number and chip number that uniquely identifies this core in the system. |
| [7:6] | - | Reserved, returns `0b00`. |
| [5] | DPGS | Disable Processor Group Selections:<br>1 = The GIC-600 supports DPG. See the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*. |
| [4] | Last | Last Redistributor:<br>0 = This Redistributor is not the last Redistributor on the chip.<br>1 = This Redistributor is the last Redistributor on the chip. |

**Table 4-23 GICR_TYPER bit assignments (continued)**

| Bits | Name | Function |
|---|---|---|
| [3] | DirectLPI | Direct injection of LPIs:<br>0 = The GIC-600 does not support LPIs.<br>1 = The GIC-600 supports LPIs and there is no ITS in the system. |
| [2] | - | Reserved, returns 0. |
| [1] | VLPIS | Virtual LPI support:<br>0 = The GIC-600 does not support Virtual LPIs.<br>See the *Arm® GICv3 and GICv4 Software Overview*. |
| [0] | PLPIS | Physical LPI support:<br>0 = The GIC-600 does not support LPIs.<br>1 = The GIC-600 supports LPIs. |

### 4.4.3 Power Management Control Register, GICR_WAKER

This register controls whether the GIC-600 can be powered down.

The GICR_WAKER characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.4 Redistributor registers for control and physical LPIs summary* on page 4-126.

The following figure shows the bit assignments.

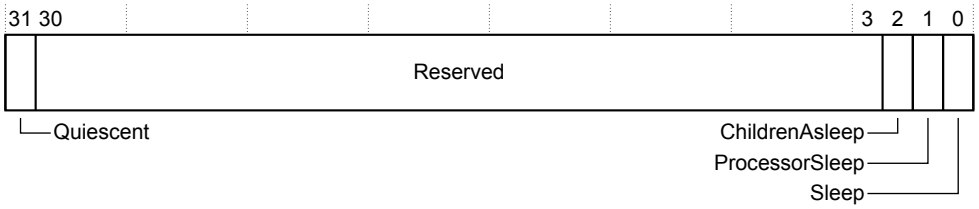


**Figure 4-19 GICR_WAKER bit assignments**

The following table shows the bit assignments.

**Table 4-24 GICR_WAKER bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31] | Quiescent | Indicates that the GIC-600 is idle and can be powered down if necessary. |
| [30:3] | - | Reserved, RAZ. |
| [2] | ChildrenAsleep | Indicates that the bus between the CPU interface and this Redistributor is quiescent. |

**Table 4-24 GICR_WAKER bit assignments (continued)**

| Bits | Name | Function |
|---|---|---|
| [1] | ProcessorSleep | Indicates:<br>0 = This Redistributor never asserts **wake_request**.<br>1 = This Redistributor must assert a **wake_request** if there is a pending interrupt targeted at the connected core. See *3.6.2 Processor core power management* on page 3-58. |
| [0] | Sleep | Indicates the sleep state:<br>0 = Normal operation.<br>1 = The GIC-600 ensures that all the caches are consistent with external memory and that it is safe to power down. See *3.6.3 Other power management* on page 3-59. |

***Related reference***
*3.6.3 Other power management* on page 3-59

### 4.4.4 Function Control Register, GICR_FCTLR

This register controls the scrubbing of all RAMs in the associated Redistributor.

The GICR_FCTLR characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.4 Redistributor registers for control and physical LPIs summary* on page 4-126.

The following figure shows the bit assignments.

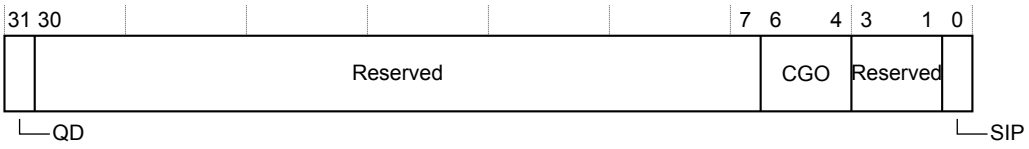


**Figure 4-20 GICR_FCTLR bit assignments**

The following table shows the bit assignments.

**Table 4-25 GICR_FCTLR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31] | QD | Q-Channel deny:<br>0 = Allow Q-Channel accesses.<br>1 = Deny Q-Channel accesses. |
| [30:7] | - | Reserved, RAZ/WI. |

**Table 4-25 GICR_FCTLR bit assignments (continued)**

| Bits | Name | Function |
|---|---|---|
| [6:4] | CGO | One bit per clock gate:<br>0 = Use full clock gating.<br>1 = Leave clock running.<br>**Note**<br>CGO must be set if clock gates are not implemented. |
| [3:1] | - | Reserved, RAZ/WI. |
| [0] | SIP | Scrub in progress:<br>0 = No scrub in progress.<br>1 = Scrub in progress.<br>This bit is read and written by software. When a scrub is complete, the GIC clears the bit to 0. |

### 4.4.5 Power Register, GICR_PWRR

This register controls the power up sequence of the Redistributors. Software must write to this register during the power up sequence.

The GICR_PWRR characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.4 Redistributor registers for control and physical LPIs summary* on page 4-126.

The following figure shows the bit assignments.

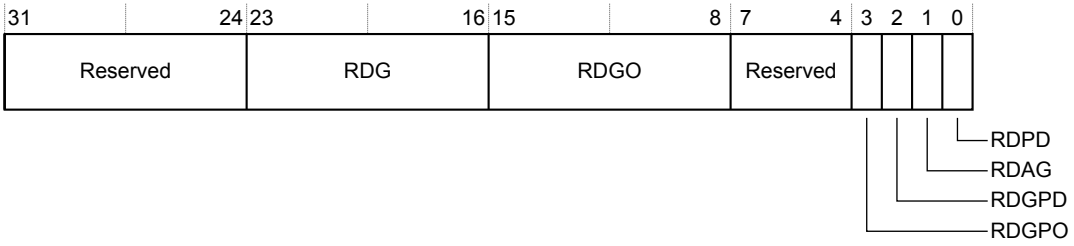


**Figure 4-21 GICR_PWRR bit assignments**

The following table shows the bit assignments.

**Table 4-26 GICR_PWRR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:24] | - | Reserved, RAZ. |
| [23:16] | RDG | RDGroup. This read-only field indicates the number of the current Redistributor. Must be packed from 0. |
| [15:8] | RDGO | RDGroupOffset. This read-only field indicates the offset of the current core that is connected to the current Redistributor. Must be packed from 0 but does not necessarily map to a single cluster because the AXI4-Stream bus can be subdivided. |

**Table 4-26 GICR_PWRR bit assignments (continued)**

| Bits | Name | Function |
|---|---|---|
| [7:4] | - | Reserved, RAZ. |
| [3] | RDGPO | RDGroupPoweredOff. This read-only bit indicates:<br>0 = Redistributor is powered up and can be accessed.<br>1 = It is safe to power down the Redistributor. |
| [2] | RDGPD | RDGroupPowerDown. This read-only bit indicates the intentional power state of the Redistributor:<br>0 = Intend to power up.<br>1 = Intend to power down.<br>The Redistributor has reached its intentional power state when RDGPD = RDGPO. |
| [1] | RDAG | RDApplyGroup. This write-only bit applies the RDPD value to all Redistributors in the group.<br>If the RDPD value cannot be applied to all cores in the group, then the GIC ignores this request. |
| [0] | RDPD | RDPowerDown:<br>0 = Redistributor is powered up and can be accessed.<br>1 = The core permits the Redistributor to be powered down.<br>Writes to 1 ignored if GICR_WAKER.ProcessorSleep ! = 1.<br>Writes ignored if RDGPD ! = RDGPO and changing to not match RDGPD.<br>If all other cores in the Redistributor group have RDPD == 1, then setting this bit to 1 also sets RDGPD = 1. |

***Related reference***
*3.6.1 Redistributor power management* on page 3-58

### 4.4.6 Class Register, GICR_CLASS

This register controls the assignment of interrupts to either class 0 or class 1.

The GICR_CLASS characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.4 Redistributor registers for control and physical LPIs summary* on page 4-126.

The following figure shows the bit assignments.

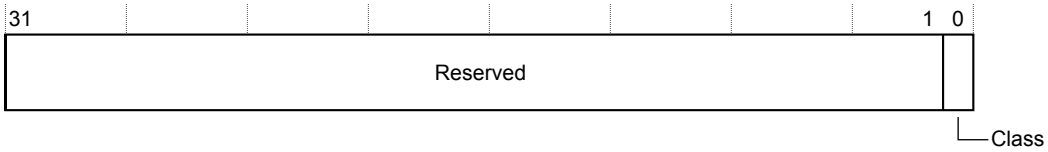


**Figure 4-22 GICR_CLASS bit assignments**

The following table shows the bit assignments.

**Table 4-27 GICR_CLASS bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:1] | - | Reserved, RAZ/WI. |
| [0] | Class | Interrupt class:<br>0 = Class 0.<br>1 = Class 1. |

### 4.4.7 Peripheral ID2 Register, GICR_PIDR2

This register returns byte[2] of the peripheral ID. The GICR_PIDR2 register is part of the set of Redistributor peripheral identification registers.

The GICR_PIDR2 characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.4 Redistributor registers for control and physical LPIs summary on page 4-126*.

The following figure shows the bit assignments.

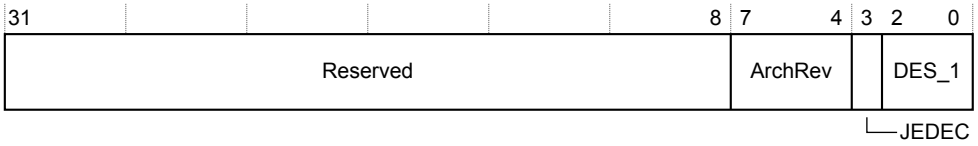


**Figure 4-23 GICR_PIDR2 bit assignments**

The following table shows the bit assignments.

**Table 4-28 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-600 complies:<br>• `0x3` = GICv3. |
| [3] | JEDEC | Indicates that a JEDEC-assigned JEP106 identity code is used. |
| [2:0] | DES_1 | Bits[6:4] of the JEP106 identity code. Bits[3:0] of the JEP106 identity code are assigned to GICR_PIDR1. |

***Related information***

*Arm® GICv3 and GICv4 Software Overview*

## 4.5 Redistributor registers for SGIs and PPIs summary

The functions for the GIC-600 SGIs and PPIs are controlled through the Redistributor registers identified with the prefix GICR.

For descriptions of registers that are not specific to the GIC-600, see the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

**Table 4-29 Redistributor registers for SGIs and PPIs summary**

| Offset | Name | Type | Width | Reset | Description | Architecture defined? |
|---|---|---|---|---|---|---|
| 0x0000-0x007C | - | - | - | - | Reserved | - |
| 0x0080 | GICR_IGROUPR0 | RW | 32 | 0x0 | Interrupt Group Register | Yes |
| 0x0084-0x0FFC | - | - | - | - | Reserved | - |
| 0x0100 | GICR_ISENABLER0 | RW | 32 | 0x0 | Interrupt Set-Enable Register | Yes |
| 0x0104-0x017C | - | - | - | - | Reserved | - |
| 0x0180 | GICR_ICENABLER0 | RW | 32 | 0x0 | Interrupt Clear-Enable Register | Yes |
| 0x0184-0x01FC | - | - | - | - | Reserved | - |
| 0x0200 | GICR_ISPENDR0 | RW | 32 | PPI wire dependent | Interrupt Set-Pending Register | Yes |
| 0x0204-0x027C | - | - | - | - | Reserved | - |
| 0x0280 | GICR_ICPENDR0 | RW | 32 | PPI wire dependent | Peripheral Clear Pending Register | Yes |
| 0x0284-0x02FC | - | - | - | - | Reserved | - |
| 0x0300 | GICR_ISACTIVER0 | RW | 32 | 0x0 | Interrupt Set-Active Register | Yes |
| 0x0304-0x037C | - | - | - | - | Reserved | - |
| 0x0380 | GICR_ICACTIVER0 | RW | 32 | 0x0 | Interrupt Clear-Active Register | Yes |
| 0x0384-0x03FC | - | - | - | - | Reserved | - |
| 0x0400-0x041C | GICR_IPRIORITYRn | RW | 32 | 0x0 | Interrupt Priority Registers | Yes |
| 0x0420-0x0BFC | - | - | - | - | Reserved | - |
| 0x0C00-0x0C04 | GICR_ICFGRn | RW | 32 | 0xAAAAAAAA | Interrupt Configuration Registers | Yes |
| 0x0C08-0x0CFC | - | - | - | - | Reserved | - |
| 0x0D00 | GICR_IGRPMODR0 | RW | 32 | 0x0 | Interrupt Group Modifier Register | Yes |
| 0x0D04-0x0DFC | - | - | - | - | Reserved | - |
| 0x0E00 | GICR_NSACR | RW | 32 | 0x0 | Non-secure Access Control Register | Yes |
| 0x0E04-0xBFFC | - | - | - | - | Reserved | - |
| 0xC000 | GICR_MISCSTATUSR | RO | 32 | 0x0 | *4.5.1 Miscellaneous Status Register, GICR_MISCSTATUSR* on page 4-136 | k |

**Table 4-29 Redistributor registers for SGIs and PPIs summary (continued)**

| Offset | Name | Type | Width | Reset | Description | Architecture defined? |
|---|---|---|---|---|---|---|
| 0xC004 | - | - | - | - | Reserved | - |
| 0xC008 | GICR_IERRVR | RO | 32 | 0x0 | *4.5.2 Interrupt Error Valid Register, GICR_IERRVR* on page 4-137 | k |
| 0xC00C | - | - | - | - | Reserved | - |
| 0xC010 | GICR_SGIDR | RW | 64 | - | *4.5.3 SGI Default Register, GICR_SGIDR* on page 4-138 | k |
| 0xC018-0xEFFC | - | - | - | - | Reserved | - |
| 0xF000 | GICR_CFGID0 | RO | 32 | Configuration dependent | *4.5.4 Configuration ID0 Register, GICR_CFGID0* on page 4-138 | k |
| 0xF004 | GICR_CFGID1 | RO | 32 | Configuration dependent | *4.5.5 Configuration ID1 Register, GICR_CFGID1* on page 4-139 | k |

This section contains the following subsections:

- *4.5.1 Miscellaneous Status Register, GICR_MISCSTATUSR* on page 4-136.
- *4.5.2 Interrupt Error Valid Register, GICR_IERRVR* on page 4-137.
- *4.5.3 SGI Default Register, GICR_SGIDR* on page 4-138.
- *4.5.4 Configuration ID0 Register, GICR_CFGID0* on page 4-138.
- *4.5.5 Configuration ID1 Register, GICR_CFGID1* on page 4-139.

### 4.5.1 Miscellaneous Status Register, GICR_MISCSTATUSR

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-600.

The GICR_MISCSTATUSR characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.5 Redistributor registers for SGIs and PPIs summary* on page 4-135.

The following figure shows the bit assignments.

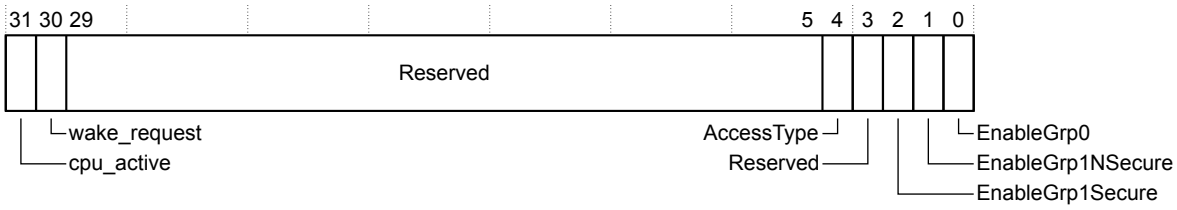


**Figure 4-24 GICR_MISCSTATUSR bit assignments**

The following table shows the bit assignments.

**Table 4-30 GICR_MISCSTATUSR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31] | cpu_active | Returns the status of the **cpu_active** signal for the core corresponding to the Redistributor whose register is being read:<br>0 = **cpu_active** input signal not active.<br>1 = **cpu_active** input signal active.<br>This bit is UNDEFINED when ProcessorSleep or ChildrenAsleep is set for a core, because the core is presumed to be powered down. |
| [30] | wake_request | 0 = **wake_request** not active.<br>1 = **wake_request** asserted. |
| [29:5] | - | Reserved. |
| [4] | AccessType | 0 = Secure access.<br>1 = Non-secure access. |
| [3] | - | Reserved. |
| [2][ab] | EnableGrp1Secure | In systems with two Security states enabled, when GICD_CTLR.DS == 0, then:<br>• For Secure reads, returns the Group 1 Secure CPU interface enable.<br>• For Non-secure reads, returns zero.<br>In systems with only a single Security state enabled, when GICD_CTLR.DS == 1, then this bit returns zero. |
| [1][ab] | EnableGrp1NSecure | In systems with two Security states enabled, when GICD_CTLR.DS == 0, then:<br>• For Secure reads, this bit returns the Group 1 Non-secure CPU interface enable.<br>• For Non-secure reads, when GICD_CTLR.ARE_NS == 1, this bit returns the Group 1 Non-secure CPU interface enable.<br>• For Non-secure reads when GICD_CTLR.ARE_NS == 0, this bit returns zero.<br>In systems with only a single Security state enabled, when GICD_CTLR.DS == 1, this bit returns the Group 1 CPU interface enable. |
| [0][ab] | EnableGrp0 | In systems with two Security states enabled, when GICD_CTLR.DS == 0, then:<br>• For Secure reads, this bit returns the Group 0 CPU interface enable.<br>• For Non-secure reads when GICD_CTLR.ARE_NS == 0, this bit returns the Group 1 Non-secure CPU interface enable.<br>• For Non-secure reads when GICD_CTLR.ARE_NS == 1, this bit returns zero.<br>In systems with only a single Security state enabled, when GICD_CTLR.DS == 1, this bit returns the Group 0 CPU interface enable. |

### 4.5.2 Interrupt Error Valid Register, GICR_IERRVR

This register indicates if the SGI or PPI data has been corrupted in SRAM.

The GICR_IERRVR characteristics are:

**Usage constraints** There are no usage constraints.

[ab] 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. Upstream Write packets maintain these copies that can de-synchronize after an incorrect powerdown sequence. This register enables you to debug this scenario. For more information, see the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.5 Redistributor registers for SGIs and PPIs summary* on page 4-135.

The following table shows the bit assignments.

**Table 4-31 GICR_IERRVR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [0] + n | valid<n><br>n = 0-31 | 0 = No error on interrupt n.<br>1 = There is an error on interrupt n so the interrupt is not delivered. |

### 4.5.3 SGI Default Register, GICR_SGIDR

This register controls the default value of SGI settings, for use in the case of a *Double-bit Error Detect Error* (DEDERR).

The GICR_SGIDR characteristics are:

**Usage constraints** Only accessible by Secure accesses.

**Configurations** Available in all GIC-600 configurations. If SGI ECC is not enabled, then this register is RES0.

**Attributes** See *4.5 Redistributor registers for SGIs and PPIs summary* on page 4-135.

The following table shows the bit assignments.

**Table 4-32 GICR_SGIDR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [3] + 4n:<br>[63, 59, 55, 51, 47, 43, 39, 35, 31, 27, 23, 19, 15, 11, 7, 3] | - | Reserved, RES0. |
| [2] + 4n:<br>[62, 58, 54, 50, 46, 42, 38, 34, 30, 26, 22, 18, 14, 10, 6, 2] | GRPMOD | As GICR_IGRPMODR0 register. |
| [1] + 4n:<br>[61, 57, 53, 49, 45, 41, 37, 33, 29, 25, 21, 17, 13, 9, 5, 1] | GRP | As GICR_IGROUPR0 register. |
| [0] + 4n:<br>[60, 56, 52, 48, 44, 40, 36, 32, 28, 24, 20, 16, 12, 8, 4, 0] | NSACR | 1 = Allow Non-secure access to interrupt <n>. |

### 4.5.4 Configuration ID0 Register, GICR_CFGID0

This register returns information about the configuration of the Redistributors.

The GICR_CFGID0 characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.5 Redistributor registers for SGIs and PPIs summary* on page 4-135.

The following figure shows the bit assignments.

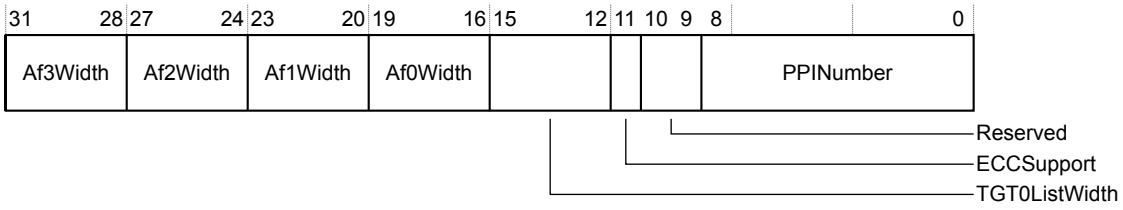


**Figure 4-25 GICR_CFGID0 bit assignments**

The following table shows the bit assignments.

**Table 4-33 GICR_CFGID0 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:28] | Af3Width | Affinity 3 width. |
| [27:24] | Af2Width | Affinity 2 width. |
| [23:20] | Af1Width | Affinity 1 width. |
| [19:16] | Af0Width | Affinity 0 width. |
| [15:12] | TGT0ListWidth | The Target0 list width - 1. |
| [11] | ECCSupport | 1 = ECC is supported. |
| [10:9] | - | Reserved, RAZ. |
| [8:0] | PPINumber | RedistributorID.<br>The **ppi_id[15:0]** tie-off signal sets the value of the ID. Each Redistributor must have a unique ID. |

***Related reference***
*A.6 Miscellaneous signals* on page Appx-A-195

### 4.5.5 Configuration ID1 Register, GICR_CFGID1

This register returns information about the configuration of the Redistributors.

The GICR_CFGID1 characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.5 Redistributor registers for SGIs and PPIs summary* on page 4-135.

The following figure shows the bit assignments.

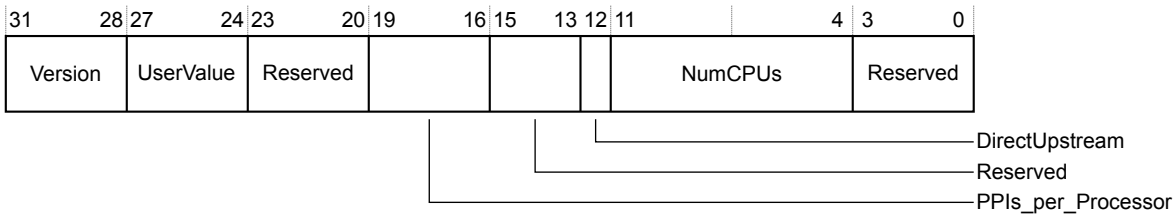


**Figure 4-26 GICR_CFGID1 bit assignments**

The following table shows the bit assignments.

**Table 4-34 GICR_CFGID1 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:28] | Version | Identifies the major and minor revisions, and product quality status of the GIC-600:<br>`0x1` = r0p0.<br>`0x3` = r0p1.<br>`0x4` = r0p2.<br>`0x5` = r1p1.<br>`0x6` = r0p3.<br>`0x7` = r1p2.<br>`0x8` = r1p3. |
| [27:24] | UserValue | Modification value that you can set. |
| [23:20] | - | Reserved, RAZ. |
| [19:16] | PPIs_per_Processor | The number of Redistributors that each core supports − 1. |
| [15:13] | - | Reserved. |
| [12] | DirectUpstream | Indicates a direct upstream connection. |
| [11:4] | NumCPUs | The number of cores that are integrated in this Redistributor. |
| [3:0] | - | Reserved, RAZ. |

## 4.6 ITS control register summary

The GIC-600 Interrupt Translation Service functions are controlled through registers that are identified with the prefix GITS.

For descriptions of registers that are not specific to the GIC-600, see the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

**Note**

This page does not exist in GIC-600 configurations that do not support LPIs or that do not have an ITS.

**Table 4-35 ITS control register summary**

| Offset | Name | Type | Width | Reset | Description | Architecture defined? |
|---|---|---|---|---|---|---|
| 0x0000 | GITS_CTLR | RW | 32 | 0x80000000 | ITS Control Register | Yes |
| 0x0004 | GITS_IIDR | RO | 32 | Configuration dependent | *4.6.1 ITS Implementer Identification Register, GITS_IIDR* on page 4-142 | Yes |
| 0x0008 | GITS_TYPER | RO | 64 | Configuration dependent | *4.6.2 Interrupt Controller Type Register, GITS_TYPER* on page 4-143 | Yes |
| 0x0010-0x001C | - | - | 32 | - | Reserved | - |
| 0x0020 | GITS_FCTLR | RW | 32 | 0x0 | *4.6.3 Function Control Register, GITS_FCTLR* on page 4-144 | k |
| 0x0024 | - | - | - | - | Reserved | - |
| 0x0028 | GITS_OPR | RW | 64 | 0x0 | *4.6.4 Operations Register, GITS_OPR* on page 4-146 | k |
| 0x0030 | GITS_OPSR | RO | 64 | 0x0 | *4.6.5 Operation Status Register, GITS_OPSR* on page 4-147 | k |
| 0x0038-0x007C | - | - | - | - | Reserved | - |
| 0x0080 | GITS_CBASER | RW | 64 | 0x0 | Command Queue Control Register<br>See the *Arm® GICv3 and GICv4 Software Overview* | Yes |
| 0x0088 | GITS_CWRITER | RW | 64 | 0x0 | Command Queue Write Pointer Register | Yes |
| 0x0090 | GITS_CREADR | RO | 64 | 0x0 | Command Queue Read Pointer Register | Yes |
| 0x0098-0x00FC | - | - | - | - | Reserved | - |
| 0x0100 | GITS_BASER0 | RW | 64 | 0x107000000000000 | ITS Translation Table Descriptor Register0 | Yes |
| 0x0108 | GITS_BASER1 | RW | 64 | 0x0 | ITS Translation Table Descriptor Register1 | Yes |
| 0x0110-0xEFFC | - | - | - | - | Reserved | - |

**Table 4-35 ITS control register summary (continued)**

| Offset | Name | Type | Width | Reset | Description | Architecture defined? |
|---|---|---|---|---|---|---|
| 0xF000 | GITS_CFGID | RO | 32 | 0x0 | *4.6.6 Configuration ID Register, GITS_CFGID* on page 4-148 | k |
| 0xF004-0xFFCC | - | - | - | - | Reserved | - |
| 0xFFD0 | GITS_PIDR4 | RO | 32 | 0x44 | Peripheral ID 4 Register | No |
| 0xFFD4 | GITS_PIDR5 | RO | 32 | 0x00 | Peripheral ID 5 Register | No |
| 0xFFD8 | GITS_PIDR6 | RO | 32 | 0x00 | Peripheral ID 6 Register | No |
| 0xFFDC | GITS_PIDR7 | RO | 32 | 0x00 | Peripheral ID 7 Register | No |
| 0xFFE0 | GITS_PIDR0 | RO | 32 | 0x94 | Peripheral ID 0 Register | No |
| 0xFFE4 | GITS_PIDR1 | RO | 32 | 0xB4 | Peripheral ID 1 Register | No |
| 0xFFE8 | GITS_PIDR2 | RO | 32 | 0x3B | *4.6.7 Peripheral ID2 Register, GITS_PIDR2* on page 4-149 | No |
| 0xFFEC | GITS_PIDR3 | RO | 32 | 0x00 | Peripheral ID 3 Register | No |
| 0xFFF0 | GITS_CIDR0 | RO | 32 | 0x0D | Component ID 0 Register | No |
| 0xFFF4 | GITS_CIDR1 | RO | 32 | 0xF0 | Component ID 1 Register | No |
| 0xFFF8 | GITS_CIDR2 | RO | 32 | 0x05 | Component ID 2 Register | No |
| 0xFFFC | GITS_CIDR3 | RO | 32 | 0xB1 | Component ID 3 Register | No |

This section contains the following subsections:

- *4.6.1 ITS Implementer Identification Register, GITS_IIDR* on page 4-142.
- *4.6.2 Interrupt Controller Type Register, GITS_TYPER* on page 4-143.
- *4.6.3 Function Control Register, GITS_FCTLR* on page 4-144.
- *4.6.4 Operations Register, GITS_OPR* on page 4-146.
- *4.6.5 Operation Status Register, GITS_OPSR* on page 4-147.
- *4.6.6 Configuration ID Register, GITS_CFGID* on page 4-148.
- *4.6.7 Peripheral ID2 Register, GITS_PIDR2* on page 4-149.

### 4.6.1 ITS Implementer Identification Register, GITS_IIDR

This register provides information about the implementer and revision of the ITS.

The GITS_IIDR characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations that have one or more ITS blocks.

**Attributes** See *4.6 ITS control register summary* on page 4-141.

The following figure shows the bit assignments.

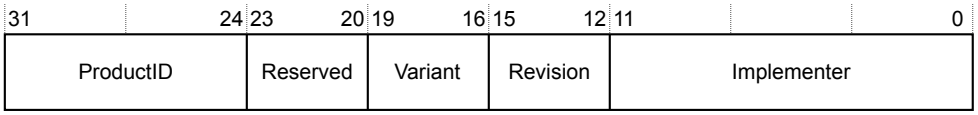


**Figure 4-27 GITS_IIDR bit assignments**

The following table shows the bit assignments.

**Table 4-36 GITS_IIDR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:24] | ProductID | Indicates the product ID:<br>`0x2` = GIC-600. |
| [23:20] | - | Reserved, RAZ. |
| [19:16] | Variant | Indicates the major revision or variant of the product for the `rmpn` identifier:<br>`0x0` = r0.<br>`0x1` = r1. |
| [15:12] | Revision | Indicates the minor revision of the product for the `rmpn` identifier:<br>`0x1` = p0.<br>`0x3` = p1.<br>`0x4` = p2.<br>`0x5` = p3. |
| [11:0] | Implementer | Identifies the implementer:<br>`0x43B` = Arm. |

### 4.6.2 Interrupt Controller Type Register, GITS_TYPER

This register returns information about the configuration of the GIC-600.

The GITS_TYPER characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations that have one or more ITS blocks.

**Attributes** See *4.6 ITS control register summary* on page 4-141.

The following figure shows the bit assignments.

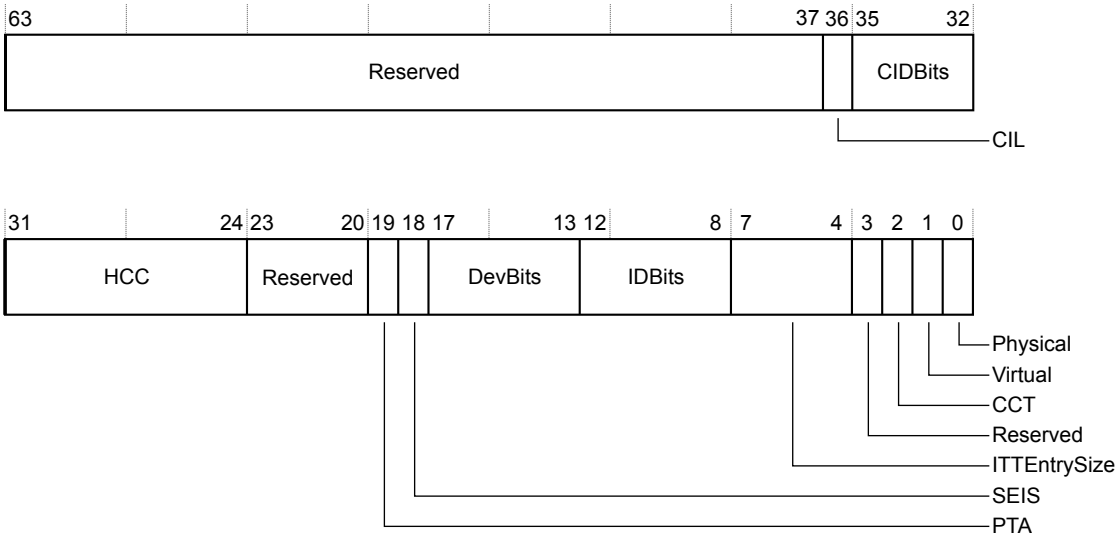


**Figure 4-28 GITS_TYPER bit assignments**

The following table shows the bit assignments.

**Table 4-37 GITS_TYPER bit assignments**

| Bits | Name | Function |
|---|---|---|
| [63:37] | - | Reserved, RAZ. |
| [36] | CIL | CollectionID limit:<br>1 = The GIC-600 supports CIL. |
| [35:32] | CIDBits | The number of CollectionID bits, minus one.<br>Set by the `col_width` configuration parameter. |
| [31:24] | HCC | Hardware collection count:<br>0 = All memory backed. |
| [23:20] | - | Reserved, returns 0. |
| [19] | PTA | Physical target addresses:<br>0 = The GIC-600 does not support physical target addresses. |
| [18] | SEIS | System error interrupts:<br>0 = The GIC-600 does not support locally generated System Error interrupts. |
| [17:13] | DevBits | The number of device identifier bits implemented, minus one.<br>Set by the `did_width` configuration parameter. |
| [12:8] | IDBits | The number of interrupt identifier bits implemented, minus one.<br>Set by the `vid_width` configuration parameter. |
| [7:4] | ITTEntrySize | The number of bytes per entry, minus one:<br>`0x3` = The GIC-600 supports a 4-byte ITT entry size. |
| [3] | - | Reserved. |
| [2] | CCT | Cumulative collection tables:<br>0 = Total number of supported collections is determined by the number of collections that are held in memory only. |
| [1] | Virtual | Virtual LPIs:<br>0 = The GIC-600 does not support Virtual LPIs.<br>See the *Arm® GICv3 and GICv4 Software Overview*. |
| [0] | Physical | Physical LPIs:<br>1 = The GIC-600 supports physical LPIs. |

### 4.6.3 Function Control Register, GITS_FCTLR

This register controls the scrubbing of all RAMs in the local GITS. The register is not distributed and only acts on the local chip.

The GITS_FCTLR characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations that have one or more ITS blocks.

**Attributes** See *4.6 ITS control register summary* on page 4-141.

The following figure shows the bit assignments.

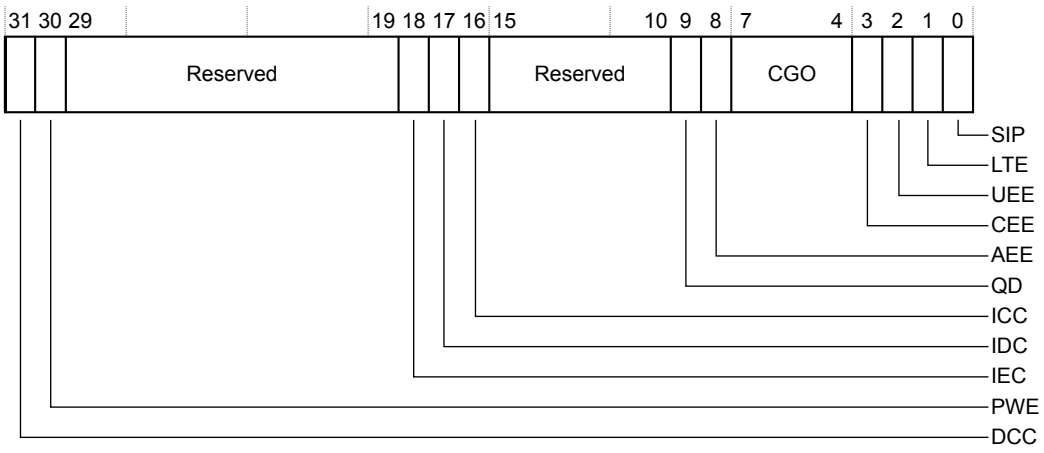


**Figure 4-29 GITS_FCTLR bit assignments**

The following table shows the bit assignments.

**Table 4-38 GITS_FCTLR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31] | DCC | Disable Cache Conversion:<br>0 = Use SMMU attribute for AMBA mapping.<br>1 = Use Direct attribute for AMBA mapping. |
| [30] | PWE | Powerdown While Enabled:<br>0 = Requests GITS_CTLR.Quiescent to indicate ITS is quiescent and can be powered down.<br>1 = Do not request GITS_CTLR.Quiescent to indicate ITS is quiescent. |
| [29:19] | - | Reserved, RAZ/WI. |
| [18] | IEC | Invalidate Event Cache (Write Only):<br>1 = Invalidate event cache.<br>0 = No effect. |
| [17] | IDC | Invalidate Device Cache (Write Only):<br>1 = Invalidate Device cache.<br>0 = No effect. |
| [16] | ICC | Invalidate Collection cache (Write Only):<br>1 = Invalidate Collection cache.<br>0 = No effect. |
| [15:10] | - | Reserved, RAZ/WI. |

**Table 4-38 GITS_FCTLR bit assignments (continued)**

| Bits | Name | Function |
|---|---|---|
| [9] | QD | Q Deny:<br>1 = Always deny Q-Channel requests.<br>0 = Do not deny Q-Channel requests. |
| [8] | AEE | Access Error Enable:<br>1 = Enable reporting of slave access errors.<br>0 = Do not enable reporting of slave access errors. |
| [7:4] | CGO | One bit per clock gate:<br>CGO[0] = CCS, Translation logic.<br>CGO[1] = Command.<br>CGO[2] = Debug.<br>CGO[3] = Map Fetch.<br>——— **Note** ———<br>CGO must be set if clock gates are not implemented.<br>——— |
| [3] | CEE | Command error enable:<br>0 = Do not enable reporting of command errors.<br>1 = Enable reporting of command errors. |
| [2] | UEE | Unmapped error enable:<br>0 = Do not enable reporting of unmapped interrupt errors.<br>1 = Enable reporting of unmapped interrupt errors. |
| [1] | LTE | Latency tracking enable:<br>0 = Disable latency tracking of interrupts.<br>1 = Enable latency tracking of interrupts. |
| [0] | SIP | Scrub in progress:<br>0 = No scrub in progress.<br>1 = Scrub in progress.<br>This bit is read and written by software. When a scrub is complete, the GIC clears the bit to 0. |

### 4.6.4 Operations Register, GITS_OPR

This register controls cache lock.

The GITS_OPR characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations that have one or more ITS blocks.

**Attributes** See *4.6 ITS control register summary* on page 4-141.

The following figure shows the bit assignments.

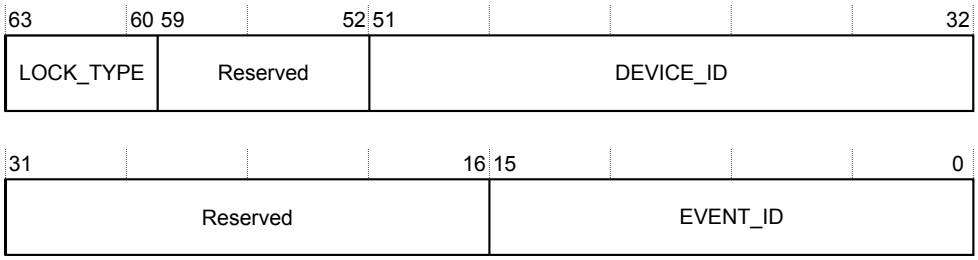


**Figure 4-30 GITS_OPR bit assignments**

The following table shows the bit assignments.

**Table 4-39 GITS_OPR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [63:60] | LOCK_TYPE | Lock type supported:<br>0 = Track.<br>1 = Trial.<br>2 = ITS lock.<br>3 = ITS unlock.<br>4 = Track abort.<br>6 = Reserved.<br>7 = Reserved.<br>8 = ITS unlock all. |
| [59:52] | - | Reserved. |
| [51:32] | DEVICE_ID | 0-maximum DeviceID supported. |
| [31:16] | - | Reserved. |
| [15:0] | EVENT_ID | 8192-maximum EventID supported. |

### 4.6.5 Operation Status Register, GITS_OPSR

This register indicates cache lock status.

The GITS_OPSR characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations that have one or more ITS blocks.

**Attributes** See *4.6 ITS control register summary* on page 4-141.

The following figure shows the bit assignments.

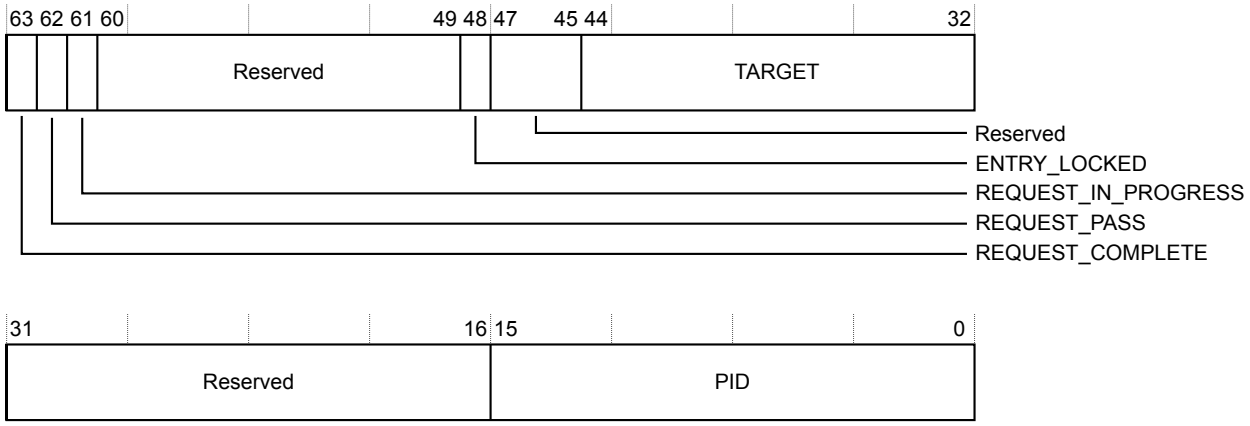


**Figure 4-31 GITS_OPSR bit assignments**

The following table shows the bit assignments.

**Table 4-40 GITS_OPSR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [63] | REQUEST_COMPLETE | Request to GITS_OPR completed. |
| [62] | REQUEST_PASS | Request to GITS_OPR completed without error. |
| [61] | REQUEST_IN_PROGRESS | Translation in progress. |
| [60:49] | - | Reserved. |
| [48] | ENTRY_LOCKED | Locked entry in cache corresponds to request. |
| [47:45] | - | Reserved. |
| [44:32] | TARGET | Target of interrupt requested. |
| [31:16] | - | Reserved. |
| [15:0] | PID | Physical ID of interrupt requested. |

### 4.6.6 Configuration ID Register, GITS_CFGID

This register returns the ID number of the ITS block.

The GITS_CFGID characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations that have one or more ITS blocks.

**Attributes** See *4.6 ITS control register summary* on page 4-141.

The following figure shows the bit assignments.

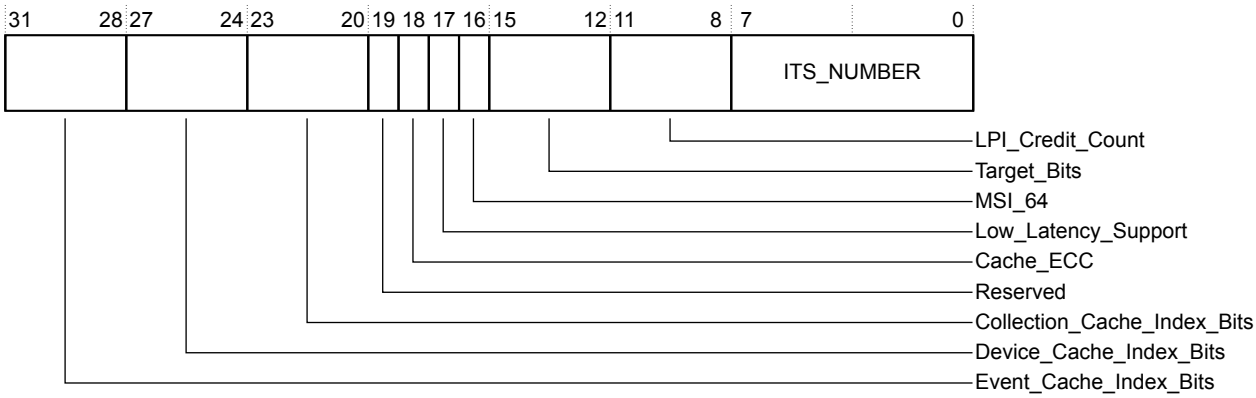


**Figure 4-32 GITS_CFGID bit assignments**

The following table shows the bit assignments.

**Table 4-41 GITS_CFGID bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:28] | Event_Cache_Index_Bits | Number of bits used to index Event cache. |
| [27:24] | Device_Cache_Index_Bits | Number of bits used to index Device cache. |
| [23:20] | Collection_Cache_Index_Bits | Number of bits used to index Collection cache. |
| [19] | - | Reserved. |
| [18] | Cache_ECC | Translation caching has ECC protection. |
| [17] | Low_Latency_Support | Lock translations in cache support. |
| [16] | MSI_64 | MSI-64 Encapsulator support. |
| [15:12] | Target_Bits | Number of bits supported for targets. |
| [11:8] | LPI_Credit_Count | <Number of LPI credits> − 1. |
| [7:0] | ITS_Number | Returns the ITS block ID. The **its_id[7:0]** tie-off signal controls the ID value. Each ITS block must have a unique ID. |

***Related reference***

*A.6 Miscellaneous signals* on page Appx-A-195

### 4.6.7 Peripheral ID2 Register, GITS_PIDR2

This register returns byte[2] of the peripheral ID. The GITS_PIDR2 register is part of the set of ITS peripheral identification registers.

The GITS_PIDR2 characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations that have one or more ITS blocks.

**Attributes** See *4.6 ITS control register summary* on page 4-141.

The following figure shows the bit assignments.

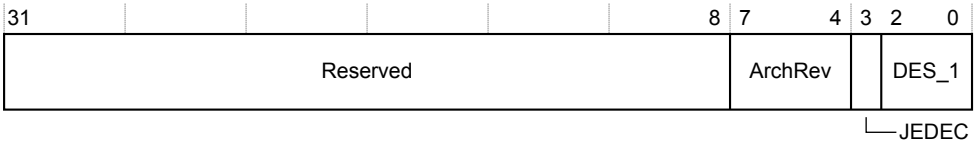


**Figure 4-33 GITS_PIDR2 bit assignments**

The following table shows the bit assignments.

**Table 4-42 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-600 complies:<br>• `0x3` = GICv3. |
| [3] | JEDEC | Indicates that a JEDEC-assigned JEP106 identity code is used. |
| [2:0] | DES_1 | Bits[6:4] of the JEP106 identity code. Bits[3:0] of the JEP106 identity code are assigned to GITS_PIDR1. |

***Related information***

*Arm® GICv3 and GICv4 Software Overview*

## 4.7 ITS translation register summary

Interrupts to be translated by the GIC-600 Interrupt Translation Service are identified by EventIDs that are written to the ITS translation register GITS_TRANSLATER.

For descriptions of registers that are not specific to the GIC-600, see the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.

**Note**

This page does not exist in GIC-600 configurations that do not support LPIs or that do not have an ITS.

**Table 4-43 ITS translation register summary**

| Offset | Name | Type | Reset | Description | Architecture defined? |
|---|---|---|---|---|---|
| 0x0000-0x003C | - | - | - | Reserved | - |
| 0x0040 | GITS_TRANSLATER | WO | - | ITS Translation Register | Yes |
| 0x0044-0xFFFC | - | - | - | Reserved | - |

## 4.8 GICT register summary

The GIC-600 trace and debug functions are controlled through registers that are identified with the prefix GICT.

**Note**

The GICD_SAC.GICTNS bit controls whether Non-secure software can access the GICT registers.

**Table 4-44 GICT register summary**

| Offset | Name | Type | Width | Reset | Description | RAS ? |
|---|---|---|---|---|---|---|
| 0x0000 + (n × 64) | GICT_ERR<n>FR | RO | 64 | Record dependent | *4.8.1 Error Record Feature Register, GICT_ERR<n>FR* on page 4-154 | Yes |
| 0x0008 + (n × 64) | GICT_ERR<n>CTLR | RW | 64 | 0x0 | *4.8.2 Error Record Control Register, GICT_ERR<n>CTLR* on page 4-155 | Yes |
| 0x0010 + (n × 64) | GICT_ERR<n>STATUS | RW | 64 | Record dependent | *4.8.3 Error Record Primary Status Register, GICT_ERR<n>STATUS* on page 4-157 | Yes |
| 0x0018 + (n × 64) | GICT_ERR<n>ADDR | RW | 64 | UNKNOWN | *4.8.4 Error Record Address Register, GICT_ERR<n>ADDR* on page 4-158 | Yes |
| 0x0020 + (n × 64) | GICT_ERR<n>MISC0 | RW | 64 | UNKNOWN | *4.8.5 Error Record Miscellaneous Register 0, GICT_ERR<n>MISC0* on page 4-159 | Yes |
| 0x0028 + (n × 64) | GICT_ERR<n>MISC1 | RW | 64 | UNKNOWN | *4.8.6 Error Record Miscellaneous Register 1, GICT_ERR<n>MISC1* on page 4-166 | Yes |
| 0xE000 | GICT_ERRGSR | RO | 64 | 0x0 | *4.8.7 Error Group Status Register, GICT_ERRGSR* on page 4-167 | Yes |
| 0xE008-0xE7FC | - | - | - | - | Reserved, RAZ/WI | - |
| 0xE800-0xE808 | GICT_ERRIRQCR<n> | RW | 64 | 0x0 | *4.8.8 Error Interrupt Configuration Registers, GICT_ERRIRQCR<n>* on page 4-167 | Yes |
| 0xE810-0xFFB8 | - | - | - | - | Reserved, RAZ/WI | - |
| 0xFFBC | GICT_DEVARCH | RO | 32 | 0x47700A00 | Device Architecture register | Yes |
| 0xFFC0-0xFFC4 | - | - | - | - | Reserved, RAZ/WI | - |
| 0xFFC8 | GICT_ERRIDR | RO | 32 | Configuration dependent | *4.8.9 Error Record ID Register, GICT_ERRIDR* on page 4-168 | Yes |
| 0xFFCC | - | - | - | - | Reserved, RAZ/WI | - |
| 0xFFD0 | GICT_PIDR4 | RO | 32 | 0x44 | Peripheral ID 4 Register | No |
| 0xFFD4 | GICT_PIDR5 | RO | 32 | 0x00 | Peripheral ID 5 Register | No |
| 0xFFD8 | GICT_PIDR6 | RO | 32 | 0x00 | Peripheral ID 6 Register | No |
| 0xFFDC | GICT_PIDR7 | RO | 32 | 0x00 | Peripheral ID 7 Register | No |
| 0xFFE0 | GICT_PIDR0 | RO | 32 | 0x95 | Peripheral ID 0 Register | No |

**Table 4-44 GICT register summary (continued)**

| Offset | Name | Type | Width | Reset | Description | RAS ? |
|---|---|---|---|---|---|---|
| 0xFFE4 | GICT_PIDR1 | RO | 32 | 0xB4 | Peripheral ID 1 Register | No |
| 0xFFE8 | GICT_PIDR2 | RO | 32 | 0x3B | *4.8.10 Peripheral ID2 Register, GICT_PIDR2* on page 4-168 | No |
| 0xFFEC | GICT_PIDR3 | RO | 32 | 0x00 | Peripheral ID 3 Register | No |
| 0xFFF0 | GICT_CIDR0 | RO | 32 | 0x0D | Component ID 0 Register | No |
| 0xFFF4 | GICT_CIDR1 | RO | 32 | 0xF0 | Component ID 1 Register | No |
| 0xFFF8 | GICT_CIDR2 | RO | 32 | 0x05 | Component ID 2 Register | No |
| 0xFFFC | GICT_CIDR3 | RO | 32 | 0xB1 | Component ID 3 Register | No |

The following table lists the error records for the various error conditions.

**Table 4-45 Error records**

| Record | Description | Type | Syndrome (SERR) |
|---|---|---|---|
| 0 | Software error in GICD programming | UEO[ac] | See *Table 3-8 Software errors, record 0* on page 3-78. |
| 1 | Correctable SPI RAM errors | CE[ad] | 7, Data value from associative memory.<br>See *Table 3-9 SPI RAM errors, records 1-2* on page 3-85. |
| 2 | Uncorrectable SPI RAM errors | UER[ae] | 7, Data value from associative memory.<br>See *Table 3-9 SPI RAM errors, records 1-2* on page 3-85. |
| 3 | Correctable SGI RAM errors | CE[ad] | 7, Control value from associative memory.<br>See *Table 3-10 SGI RAM errors, records 3-4* on page 3-86. |
| 4 | Uncorrectable SGI RAM errors | UER[ae] | 7, Control value from associative memory.<br>See *Table 3-10 SGI RAM errors, records 3-4* on page 3-86. |
| 5 | Reserved | - | - |
| 6 | Reserved | - | - |
| 7 | Correctable PPI RAM errors | CE[ad] | 7, Control value from associative memory.<br>See *Table 3-11 PPI RAM errors, records 7-8* on page 3-87. |
| 8 | Uncorrectable PPI RAM errors | UER[ae] | 7, Control value from associative memory.<br>See *Table 3-11 PPI RAM errors, records 7-8* on page 3-87. |
| 9 | Correctable LPI RAM errors | CE[ad] | 7, Control value from associative memory.<br>See *Table 3-12 LPI RAM errors, records 9-10* on page 3-87. |

[ac] Restartable error and contained.
[ad] Correctable error.
[ae] Recoverable error.

**Table 4-45 Error records (continued)**

| Record | Description | Type | Syndrome (SERR) |
|---|---|---|---|
| 10 | Uncorrectable LPI RAM errors | UER[ae] | 7, Control value from associative memory.<br>See *Table 3-12 LPI RAM errors, records 9-10* on page 3-87. |
| 11 | Correctable error from ITS RAM | CE[ad] | 6, Data value from associative memory.<br>See *Table 3-13 ITS RAM errors, records 11-12* on page 3-88. |
| 12 | Uncorrectable error from ITS RAM | UEO[ac] | 6, Data value from associative memory.<br>See *Table 3-13 ITS RAM errors, records 11-12* on page 3-88. |
| 13 + ITSnum | ITS command and translation errors | UER[ae] | 14, Illegal Access.<br>See *Table 3-15 ITS command and translation errors, records 13+* on page 3-89. |

This section contains the following subsections:

- *4.8.1 Error Record Feature Register, GICT_ERR<n>FR* on page 4-154.
- *4.8.2 Error Record Control Register, GICT_ERR<n>CTLR* on page 4-155.
- *4.8.3 Error Record Primary Status Register, GICT_ERR<n>STATUS* on page 4-157.
- *4.8.4 Error Record Address Register, GICT_ERR<n>ADDR* on page 4-158.
- *4.8.5 Error Record Miscellaneous Register 0, GICT_ERR<n>MISC0* on page 4-159.
- *4.8.6 Error Record Miscellaneous Register 1, GICT_ERR<n>MISC1* on page 4-166.
- *4.8.7 Error Group Status Register, GICT_ERRGSR* on page 4-167.
- *4.8.8 Error Interrupt Configuration Registers, GICT_ERRIRQCR<n>* on page 4-167.
- *4.8.9 Error Record ID Register, GICT_ERRIDR* on page 4-168.
- *4.8.10 Peripheral ID2 Register, GICT_PIDR2* on page 4-168.

### 4.8.1 Error Record Feature Register, GICT_ERR<n>FR

This register returns information about the ARMv8.2 RAS features that the GIC-600 implements.

The GICT_ERR<n>FR characteristics are:

**Usage constraints** If GICD_SAC.GICTNS == 0, then only Secure software can access the contents of this register.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.8 GICT register summary* on page 4-152.

The following figure shows the bit assignments.

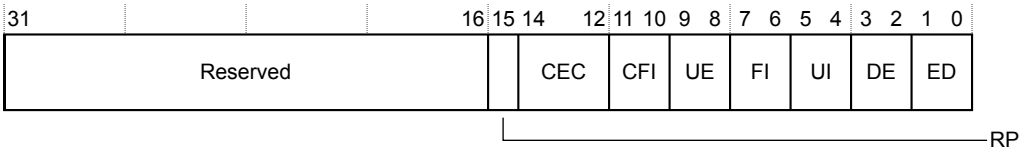


**Figure 4-34 GICT_ERR<n>FR bit assignments**

The following table shows the bit assignments.

**Table 4-46 GICT_ERR<n>FR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:16] | - | Reserved, RAZ. |
| [15] | RP | Repeat corrected error count:<br>0 = The GIC-600 does not implement a repeat corrected error counter. |
| [14:12] | CEC | Corrected error count:<br>`0b000` = The GIC-600 does not implement a standard Corrected error counter in GICT_ERR<n>MISC0. |
| [11:10] | CFI | Corrected errors fault interrupt. Depending on the configuration, returns either:<br>`0b00` = The GIC-600 does not provide a fault handling interrupt for corrected errors.<br>`0b10` = The GIC-600 provides a controllable fault handling interrupt for corrected errors. |
| [9:8] | UE | Uncorrected error. Depending on the configuration, returns either:<br>`0b00` = The GIC-600 does not provide an in-band uncorrected error reporting.<br>`0b10` = The GIC-600 provides a controllable in-band uncorrected error reporting. |
| [7:6] | FI | Fault handling interrupt for uncorrected errors. Depending on the configuration, returns either:<br>`0b00` = The GIC-600 does not provide a fault handling interrupt.<br>`0b10` = The GIC-600 provides a controllable fault handling interrupt. |
| [5:4] | UI | Error recovery interrupt for uncorrected errors. Depending on the configuration, returns either:<br>`0b00` = The GIC-600 does not provide an error recovery interrupt for uncorrected errors.<br>`0b10` = The GIC-600 provides a controllable error recovery interrupt for uncorrected errors. |
| [3:2] | DE | Deferring of errors support:<br>`0b00` = The GIC-600 does not support the deferring of errors. |
| [1:0] | ED | Uncorrected error reporting:<br>`0b01` = Uncorrected error reporting is always enabled. |

### 4.8.2 Error Record Control Register, GICT_ERR<n>CTLR

This register controls how interrupts are handled.

The GICT_ERR<n>CTLR characteristics are:

**Usage constraints** If GICD_SAC.GICTNS == 0, then only Secure software can access the functions of this register.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.8 GICT register summary* on page 4-152.

The following figure shows the bit assignments.

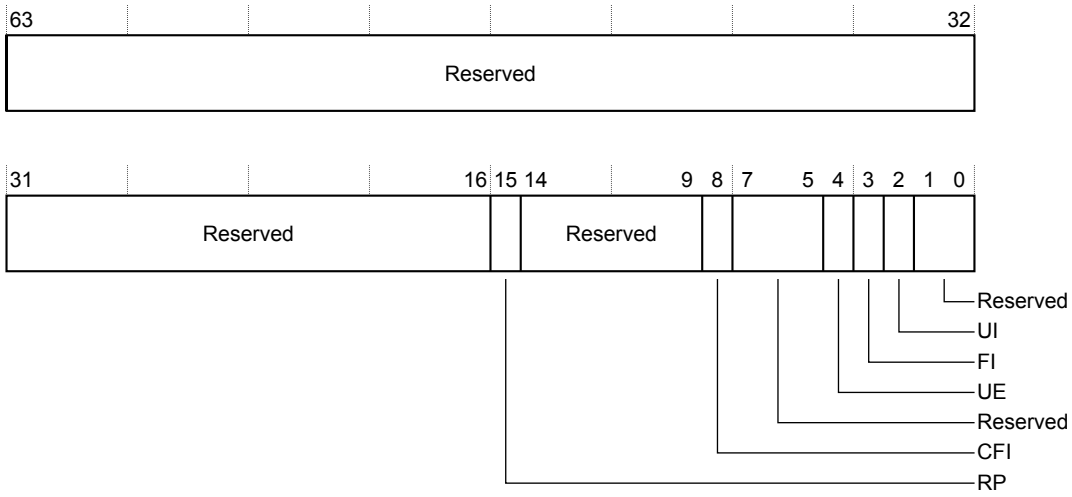


**Figure 4-35 GICT_ERR<n>CTLR bit assignments**

The following table shows the bit assignments.

**Table 4-47 GICT_ERR<n>CTLR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [63:16] | - | Reserved, RAZ. |
| [15] | RP | 0 = An error response to a transaction is reported. |
| [14:9] | - | Reserved, RAZ. |
| [8] | CFI | Controls whether a corrected error generates a fault handling interrupt.<br>SBZ on non-correctable errors else:<br>0 = The GIC-600 does not assert a fault handling interrupt for corrected errors.<br>1 = The GIC-600 asserts a fault handling interrupt when a corrected error occurs. |
| [7:5] | - | Reserved, RAZ. |
| [4] | UE | Uncorrected error.<br>RAZ/WI for all records except GICT error record (0) else:<br>0 = Do not send external abort with transaction.<br>1 = Send external abort with transaction. |
| [3] | FI | Fault handling interrupt.<br>SBZ on *Correctable Error* (CE) records else:<br>0 = Fault handling interrupt is not generated on any error.<br>1 = Fault handling interrupt is generated on all uncorrectable errors. |
| [2] | UI | Error recovery interrupt for uncorrected error.<br>SBZ on CE records else:<br>0 = Error recovery interrupt is not generated on any error.<br>1 = Error recovery interrupt is generated on all uncorrectable errors. |
| [1:0] | - | Reserved, RAZ. |

### 4.8.3 Error Record Primary Status Register, GICT_ERR<n>STATUS

This register indicates information relating to the recorded errors.

The GICT_ERR<n>STATUS characteristics are:

**Usage constraints** If GICD_SAC.GICTNS == 0, then only Secure software can access the functions of this register.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.8 GICT register summary* on page 4-152.

The following figure shows the bit assignments.

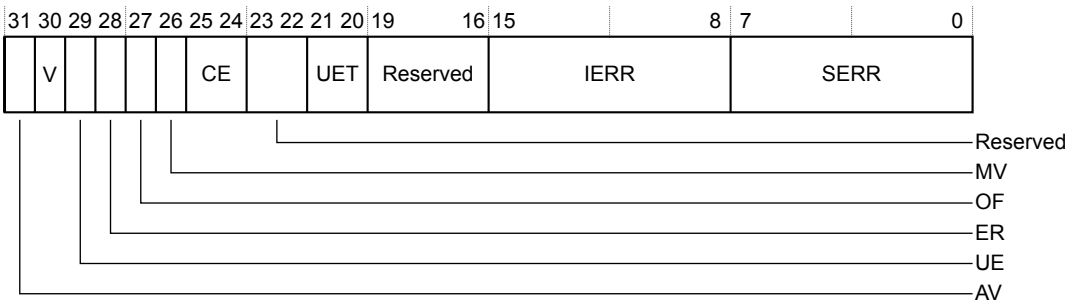


**Figure 4-36 GICT_ERR<n>STATUS bit assignments**

The following table shows the bit assignments.

**Table 4-48 GICT_ERR<n>STATUS bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31] | AV | Indicates if the address is valid:<br>0 = GICT_ERR<n>ADDR is not valid.<br>1 = GICT_ERR<n>ADDR contains an address that is associated with the highest priority error that this record stores. Only present in record 0. |
| [30] | V | Indicates if this register is valid:<br>0 = GICT_ERR<n>STATUS is not valid.<br>1 = GICT_ERR<n>STATUS is valid. One or more errors are recorded. |
| [29] | UE | Uncorrectable error bit.<br>SBZ in *Correctable Error* (CE) records. |
| [28] | ER | Indicates that at least one error has been reported over ACE-Lite.<br>Set for record 0 only, and only for accesses to corrupted data, and bad incoming access. |
| [27] | OF | Record has overflowed. |
| [26] | MV | Indicates if the GICT miscellaneous registers are valid:<br>0 = GICT_ERR<n>MISC0 and GICT_ERR<n>MISC1 are not valid.<br>1 = GICT_ERR<n>MISC0 and GICT_ERR<n>MISC1 are valid. |

**Table 4-48 GICT_ERR<n>STATUS bit assignments (continued)**

| Bits | Name | Function |
|---|---|---|
| [25:24] | CE | Correctable Error. Indicates errors that are correctable as shown in *Table 4-45 Error records* on page 4-153:<br>`0b00` = No CE recorded.<br>`0b10` = At least one CE recorded. |
| [23:22] | - | Reserved, RAZ/WI. |
| [21:20] | UET | RES0 unless UE == 1, in which case:<br>`0b10` = UEO.<br>`0b11` = UER. |
| [19:16] | - | Reserved, RAZ/WI. |
| [15:8] | IERR | IMPLEMENTATION-DEFINED error code:<br>Returns information shown in *Table 4-51 Data field encoding* on page 4-160.<br>This field is RO in Record 0. |
| [7:0] | SERR | Architecturally defined primary error code:<br>Returns information shown in *Table 4-51 Data field encoding* on page 4-160.<br>This field is RO in Record 0. |

### 4.8.4 Error Record Address Register, GICT_ERR<n>ADDR

This register contains the address and security status of the write. This register is only present for GICT software record 0.

The GICT_ERR<n>ADDR characteristics are:

**Usage constraints** If GICD_SAC.GICTNS == 0, then only Secure software can access the functions of this register.

Ignores writes if ERR<n>STATUS.AV == 1.

All bits are RAZ/WI if GICT_ERR<n>STATUS.IERR = 0, 12, or 13.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.8 GICT register summary* on page 4-152.

The following figure shows the bit assignments.

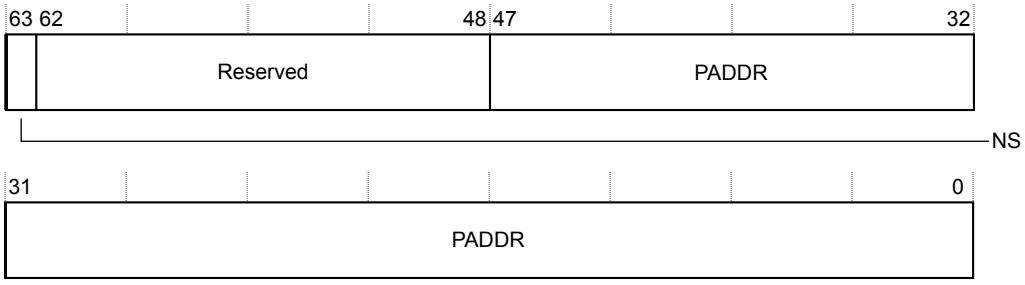


**Figure 4-37 GICT_ERR<n>ADDR bit assignments**

The following table shows the bit assignments.

**Table 4-49 GICT_ERR<n>ADDR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [63] | NS | Non-secure attribute:<br>0 = The address is Secure.<br>1 = The address is Non-secure. |
| [62:48] | - | Reserved, RAZ/WI. |
| [47:0] | PADDR | The error address. |

### 4.8.5 Error Record Miscellaneous Register 0, GICT_ERR<n>MISC0

This register contains the Corrected error counter and information that assists with identifying the RAM in which the error was detected.

The GICT_ERR<n>MISC0 characteristics are:

**Usage constraints** If GICD_SAC.GICTNS == 0, then only Secure software can access the functions of this register.

If GICT_ERR<n>STATUS.MV == 1, then the GICT_ERR<n>MISC0 ignores writes to the Data field.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.8 GICT register summary* on page 4-152.

The following figure shows the bit assignments.

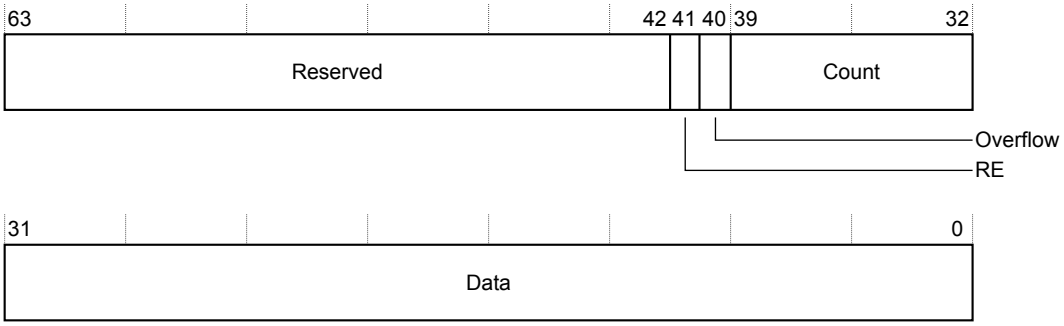


**Figure 4-38 GICT_ERR<n>MISC0 bit assignments**

The following table shows the bit assignments.

**Table 4-50 GICT_ERR<n>MISC0 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [63:42] | - | Reserved, RAZ. |
| [41] | RE | Rounding Error.<br>The Rounding Error counter is under-reporting. |

**Table 4-50 GICT_ERR<n>MISC0 bit assignments (continued)**

| Bits | Name | Function |
|---|---|---|
| [40] | Overflow | Sticky overflow bit:<br>0 = Counter has not overflowed.<br>1 = Counter has overflowed.<br>If the corrected fault handling interrupt is enabled, then the GIC-600 generates a fault handling interrupt. |
| [39:32] | Count | Corrected error count.<br>Error counter is not 0 or is more than 134. Incremented for each corrected error that does not match the recorded syndrome. |
| [31:0] | Data | Information associated with the error. A description of each error code is given in one of the following tables:<br>• *Table 3-8 Software errors, record 0* on page 3-78.<br>• *Table 3-9 SPI RAM errors, records 1-2* on page 3-85.<br>• *Table 3-10 SGI RAM errors, records 3-4* on page 3-86.<br>• *Table 3-11 PPI RAM errors, records 7-8* on page 3-87.<br>• *Table 3-12 LPI RAM errors, records 9-10* on page 3-87.<br>• *Table 3-13 ITS RAM errors, records 11-12* on page 3-88.<br>• *Table 3-15 ITS command and translation errors, records 13+* on page 3-89. |

The following table shows the Data field encoding for each error record and syndrome.

**Table 4-51 Data field encoding**

| Record | GICT_ERR<n>STATUS.IERR (syndrome) | GICT_ERR<n>STATUS.SERR | Value and description of GICT_ERR<n>MISC0.Data (Other bits RES0)<br>Always packed from 0 (lowest = 0) |
|---|---|---|---|
| Software Error (0) | `0x0`, SYN_ACE_BAD<br>Illegal ACE-Lite Slave Access. | `0xE` | AccessRnW, bit[12].<br>AccessSparse, bit[11].<br>AccessSize, bits[10:8].<br>AccessLength, bits[7:0]. |
| Software Error (0) | `0x1`, SYN_PPI_PWRDWN<br>Attempt to access a powered down Redistributor. | `0xF` | Redistributor, bits[24:16].<br>Core, bits[8:0]. |
| Software Error (0) | `0x2`, SYN_PPI_PWRCHANGE<br>Attempt to power down Redistributor rejected. | `0xF` | Redistributor, bits[24:16].<br>Core, bits[8:0]. |
| Software Error (0) | `0x3`, SYN_GICR_ARE<br>Attempt to access GICR or GICD registers in mode that cannot work. | `0xF` | Core, bits[8:0]. |

**Table 4-51 Data field encoding (continued)**

| Record | GICT_ERR<n>STATUS.IERR (syndrome) | GICT_ERR<n>STATUS.SERR | Value and description of GICT_ERR<n>MISC0.Data (Other bits RES0)<br>Always packed from 0 (lowest = 0) |
|---|---|---|---|
| Software Error (0) | 0x4, SYN_PROPBASE_ACC<br>Attempt to reprogram PROPBASE registers to a value that is not accepted because another value is already in use. | 0xF | Core, bits[8:0]. |
| Software Error (0) | 0x5, SYN_PENDBASE_ACC<br>Attempt to reprogram PENDBASE registers to a value that is not accepted because another value is already in use. | 0xF | Core, bits[8:0]. |
| Software Error (0) | 0x6, SYN_LPI_CLR<br>Attempt to reprogram ENABLE_LPI when not enabled and not asleep. | 0xF | Core, bits[8:0]. |
| Software Error (0) | 0x7, SYN_WAKER_CHANGE<br>Attempt to change GICR_WAKER abandoned due to handshake rules. | 0xF | Core, bits[8:0]. |
| Software Error (0) | 0x8, SYN_SLEEP_FAIL<br>Attempt to put GIC to sleep failed because cores are not fully asleep. | 0xF | Core, bits[8:0]. |
| Software Error (0) | 0x9, SYN_PGE_ON_QUIESCE<br>Core put to sleep before its Group enables were cleared. | 0xF | Core, bits[8:0]. |
| Software Error (0) | 0xA, SYN_GICD_CTLR<br>Attempt to update GICD_CTLR was prevented due to RWP or Group enable restrictions. | 0xF | Data, bits[7:0]. |
| Software Error (0) | 0x10, SYN_SGI_NO_TGT<br>SGI sent with no valid destinations. | 0xE | Core, bits[8:0]. |
| Software Error (0) | 0x11,<br>SYN_SGI_CORRUPTED<br>SGI corrupted without effect. | 0x6 | Core, bits[8:0]. |
| Software Error (0) | 0x12,<br>SYN_GICR_CORRUPTED<br>Data was read from GICR register space that encountered an uncorrectable error. | 0x6 | GICT_ERR0ADDR is populated. |

**Table 4-51 Data field encoding (continued)**

| Record | GICT_ERR<n>STATUS.IERR (syndrome) | GICT_ERR<n>STATUS.SERR | Value and description of GICT_ERR<n>MISC0.Data (Other bits RES0)<br>Always packed from 0 (lowest = 0) |
|---|---|---|---|
| Software Error (0) | 0x13,<br>SYN_GICD_CORRUPTED<br>Data was read from GICD register space that encountered an uncorrectable error. | 0x6 | GICT_ERR0ADDR is populated. |
| Software Error (0) | 0x14,<br>SYN_ITS_OFF<br>Data was read from an ITS that is powered down. | 0xF | GICT_ERR0ADDR is populated. |
| Software Error (0) | 0x18, SYN_SPI_BLOCK.<br>Attempt to access a SPI block that is not implemented. | 0xE | Block, bits[4:0]. |
| Software Error (0) | 0x19, SYN_SPI_OOR<br>Attempt to access a non-implemented SPI using (SET\|CLR)SPI. | 0xE | ID, bits[9:0]. |
| Software Error (0) | 0x1A, SYN_SPI_NO_DEST_TGT<br>A SPI has no legal target destinations. | 0xF | ID, bits[9:0]. |
| Software Error (0) | 0x1B, SYN_SPI_NO_DEST_1OFN<br>A 1 of N SPI cannot be delivered due to bad DPG/GICR_CLASS programming. | 0xF | ID, bits[9:0]. |
| Software Error (0) | 0x1C, SYN_COL_OOR<br>A collator message is received for a non-implemented SPI, or is larger than the number of owned SPIs in a multichip configuration. | 0xF | ID, bits[9:0]. |
| Software Error (0) | 0x1D, SYN_DEACT_IN<br>A Deactivate to a non-existent SPI, or with incorrect groups set. Deactivates to LPI and non-existent PPI are not reported. | 0xE | None. |
| Software Error (0) | 0x1E, SYN_SPI_CHIP_OFFLINE<br>An attempt was made to send a SPI to an offline chip. | 0xF | ID, bits[9:0]. |

**Table 4-51 Data field encoding (continued)**

| Record | GICT_ERR<n>STATUS.IERR (syndrome) | GICT_ERR<n>STATUS.SERR | Value and description of GICT_ERR<n>MISC0.Data (Other bits RES0)<br>Always packed from 0 (lowest = 0) |
|---|---|---|---|
| Software Error (0) | 0x28, SYN_ITS_REG_SET_OOR<br>An attempt was made to set an *Out Of Range* (OOR) interrupt. Only valid when GICR LPI injection registers are supported. | 0xE | Core, bits[24:16].<br>Data, bits[15:0]. |
| Software Error (0) | 0x29, SYN_ITS_REG_CLR _OOR<br>An attempt was made to clear an OOR interrupt. Only valid when GICR LPI injection registers are supported. | 0xE | Core, bits[24:16].<br>Data, bits[15:0]. |
| Software Error (0) | 0x2A, SYN_ITS_REG_INV_OOR<br>An attempt was made to invalidate an OOR interrupt. Only valid when GICR LPI injection registers are supported. | 0xE | Core, bits[24:16].<br>Data, bits[15:0]. |
| Software Error (0) | 0x2B, SYN_ITS_REG_SET_ENB<br>An attempt was made to set an interrupt when LPIs are not enabled. Only valid when GICR LPI injection registers are supported. | 0xF | Core, bits[24:16].<br>Data, bits[15:0]. |
| Software Error (0) | 0x2C, SYN_ITS_REG_CLR _ENB<br>An attempt was made to clear an interrupt when LPIs are not enabled. Only valid when GICR LPI injection registers are supported. | 0xF | Core, bits[24:16].<br>Data, bits[15:0]. |
| Software Error (0) | 0x2D, SYN_ITS_REG_INV_ENB<br>An attempt was made to invalidate an interrupt when LPIs are not enabled. Only valid when GICR LPI injection registers are supported. | 0xF | Core, bits[24:16].<br>Data, bits[15:0]. |
| Software Error (0) | 0x40, SYN_LPI_PROP_READ_FAIL<br>An attempt was made to read properties for a single interrupt where an error response was received with the data. | 0x12 | Target, bits[31:16].<br>ID, bits[15:0]. |
| Software Error (0) | 0x41, SYN_PT_PROP_READ_FAIL<br>An attempt was made to read properties for a block of interrupts where an error response was received with the data. | 0x12 | Target, bits[31:16].<br>ID, bits[15:0]. |

**Table 4-51 Data field encoding (continued)**

| Record | GICT_ERR<n>STATUS.IERR (syndrome) | GICT_ERR<n>STATUS.SERR | Value and description of GICT_ERR<n>MISC0.Data (Other bits RES0)<br>Always packed from 0 (lowest = 0) |
|---|---|---|---|
| Software Error (0) | 0x42, SYN_PT_COARSE_MAP_READ_FAIL<br>An attempt was made to read the coarse map for a target where an error response was received with the data. | 0x12 | Target, bits[31:16]. |
| Software Error (0) | 0x43, SYN_PT_COARSE_MAP_WRITE_FAIL<br>An attempt was made to write the coarse map for a target with an error received with the write response. | 0x12 | Target, bits[31:16]. |
| Software Error (0) | 0x44, SYN_PT_TABLE_READ_FAIL<br>An attempt was made to read a block of interrupts from a Pending table, where an error response was received with the data. | 0x12 | Target, bits[31:16].<br>ID, bits[15:0]. |
| Software Error (0) | 0x45, SYN_PT_TABLE_WRITE_FAIL<br>An attempt was made to write back a block of interrupts from a Pending table with an error received with the write response. | 0x12 | Target, bits[31:16].<br>ID, bits[15:0]. |
| Software Error (0) | 0x46, SYN_PT_SUB_TABLE_READ_FAIL<br>An attempt was made to read a sub-block of interrupts from a Pending table, where an error response was received with the data. | 0x12 | Target, bits[31:16].<br>ID, bits[15:0]. |
| Software Error (0) | 0x47, SYN_PT_TABLE_WRITE_FAIL_BYTE<br>An attempt was made to write back a subblock of interrupts from a Pending table, with an error received with the write response. | 0x12 | Target, bits[31:16].<br>ID, bits[15:0]. |
| Correctable SPI RAM errors (1) | 0x0 | 0x7 | Bit location, ID, bits[$\log_2$(SPIs)+]. |
| Uncorrectable SPI RAM errors (2) | 0x0 | 0x7 | ID, bits[$\log_2$(SPIs) − 1:0]. |
| Correctable SGI RAM errors (3) | 0x0 | 0x7 | Bit location, $\log_2$(width).<br>Address, bits[(ceiling(cores / 16) × 16) − 1:0]. |

**Table 4-51 Data field encoding (continued)**

| Record | GICT_ERR<n>STATUS.IERR (syndrome) | GICT_ERR<n>STATUS.SERR | Value and description of GICT_ERR<n>MISC0.Data (Other bits RES0)<br>Always packed from 0 (lowest = 0) |
|---|---|---|---|
| Uncorrectable SGI RAM errors (4) | 0x0 | 0x7 | Address, bits[(ceiling(cores / 16) × 16) − 1:0]. |
| Reserved (5) | - | - | - |
| Reserved (6) | - | - | - |
| Correctable PPI RAM errors (7) | 0x0 | 0x7 | PPI block, bits[18+].<br>Bit location, bits[17:12].<br>Offset, bits[11:8].<br>SGI/Int, bit[7].<br>Core, bits[6:0]. |
| Uncorrectable PPI RAM errors (8) | 0x0 | 0x7 | PPI block, bits[12+].<br>Offset, bits[11:8].<br>SGI/Int, bit[7].<br>Core, bits[6:0]. |
| Correctable LPI RAM errors (9) | 0x0 | 0x7 | Bit location, bit[15+].<br>Reserved, bit[14].<br>Pending[af], bits[13:12].<br>Reserved, bits[11:10].<br>Address, bits[9:0]. |
| Uncorrectable LPI RAM errors (10) | 0x0 | 0x7 | Pending, bits[13:12]<br>Reserved, bits[11:10]<br>Address, bits[9:0]<br>The corresponding data is stored in GICT_ERR<n>MISC1. |
| Correctable error from ITS RAM (11) | 0x0 | 0x6 | Bit location, bits[(x + 15)+].<br>Address, bits[(x + 14)+].<br>RAM, bits[x + 2:x].<br>ITS, bits[x − 1:0].<br>x = $\log_2$(ITS). |

[af] Pending bits[13:12] indicate if there were pending interrupts in the cache at the time of the corruption.

**Table 4-51 Data field encoding (continued)**

| Record | GICT_ERR<n>STATUS.IERR (syndrome) | GICT_ERR<n>STATUS.SERR | Value and description of GICT_ERR<n>MISC0.Data (Other bits RES0)<br>Always packed from 0 (lowest = 0) |
|---|---|---|---|
| Uncorrectable error from ITS RAM (12) | `0x0` | `0x6` | Address, bits[(x + 3)+].<br>RAM, bits[x + 2:x].<br>ITS, bits[x − 1:0].<br>x = $\log_2$(ITS). |
| Command or translation error in ITS (13+) | `0x0`, Architectural<br>`0x1`, NonArchitectural | `0xE` | ITS 24-bit syndrome |

### 4.8.6 Error Record Miscellaneous Register 1, GICT_ERR<n>MISC1

This register contains the data value of an uncorrectable error in the LPI RAM, or ITS software information for one of 13, or more, error records. The GIC-600 only supports a single MISC1 register, so n = 10, and therefore this register is identified as GICT_ERR10MISC1.

The GICT_ERR10MISC1 characteristics are:

**Usage constraints** If GICD_SAC.GICTNS == 0, then only Secure software can access the functions of this register.

If GICT_ERR10STATUS.MV == 1, then GICT_ERR10MISC1 ignores writes.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.8 GICT register summary* on page 4-152.

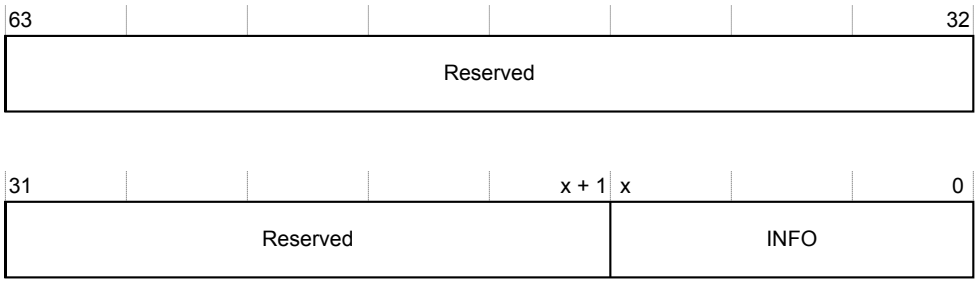


**Figure 4-39 GICT_ERR10MISC1 bit assignments**

The following table shows the bit assignments.

**Table 4-52 GICT_ERR10MISC1 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [63:x + 1] | - | Reserved, RAZ. |
| [x:0] | INFO | Value represents either data that is written to the LPI RAM when an uncorrectable error is detected, or ITS software information for one of 13, or more, error records. The value x depends on the width of the LPI RAM, which is set during configuration of the GIC-600. |

### 4.8.7 Error Group Status Register, GICT_ERRGSR

This register shows the status of the GIC-600 ARMv8.2 RAS architecture-compliant error records for correctable and uncorrectable RAM ECC errors, ITS command and translation errors, and uncorrectable software errors.

The GICT_ERRGSR characteristics are:

**Usage constraints** None.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.8 GICT register summary* on page 4-152.

The following figure shows the bit assignments.

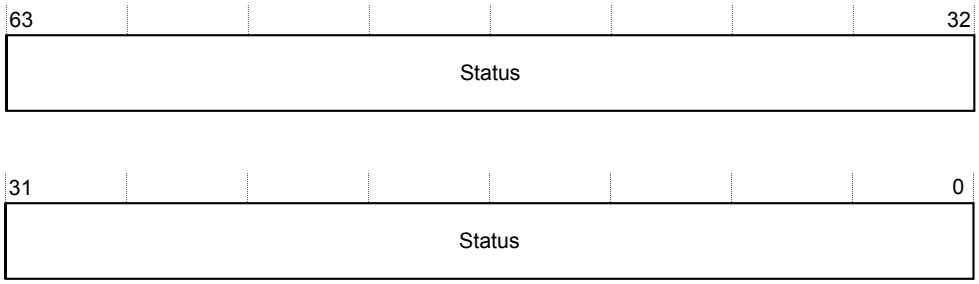


**Figure 4-40 GICT_ERRGSR bit assignments**

The following table shows the bit assignments.

**Table 4-53 GICT_ERRGSR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [n] | Status | Indicates the status of error record n, where n is 0-13+ depending on the configuration:<br>0 = The error record is not reporting any errors.<br>1 = The error record is reporting one or more errors. |

### 4.8.8 Error Interrupt Configuration Registers, GICT_ERRIRQCR<n>

GICT_ERRIRQCR0 controls the fault handling interrupts. GICT_ERRIRQCR1 controls the error recovery interrupts.

The GICT_ERRIRQCR<n> characteristics are:

**Usage constraints** If GICD_SAC.GICTNS == 0, then only Secure software can access the functions of this register.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.8 GICT register summary* on page 4-152.

The following figure shows the bit assignments.

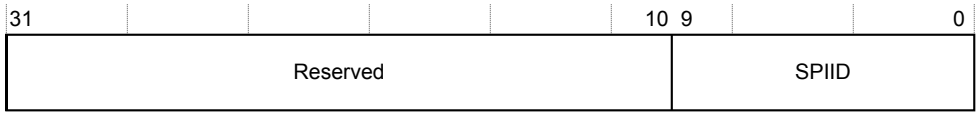


**Figure 4-41 GICT_ERRIRQCR<n> bit assignments**

The following table shows the bit assignments.

**Table 4-54 GICT_ERRIRQCR<n> bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:10] | - | Reserved, RAZ. |
| [9:0] | SPIID | SPI ID.<br>Returns 0 if an invalid entry is written.<br>In a multichip configuration, the SPIID field must only be programmed to an SPI ID that the chip owns. The relevant GICD_CHIPRn register controls the SPI ownership.<br>Arm recommends that if these registers are used, then the SPI must not be used for another device either with a wire or as a message-based interrupt. |

### 4.8.9 Error Record ID Register, GICT_ERRIDR

This register returns information about the configuration of the GIC-600 GICT such as whether an LPI or ITS is available.

The GICT_ERRIDR characteristics are:

**Usage constraints** If GICD_SAC.GICTNS == 0, then only Secure software can read this register.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.8 GICT register summary* on page 4-152.

The following table shows the bit assignments.

**Table 4-55 GICT_ERRIDR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:16] | - | Reserved, RAZ. |
| [15:0] | NUM | Identifies the device configuration:<br>10 = No LPI available.<br>12 = LPI available but no ITS.<br>14 = LPI available and 1 × ITS.<br>15 = LPI available and 2 × ITS.<br>16 = LPI available and 3 × ITS. |

### 4.8.10 Peripheral ID2 Register, GICT_PIDR2

This register returns byte[2] of the peripheral ID. The GICT_PIDR2 register is part of the set of the trace and debug peripheral identification registers.

The GICT_PIDR2 characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.8 GICT register summary* on page 4-152.

The following figure shows the bit assignments.

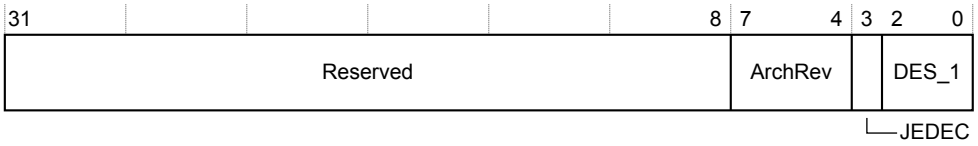


**Figure 4-42 GICT_PIDR2_bit assignments**

The following table shows the bit assignments.

**Table 4-56 GICT_PIDR2 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:8] | - | Reserved, RAZ. |
| [7:4] | ArchRev | Identifies the version of the GIC architecture with which the GIC-600 complies:<br>• `0x3` = GICv3. |
| [3] | JEDEC | Indicates that a JEDEC-assigned JEP106 identity code is used. |
| [2:0] | DES_1 | Bits[6:4] of the JEP106 identity code. Bits[3:0] of the JEP106 identity code are assigned to GICT_PIDR1. |

## 4.9 GICP register summary

The GIC-600 Performance Monitoring Unit functions are controlled through registers that are identified with the prefix GICP.

**Table 4-57 GICP register summary**

| Offset | Name | Type | Width | Reset | Description | Architecture defined? |
|---|---|---|---|---|---|---|
| 0x000 + (n × 4) | GICP_EVCNTRn | RW | 32 | UNKNOWN | *4.9.1 Event Counter Registers, GICP_EVCNTRn* on page 4-171 | k |
| 0x400 + (n × 4) | GICP_EVTYPERn [ag] | RW | 32 | UNKNOWN | *4.9.2 Event Type Configuration Registers, GICP_EVTYPERn* on page 4-172 | k |
| 0x600 + (n × 4) | GICP_SVRn [ag] | RO | 32 | UNKNOWN | *4.9.3 Shadow Value Registers, GICP_SVRn* on page 4-175 | k |
| 0xA00 + (n × 4) | GICP_FRn [ag] | RW | 32 | UNKNOWN | *4.9.4 Filter Registers, GICP_FRn* on page 4-175 | k |
| 0xC00 | GICP_CNTENSET0 | RW | 64 | 0x0 | *4.9.5 Counter Enable Set Register, GICP_CNTENSET0* on page 4-176 | k |
| 0xC20 | GICP_CNTENCLR0 | RW | 64 | 0x0 | *4.9.6 Counter Enable Clear Register 0, GICP_CNTENCLR0* on page 4-177 | k |
| 0xC40 | GICP_INTENSET0 | RW | 64 | 0x0 | *4.9.7 Interrupt Contribution Enable Set Register 0, GICP_INTENSET0* on page 4-177 | k |
| 0xC60 | GICP_INTENCLR0 | RW | 64 | 0x0 | *4.9.8 Interrupt Contribution Enable Clear Register 0, GICP_INTENCLR0* on page 4-178 | k |
| 0xC80 | GICP_OVSCLR0 | RW | 64 | 0x0 | *4.9.9 Overflow Status Clear Register 0, GICP_OVSCLR0* on page 4-179 | k |
| 0xCC0 | GICP_OVSSET0 | RW | 64 | 0x0 | *4.9.10 Overflow Status Set Register 0, GICP_OVSSET0* on page 4-179 | k |
| 0xD88 | GICP_CAPR | WO | 32 | - | *4.9.11 Counter Shadow Value Capture Register, GICP_CAPR* on page 4-180 | k |
| 0xE00 | GICP_CFGR | RO | 32 | 0x401F04 | *4.9.12 Configuration Information Register, GICP_CFGR* on page 4-180 | k |
| 0xE04 | GICP_CR | RW | 32 | 0x0 | *4.9.13 Control Register, GICP_CR* on page 4-181 | k |
| 0xE50 | GICP_IRQCR | RW | 32 | 0x0 | *4.9.14 Interrupt Configuration Register, GICP_IRQCR* on page 4-182 | k |
| 0xFB8 | GICP_PMAUTHSTATUS | RO | 32 | 0x088 | - | - |
| 0xFBC | GICP_PMDEVARCH | RO | 32 | 0x23B02A56 | - | - |
| 0xFCC | GICP_PMDEVTYPE | RO | 32 | 0x56 | - | - |
| 0xFD0 | GICP_PIDR4 | RO | 32 | 0x44 | Peripheral ID 4 Register | No |

[ag] n = 0-4.[ag]

**Table 4-57 GICP register summary (continued)**

| Offset | Name | Type | Width | Reset | Description | Architecture defined? |
|---|---|---|---|---|---|---|
| 0xFD4 | GICP_PIDR5 | RO | 32 | 0x00 | Peripheral ID 5 Register | No |
| 0xFD8 | GICP_PIDR6 | RO | 32 | 0x00 | Peripheral ID 6 Register | No |
| 0xFDC | GICP_PIDR7 | RO | 32 | 0x00 | Peripheral ID 7 Register | No |
| 0xFE0 | GICP_PIDR0 | RO | 32 | 0x96 | Peripheral ID 0 Register | No |
| 0xFE4 | GICP_PIDR1 | RO | 32 | 0xB4 | Peripheral ID 1 Register | No |
| 0xFE8 | GICP_PIDR2 | RO | 32 | 0x3B | *4.9.15 Peripheral ID2 Register, GICP_PIDR2* on page 4-182 | No |
| 0xFEC | GICP_PIDR3 | RO | 32 | 0x00 | Peripheral ID 3 Register | No |
| 0xFF0 | GICP_CIDR0 | RO | 32 | 0x0D | Component ID 0 Register | No |
| 0xFF4 | GICP_CIDR1 | RO | 32 | 0x90 | Component ID 1 Register | No |
| 0xFF8 | GICP_CIDR2 | RO | 32 | 0x05 | Component ID 2 Register | No |
| 0xFFC | GICP_CIDR3 | RO | 32 | 0xB1 | Component ID 3 Register | No |

This section contains the following subsections:


- *4.9.1 Event Counter Registers, GICP_EVCNTRn* on page 4-171.
- *4.9.2 Event Type Configuration Registers, GICP_EVTYPERn* on page 4-172.
- *4.9.3 Shadow Value Registers, GICP_SVRn* on page 4-175.
- *4.9.4 Filter Registers, GICP_FRn* on page 4-175.
- *4.9.5 Counter Enable Set Register, GICP_CNTENSET0* on page 4-176.
- *4.9.6 Counter Enable Clear Register 0, GICP_CNTENCLR0* on page 4-177.
- *4.9.7 Interrupt Contribution Enable Set Register 0, GICP_INTENSET0* on page 4-177.
- *4.9.8 Interrupt Contribution Enable Clear Register 0, GICP_INTENCLR0* on page 4-178.
- *4.9.9 Overflow Status Clear Register 0, GICP_OVSCLR0* on page 4-179.
- *4.9.10 Overflow Status Set Register 0, GICP_OVSSET0* on page 4-179.
- *4.9.11 Counter Shadow Value Capture Register, GICP_CAPR* on page 4-180.
- *4.9.12 Configuration Information Register, GICP_CFGR* on page 4-180.
- *4.9.13 Control Register, GICP_CR* on page 4-181.
- *4.9.14 Interrupt Configuration Register, GICP_IRQCR* on page 4-182.
- *4.9.15 Peripheral ID2 Register, GICP_PIDR2* on page 4-182.


### 4.9.1 Event Counter Registers, GICP_EVCNTRn

These registers contain the values of event counter n. The GIC-600 supports five counters, n = 0-4.

The GICP_EVCNTRn characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.9 GICP register summary* on page 4-170.

The following figure shows the bit assignments.

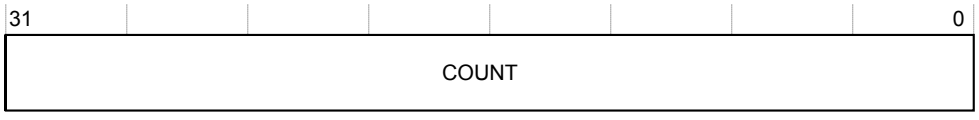


**Figure 4-43 GICP_EVCNTRn bit assignments**

The following table shows the bit assignments.

**Table 4-58 GICP_EVCNTRn bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:0] | COUNT | Counter value.<br>If the counter is enabled, the counter value increments when an event matching GICP_EVTYPERn.EVENT occurs. |

### 4.9.2 Event Type Configuration Registers, GICP_EVTYPERn

These registers configure which events are counted by the event counter n. The GIC-600 supports five counters, n = 0-4.

The GICP_EVTYPERn characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.9 GICP register summary* on page 4-170.

The following figure shows the bit assignments.

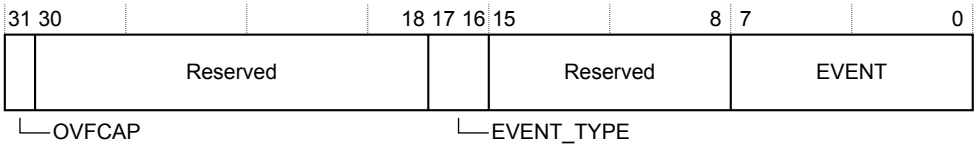


**Figure 4-44 GICP_EVTYPERn bit assignments**

The following table shows the bit assignments.

**Table 4-59 GICP_EVTYPERn bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31] | OVFCAP | When set to 1, an overflow of counter n triggers a capture if GICP_CAPR.CAPTURE is set. |
| [30:18] | - | Reserved. |
| [17:16] | EVENT_TYPE | Event tracking type:<br>0b00 = Count events.<br>0b10 = MaximumEvent.<br>0b11 = Reserved. |
| [15:8] | - | Reserved. |
| [7:0] | EVENT | Event identifier.<br>All events reset to an UNKNOWN value.<br>Registers corresponding to unimplemented counters are RES0.<br>See *Table 4-60 EVENT field encoding* on page 4-173. |

The following table shows the events that the GIC can count.

**Table 4-60 EVENT field encoding**

| Event | Description | EventID | Filter |
|---|---|---|---|
| CLK | Clock cycle. | 0x0 | None |
| CLK_NG | Clock cycle that prevents Q-Channel clock gating. | 0x1 | None |
| - | Reserved. | 0x2-0x3 | - |
| DN_MSG | Downstream message to core excluding PPIs. | 0x4 | Target |
| DN_SET | Set to core SPIs and LPIs. | 0x5 | Target/ID range |
| DN_SET1OFN | Set to core, which is a 1 of N interrupts. | 0x6 | Target/ID range |
| - | Reserved. | 0x7 | - |
| UP_MSG | Upstream message from core. | 0x8 | Target |
| UP_ACT | Upstream Activate. | 0x9 | Target/ID range |
| UP_REL | Upstream Release. | 0xA | Target/ID range |
| UP_ACTREL | Upstream Activate or Release. | 0xB | Target/ID range |
| UP_SET_COMP | A Set followed by Activate. This counts the set and then decrements on Release. | 0xC | Target/ID range |
| UP_DEACT | Upstream Deactivate. SPIs only. | 0xD | Target/ID range |
| SGI_BRD | Broadcast SGI messages. Target = source. | 0x10 | Target/ID range |
| SGI_TAR | Targeted SGI messages. Target = source. | 0x11 | Target/ID range |
| SGI_ALL | All SGI messages. Target = source. | 0x12 | Target/ID range |
| SGI_ACC | Accepted SGI. Target = source. | 0x13 | Target/ID range |
| SGI_BRD_CC_IN | Broadcast SGI message from cross-chip. | 0x14 | ID range |
| SGI_TAR_CC_IN | Targeted SGI Message from cross-chip. | 0x15 | ID range |
| SGI_TAR_CC_OUT | Targeted SGI sent cross-chip. | 0x16 | Chip/ID range |
| ITS_NLL_LPI | Incoming LPI. | 0x20 | Target/ID range/ITS |
| ITS_LL_LPI | Incoming low latency LPI. | 0x21 | Target/ID range/ITS |
| ITS_LPI | Incoming LPI (or low latency). | 0x22 | Target/ID range/ITS |
| ITS_LPI_CMD | Incoming LPI command. | 0x23 | Target/ID range/ITS |
| ITS_DID_MISS | Number of DeviceID cache misses. | 0x24 | Target/ID range/ITS |
| ITS_VID_MISS | Number of EventID cache misses. | 0x25 | Target/ID range/ITS |
| ITS_COL_MISS | Number of Collection cache misses. | 0x26 | Target/ID range/ITS |
| ITS_LAT | Latency of the ITS transaction. | 0x27 | Target/ID range/ITS |
| ITS_MPFA | Number of free slots during translation. | 0x28 | Target/ID range/ITS |
| LPI_CC_OUT | LPI sent cross-chip. | 0x29 | ID range/Chip |
| LPI_CMD_CC_OUT | LPI command sent cross-chip. | 0x2A | ID range/Chip |
| LPI_CC_IN | LPI coming in from cross-chip. | 0x2B | Target/ID range/Chip |
| LPI_CMD_CC_IN | LPI command coming in from cross-chip. | 0x2C | Target/ID range/Chip |
| LPI_OWN_STORED | LPI stored in own location. | 0x30 | - |

**Table 4-60 EVENT field encoding (continued)**

| Event | Description | EventID | Filter |
|---|---|---|---|
| LPI_OOL_STORED | LPI stored out of location. | 0x31 | - |
| LPI_HIT_EN | LPI property read cache hit enabled. Uses the filter from counter 0 only. | 0x32 | Target/ID range |
| LPI_HIT_DIS | LPI property read cache hit disabled. Uses the filter from counter 0 only. | 0x33 | Target/ID range |
| LPI_HIT | LPI property read cache hit. Uses the filter from counter 0 only. | 0x34 | Target/ID range |
| LPI_MATCH | LPI coalesced. Uses the filter from counter 0 only. | 0x35 | Target/ID range |
| LPI_FAS | Number of slots free on new LPI. | 0x36 | None |
| LPI_PROP_EN | Enabled LPI property fetch. Uses the filter from counter 0. | 0x37 | Target/ID range |
| LPI_PROP_DIS | Disabled LPI property fetch. Uses the filter from counter 0. | 0x38 | Target/ID range |
| LPI_PROP | LPI property fetch. Uses the filter from counter 0. | 0x39 | Target/ID range |
| LPI_COMP_INC_MERGE | Indicates that an LPI has completed.<br>Uses the filter from counter 0. | 0x3A | Target/ID range |
| SPI_COL_MSG | New message from SPI Collator. | 0x50 | ID range |
| SPI_ENABLED | SPI enabled (new SPI or register access if pending). | 0x51 | ID range |
| SPI_DISABLED | SPI disabled (new SPI that is disabled or register access if pending). | 0x52 | ID range |
| SPI_PENDING_SET | New SPI pending valid. | 0x53 | ID range |
| SPI_PENDING_CLR | SPI pending bit cleared. | 0x54 | ID range |
| SPI_MATCH | Collated edge-based SPI. Excludes collation in the collator. | 0x55 | ID range |
| SPI_CC_IN | SPI from remote chip. | 0x57 | ID range |
| SPI_CC_OUT | SPI sent to remote chip. | 0x58 | ID range |
| SPI_CC_DEACT | SPI deactivate message sent. | 0x5A | ID range |
| PT_IN_EN | Enabled interrupt written to Pending table. | 0x60 | Target/ID range |
| PT_IN_DIS | Disabled interrupt written to Pending table. | 0x61 | Target/ID range |
| PT_PRI | Priority of interrupt written to Pending table. | 0x62 | Target/ID range |
| PT_IN | Interrupt written to Pending table. | 0x63 | Target/ID range |
| PT_MATCH | Interrupt already set in Pending table. | 0x64 | Target/ID range |
| PT_OUT_EN | Enabled interrupt taken out of Pending table (also covered PT_MATCH when enabled). | 0x65 | Target/ID range |
| PT_OUT_DIS | Disabled interrupt taken out of Pending table (also covered PT_MATCH when disabled). | 0x66 | Target/ID range |
| PT_OUT | Disabled interrupt taken out of Pending table (also covered PT_MATCH). | 0x67 | Target/ID range |
| PT_BLOCK_SENT_CC | Pending table block that is sent as part of MOVALL. | 0x68 | None |
| SPI_CC_LATENCY | SPIs outstanding. | 0x70 | Chip |

**Table 4-60 EVENT field encoding (continued)**

| Event | Description | EventID | Filter |
|---|---|---|---|
| SPI_CC_LAT_WAIT | SPIs waiting to be sent. | 0x71 | Chip |
| LPI_CC_LATENCY | LPIs outstanding. | 0x72 | Chip |
| LPI_CC_LAT_WAIT | LPI waiting to be sent. | 0x73 | Chip |
| SGI_CC_LATENCY | SGIs outstanding. | 0x74 | Chip |
| SGI_LAT_WAIT | SGIs waiting to be sent. | 0x75 | Chip |
| ACC | Counter(n − 1) − Counter(n − 2) every cycle. Prevents clock gating. | 0x80 | None |
| OFLOW | Overflow of Counter n − 1. | 0x81 | None |

### 4.9.3 Shadow Value Registers, GICP_SVRn

These registers contain the shadow value of event counter n. The GIC-600 supports five counters, n = 0-4.

The GICP_SVRn characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.9 GICP register summary* on page 4-170.

The following figure shows the bit assignments.

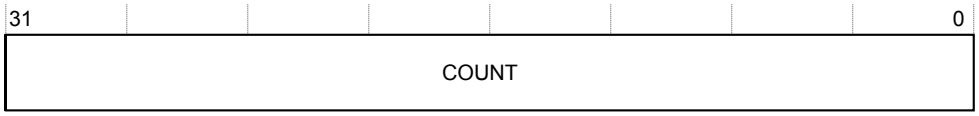


**Figure 4-45 GICP_SVRn bit assignments**

The following table shows the bit assignments.

**Table 4-61 GICP_SVRn bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:0] | COUNT | Captured counter value.<br>This field holds the captured counter values of the corresponding entry in GICP_EVCNTRn. |

### 4.9.4 Filter Registers, GICP_FRn

These registers configure the filtering of event counter n. The GIC-600 supports five counters, n = 0-4.

The GICP_FRn characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.9 GICP register summary* on page 4-170.

The following figure shows the bit assignments.

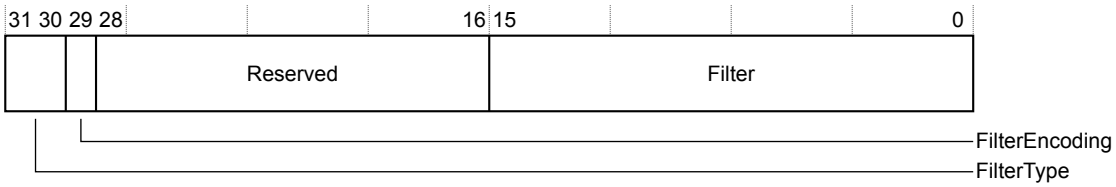


**Figure 4-46 GICP_FRn bit assignments**

The following table shows the bit assignments.

**Table 4-62 GICP_FRn bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:30] | FilterType | Filter type:<br>`0b00` = Filter on core.<br>`0b01` = Filter on INTID.<br>`0b10` = Filter on chip or ITS.<br>`0b11` = Reserved, no effect. |
| [29] | FilterEncoding | 0 = Filter on range.<br>1 = Filter on an exact match. |
| [28:16] | - | Reserved. |
| [15:0] | Filter | If the corresponding GICP_EVTYPERn.EVENT indicates an event that cannot be filtered, then the value in this register is ignored.<br>When the corresponding FilterEncoding == 1, counter `n` counts events that are only associated with an exact match of the FilterType.<br>When FilterEncoding == 0, this field is encoded so that the least-significant bit[0] indicates the uppermost of a contiguous span of least-significant FilterType bits that are ignored for the purposes of matching. |

### 4.9.5 Counter Enable Set Register, GICP_CNTENSET0

These registers contain the enables for each event counter. The GIC-600 supports five event counters.

The GICP_CNTENSET0 characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.9 GICP register summary* on page 4-170.

The following figure shows the bit assignments.

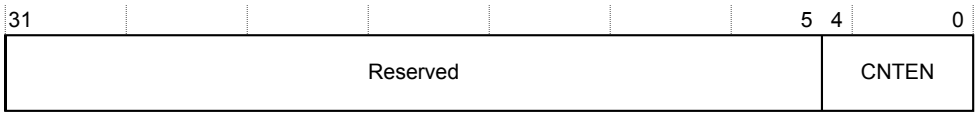


**Figure 4-47 GICP_CNTENSET0 bit assignments**

The following table shows the bit assignments.

**Table 4-63 GICP_CNTENSET0 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:5] | - | Reserved, RAZ |
| [4:0] | CNTEN | Counter enable.<br>The CNTEN[n] bit is the enable for counter n. This field resets to an UNKNOWN value.<br>Writing 1 to a bit location sets the enable for the associated counter.<br>Writing 0 to a bit location has no effect. To disable a counter, use the GICP_CNTENCLR0 register.<br>Reads return the state of the counter enables.<br>Counter n is enabled when CNTEN[n] == 1 and GICP_CR.E == 1. |

### 4.9.6 Counter Enable Clear Register 0, GICP_CNTENCLR0

This register contains the disables for each event counter. The GIC-600 supports five event counters.

The GICP_CNTENCLR0 characteristics are:

**Usage constraints** There are no usage constraints.
**Configurations** Available in all GIC-600 configurations.
**Attributes** See *4.9 GICP register summary* on page 4-170.

The following figure shows the bit assignments.

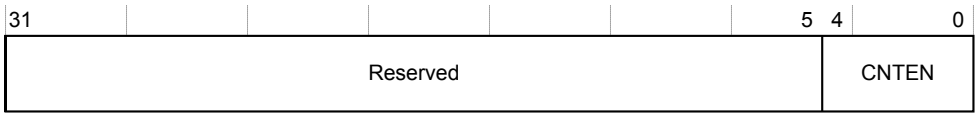


**Figure 4-48 GICP_CNTENCLR0 bit assignments**

The following table shows the bit assignments.

**Table 4-64 GICP_CNTENCLR0 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:5] | - | Reserved, RAZ |
| [4:0] | CNTEN | Counter disable. The CNTEN[n] bit is the disable for counter n. This field resets to an UNKNOWN value.<br>Writing 1 to a bit location clears the enable for the associated counter.<br>Writing 0 to a bit location has no effect. To enable a counter, use the GICP_CNTENSET0 register.<br>Reads return the state of the counter enables.<br>Counter n is disabled when CNTEN[n] == 0 or GICP_CR.E == 1. |

### 4.9.7 Interrupt Contribution Enable Set Register 0, GICP_INTENSET0

This register contains the set mechanism for the counter interrupt contribution enables. The GIC-600 supports five counters, n = 0-4.

The GICP_INTENSET0 characteristics are:

**Usage constraints** There are no usage constraints.
**Configurations** Available in all GIC-600 configurations.
**Attributes** See *4.9 GICP register summary* on page 4-170.

The following figure shows the bit assignments.

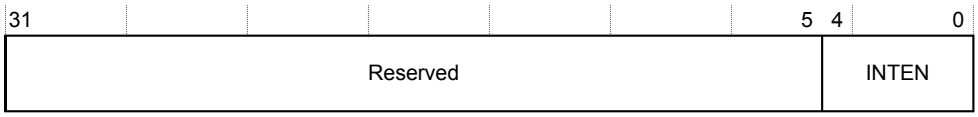


**Figure 4-49 GICP_INTENSET0 bit assignments**

The following table shows the bit assignments.

**Table 4-65 GICP_INTENSET0 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:5] | - | Reserved, RAZ |
| [4:0] | INTEN | Interrupt enable. The INTEN[n] bit is the interrupt enable for counter n. This field resets to an UNKNOWN value.<br>Writing 1 to a bit location sets the interrupt enable for the associated counter.<br>Writing 0 to a bit location has no effect. To disable a counter interrupt enable, use the GICP_INTENCLR0 register.<br>Reads return the state of the interrupt enables.<br>The interrupt enable for counter n is enabled when INTEN[n] == 1 and GICP_CR.E == 1.<br>Overflow of counter n sets GICP_OVSSET0.OVS[n] to 1 and that triggers the PMU interrupt if INTEN[n] == 1. |

### 4.9.8 Interrupt Contribution Enable Clear Register 0, GICP_INTENCLR0

This register contains the clear mechanism for the counter interrupt contribution enables. The GIC-600 supports five counters, n = 0-4.

The GICP_INTENCLR0 characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.9 GICP register summary* on page 4-170.

The following figure shows the bit assignments.

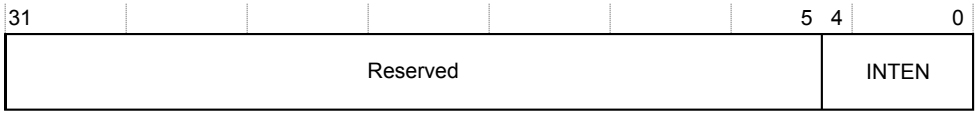


**Figure 4-50 GICP_INTENCLR0 bit assignments**

The following table shows the bit assignments.

**Table 4-66 GICP_INTENCLR0 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:5] | - | Reserved, RAZ. |
| [4:0] | INTEN | Interrupt enable. The INTEN[n] bit is the interrupt disable for counter n. This field resets to an UNKNOWN value.<br>Writing 1 to a bit location clears the interrupt enable for the associated counter.<br>Writing 0 to a bit location has no effect. To enable a counter interrupt enable, use the GICP_INTENSET0 register.<br>Reads return the state of the interrupt enables. |

### 4.9.9 Overflow Status Clear Register 0, GICP_OVSCLR0

This register provides the clear mechanism for the counter overflow status bits and provides read access to the counter overflow status bit values. The GIC-600 supports five counters, n = 0-4.

The GICP_OVSCLR0 characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.9 GICP register summary* on page 4-170.

The following figure shows the bit assignments.

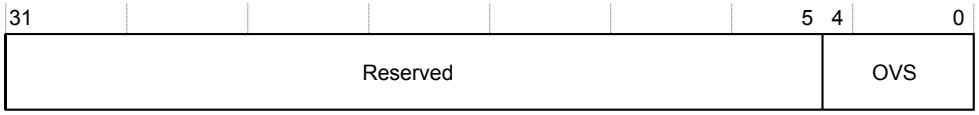


**Figure 4-51 GICP_OVSCLR0 bit assignments**

The following table shows the bit assignments.

**Table 4-67 GICP_OVSCLR0 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:5] | - | Reserved, RAZ. |
| [4:0] | OVS | Overflow status. The OVS[n] bit is the overflow clear for counter n. This field resets to zero.<br>Writing 1 to a bit location clears the overflow status for the associated counter.<br>Writing 0 to a bit location has no effect. To set a counter overflow status, use the GICP_OVSSET0 register.<br>Reads return the state of the overflow status bits.<br>Overflow of counter n, that is a transition past the maximum unsigned value of the counter that causes the value to wrap and become zero, and sets the corresponding OVS bit. In addition, this event can trigger the PMU interrupt and cause a capture of the PMU counter values, see *4.9.2 Event Type Configuration Registers, GICP_EVTYPERn* on page 4-172. |

### 4.9.10 Overflow Status Set Register 0, GICP_OVSSET0

This register provides the set mechanism for the counter overflow status bits and provides read access to the counter overflow status bit values. The GIC-600 supports five counters, n = 0-4.

The GICP_OVSSET0 characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.9 GICP register summary* on page 4-170.

The following figure shows the bit assignments.

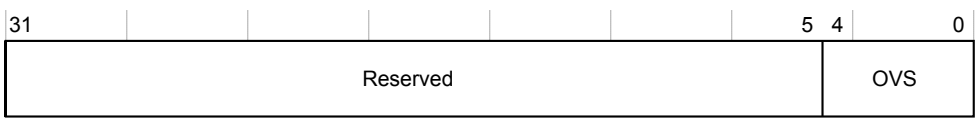


**Figure 4-52 GICP_OVSSET0 bit assignments**

The following table shows the bit assignments.

**Table 4-68 GICP_OVSSET0 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:5] | - | Reserved, RAZ. |
| [4:0] | OVS | Overflow status. The OVS[n] bit is the overflow set for counter n. This field resets to zero.<br>Writing 1 to a bit location sets the overflow status for the associated counter.<br>Writing 0 to a bit location has no effect. To clear a counter overflow status, use the GICP_OVSCLR0 register.<br>Reads return the state of the overflow status bits.<br>When the agent controlling the GIC-600 sets an OVS bit, it is similar to an OVS bit being set because of a counter overflow. However, it is IMPLEMENTATION DEFINED whether the overflow triggers the PMU interrupt or performs a capture of the PMU counter values.<br>Setting the OVS bit triggers the overflow interrupt if it is enabled. |

### 4.9.11 Counter Shadow Value Capture Register, GICP_CAPR

This register controls the counter shadow value capture mechanism.

The GICP_CAPR characteristics are:

**Usage constraints** There are no usage constraints.
**Configurations** Available in all GIC-600 configurations.
**Attributes** See *4.9 GICP register summary* on page 4-170.

The following figure shows the bit assignments.

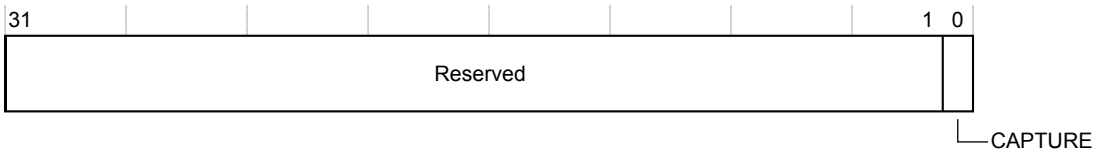


**Figure 4-53 GICP_CAPR bit assignments**

The following table shows the bit assignments.

**Table 4-69 GICP_CAPR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:1] | - | Reserved. |
| [0] | CAPTURE | When GICP_CFGR.CAPTURE == 1, a write of 1 to this bit triggers a capture of all values within the PMU into their respective shadow registers.<br>When GICP_CFGR.CAPTURE == 0, this bit is zero. |

***Related reference***
*A.6 Miscellaneous signals* on page Appx-A-195

### 4.9.12 Configuration Information Register, GICP_CFGR

This register returns information about the PMU implementation.

The GICP_CFGR characteristics are:

**Usage constraints** There are no usage constraints.
**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.9 GICP register summary* on page 4-170.

The following figure shows the bit assignments.

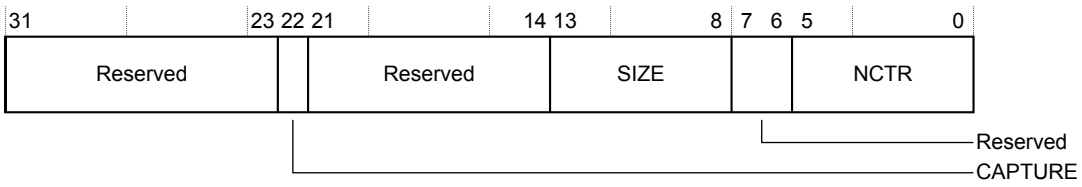


**Figure 4-54 GICP_CFGR bit assignments**

The following table shows the bit assignments.

**Table 4-70 GICP_CFGR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:23] | - | Reserved, RAZ. |
| [22] | CAPTURE | Set to 1, to indicate that the GIC supports capture. |
| [21:14] | - | Reserved, RAZ. |
| [13:8] | SIZE | Set to 31, to indicate that the GIC supports 32-bit counters. |
| [7:6] | - | Reserved, RAZ. |
| [5:0] | NCTR | Set to 4, to indicate that the GIC provides five counters. |

### 4.9.13 Control Register, GICP_CR

This register controls whether all counters are enabled or disabled.

The GICP_CR characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.9 GICP register summary* on page 4-170.

The following figure shows the bit assignments.

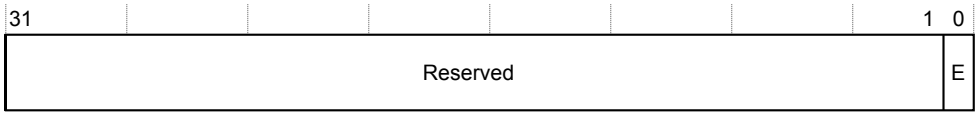


**Figure 4-55 GICP_CR bit assignments**

The following table shows the bit assignments.

**Table 4-71 GICP_CR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:1] | - | Reserved. |
| [0] | E | Global counter enable:<br>0 = No events are counted and values in the GICP_EVCNTRn registers do not change.<br>1 = The counters are enabled.<br>Resets to 0.<br>This bit takes precedence over the GICP_CNTENSET0.CNTEN bits. |

### 4.9.14 Interrupt Configuration Register, GICP_IRQCR

This register controls the counter interrupts.

The GICP_IRQCR characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.9 GICP register summary* on page 4-170.

The following figure shows the bit assignments.

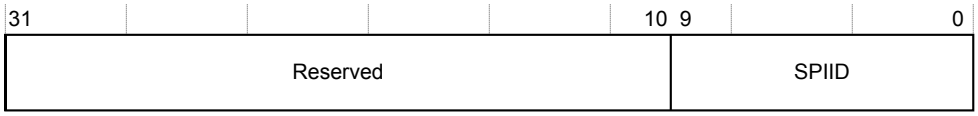


**Figure 4-56 GICP_IRQCR bit assignments**

The following table shows the bit assignments.

**Table 4-72 GICP_IRQCR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:10] | - | Reserved, RAZ. |
| [9:0] | SPIID | SPI ID.<br>Returns 0 if an invalid entry is written.<br>Creates a level-triggered interrupt if it is owned on chip. Otherwise it behaves as a normal message-based SPI.<br>In a multichip configuration, the SPIID field must only be programmed to an SPI ID that the chip owns. The relevant GICD_CHIPRn register controls the SPI ownership.<br>Arm recommends that if these registers are used, then the SPI must not be used for another device either with a wire or as a message-based interrupt. |

### 4.9.15 Peripheral ID2 Register, GICP_PIDR2

This register returns byte[2] of the peripheral ID. The GICP_PIDR2 register is part of the set of performance monitoring peripheral identification registers.

The GICP_PIDR2 characteristics are:

**Usage constraints** There are no usage constraints.

**Configurations** Available in all GIC-600 configurations.

**Attributes** See *4.9 GICP register summary* on page 4-170.

The following figure shows the bit assignments.

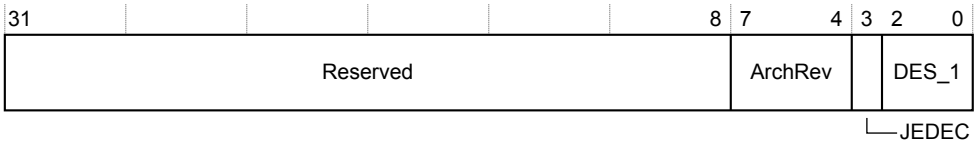


**Figure 4-57 GICP_PIDR2_bit assignments**

The following table shows the bit assignments.

**Table 4-73 GICP_PIDR2 bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:8] | - | Reserved, RAZ. |
| [7:4] | ArchRev | Identifies the version of the GIC architecture with which the GIC-600 complies:<br>• `0x3` = GICv3. |
| [3] | JEDEC | Indicates that a JEDEC-assigned JEP106 identity code is used. |
| [2:0] | DES_1 | Bits[6:4] of the JEP106 identity code. Bits[3:0] of the JEP106 identity code are assigned to GICP_PIDR1. |

# Appendix A
# **Signal descriptions**

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

It contains the following sections:


- *A.1 Common control signals* on page Appx-A-185.
- *A.2 Power control signals* on page Appx-A-187.
- *A.3 Interrupt signals* on page Appx-A-188.
- *A.4 CPU interface signals* on page Appx-A-189.
- *A.5 ACE interface signals* on page Appx-A-190.
- *A.6 Miscellaneous signals* on page Appx-A-195.
- *A.7 Interblock signals* on page Appx-A-196.
- *A.8 Interdomain signals* on page Appx-A-199.
- *A.9 Interchip signals* on page Appx-A-200.

## A.1 Common control signals

The following table shows the GIC-600 common control signal set.

**Table A-1 Common control signals**

| Signal name | Type | Source or destination | Description |
|---|---|---|---|
| **[<domain>]clk** | Input | Clock source | Clock input. |
| **[<domain>]reset_n** | Input | Reset source | Active-LOW reset. Minimum of one cycle. |
| **dbg_[<domain>]reset_n** | Input | Reset source | Active-LOW reset for the PMU and error records.<br>Only present for domain containing the Distributor. |
| Test signals | | | |
| **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. |
| MBIST controller signals | | | |
| **[<domain>_]mbistack** | Output | MBIST controller | MBIST mode ready.<br>GIC-600 acknowledges that it is ready for MBIST testing. |
| **[<domain>_]mbistreq** | Input | | MBIST mode request.<br>Request to GIC-600 to enable MBIST testing. This signal must be tied LOW during functional operation. |
| **[<domain>_]nmbistreset** | Input | | Resets MBIST logic.<br>Resets functional logic to enable MBIST operation by an active-LOW signal. This signal must be tied HIGH during functional operation. |

**Table A-1 Common control signals (continued)**

| Signal name | Type | Source or destination | Description |
|---|---|---|---|
| **[<domain>_]mbistaddr[variable:0]**[ah] | Input | MBIST controller | Logical address.<br>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. |
| **[<domain>_]mbistindata[variable:0]**[ah] | Input | | Data in.<br>Write data. Width that is based on the RAM with the largest number of data bits. |
| **[<domain>_]mbistoutdata[variable:0]**[ah] | Output | | Data out.<br>Read data. Width that is based on the RAM with the largest number of data bits. |
| **[<domain>_]mbistwriteen** | Input | | Write control (**mbistwriteen**) and read control (**mbistreaden**). No access occurs if both enables are LOW. It is illegal to activate both enables simultaneously. |
| **[<domain>_]mbistreaden** | Input | | |
| **[<domain>_]mbistarray[variable:0]**[ah] | Input | | Array selector.<br>This signal controls which RAM array is accessed. For the single RAM configuration, this port is unused.<br>This signal is not present on a block containing only one RAM. |
| **[<domain>_]mbistcfg** | Input | | MBIST ALL enable.<br>When enabled, allows simultaneous access to all RAM arrays for maximum array power consumption.<br>This signal is not present on a block containing only one RAM. |

[ah] The variable is configuration-dependent.

## A.2 Power control signals

The following table shows the GIC-600 power control signals.

**Table A-2 Power control signals**

| Signal name | Type | Source or destination | Description |
|---|---|---|---|
| **cpu_active[_<ppi_block>][_<bus>] [<num_cpus − 1:0>]** | Input | Power controller | Indicates if the core is active and not in a low-power state such as retention. This is used for lowering the priority of selection for 1 of N SPIs. There is 1 bit per core on the ICC bus. |
| **wake_request[<num_cpus − 1:0>]** | Output | Power controller | Wake request signal to power controller indicating that an interrupt is targeting this core and it must be woken. When asserted, the **wake_request** is sticky unless the Distributor is put into the gated state. |
| **qreqn_col** | Input | Low-power interface | Low-power interface to flush out the path between the SPI Collator and the Distributor to aid in power down.<br>When asserted, messages are not sent to the Distributor until low-power state is exited.<br>**Note**<br>It is only safe to stop the Collator clock if all interrupts are level sensitive, or if edge-triggered, pulse extended into the SPI Collator. |
| **qacceptn_col** | Output | | |
| **qdeny_col** | Output | | |
| **qactive_col** | Output | | |
| **qreqn_its[<its>]** | Input | Low-power interface[ai] | Required to flush out the path between the ITS and the Distributor.<br>There is one Q-Channel for each ITS.<br>All Distributor ITS Q-Channels are combined as a single set of vectored signals, **qreqn_its**[num_ITS − 1:0]. |
| **qacceptn_its[<its>]** | Output | | |
| **qdeny_its[<its>]** | Output | | |
| **qactive_its[<its>]** | Output | | |
| **[<domain_>]clkqreqn** | Input | Clock controller | Low-power interface for clock gating of everything in the domain.<br>**[<domain_>]clkqreqn** is synchronized into the GIC-600.<br>This bus must be treated asynchronously. |
| **[<domain_>]clkqacceptn** | Output | | |
| **[<domain_>]clkqdeny** | Output | | |
| **[<domain_>]clkqactive** | Output | | |
| **[<domain_>]pwrqreqn** | Input | Power controller | ADB power interface within the domain.<br>See the *Arm® CoreLink™ ADB-400 AMBA® Domain Bridge User Guide*. |
| **[<domain_>]pwrqacceptn** | Output | | |
| **[<domain_>]pwrqdeny** | Output | | |
| **[<domain_>]pwrqactive** | Output | | |
| **preq** | Input | Power controller | This P-Channel interface is only present in multichip configurations.<br>See *3.17.5 Power control and P-Channel* on page 3-100.<br>**preq** is synchronized into the GIC-600.<br>**pstate** must be stable when **preq** is asserted.<br>This bus must be treated asynchronously. |
| **pstate[4:0]** | Input | | |
| **paccept** | Output | | |
| **pdeny** | Output | | |
| **pactive** | Output | | |

[ai] These signals are not present in monolithic configurations where the Distributor and ITS share ACE-Lite ports.

## A.3 Interrupt signals

The following table shows the GIC-600 interrupt signal set.

**Table A-3 Interrupt signals**

| Signal name | Type | Source or destination | Description |
|---|---|---|---|
| **ppi<n><[_<ppi_block>] [_<bus>][_<num_cpus − 1:0>]** | Input | Interrupt source | PPI input wires for interrupt <n>. One bit per core.<br>n is 16-31 if the number of PPIs per core is 16.<br>n is 20-31 if the number of PPIs per core is 12.<br>n is 22-27, 29, 30 if number of PPIs per core is 8. |
| **ppi<n>_r_[_<ppi_block>] [_<bus>]** | Output | Interrupt source | PPI output after synchronization and edge detection. You can use it for cross-domain pulse detection. |
| **spi[variable:0]** | Input | Interrupt source | This is the number of SPI wires that are supported by the GIC.<br>—————— **Note** ——————<br>This is not the same as the number of SPIs supported because they could be message-based only or be on another chip.<br>———————————— |
| **spi_r[variable:0]** | Output | Interrupt source | SPI output after synchronization and edge detection. Can be used for cross-domain pulse detection. |

## A.4 CPU interface signals

The following table shows the GIC-600 CPU interface signal set.

**Table A-4 CPU interface signals**

| Signal name | Type | Source or destination | Description |
|---|---|---|---|
| **icctready[_<ppi_num>] [_<bus>]** | Output | Core block | GIC Stream-compliant bus for communication from the core block to the Redistributor. It is fully credited and can be sent over any free-flowing interconnect.<br>See the *Redistributor to downstream CPU interface table* in the *GIC Stream Redistributor to downstream CPU interface Appendix* of the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.<br>IDs **icctid** values of **<num_cpus − 1:0>** are used. Issuing other values is UNPREDICTABLE. |
| **icctvalid[_<ppi_num>] [_<bus>]** | Input | | |
| **icctdata[_<ppi_num>] [_<bus>][15:0]** | Input | | |
| **icctid[_<ppi_num>][_<bus>] [variable:0]**[aj] | Input | | |
| **icctlast[_<ppi_num>] [_<bus>]** | Input | | |
| **iccttwakeup[_<ppi_num>] [_<bus>]** | Input | | Registered wake signal to indicate that a message is arriving or is about to arrive on the **icc** bus. Signals **icctvalid** and **icctready** control data transfer. |
| **iritready[_<ppi_num>] [_<bus>]** | Input | Core block | GIC Stream-compliant bus for communication from the Redistributor to the core block. It is fully credited and can be sent over any free-flowing interconnect.<br>See the *Redistributor to downstream CPU interface table* in the *GIC Stream Redistributor to downstream CPU interface Appendix* of the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.<br>IDs **iritdest** values of **<num_cpus − 1:0>** are used. |
| **iritvalid[_<ppi_num>] [_<bus>]** | Output | | |
| **iritdata[_<ppi_num>] [_<bus>][15:0]** | Output | | |
| **iritdest [_<ppi_num>] [_<bus>][variable:0]**[aj] | Output | | |
| **iritlast[_<ppi_num>] [_<bus>]** | Output | | |
| **iritwakeup[_<ppi_num>] [_<bus>]** | Output | | Registered wake signal to indicate that a message is arriving or is about to arrive on the **iri** bus. Signals **iritvalid** and **iritready** control data transfer. |

[aj] The variable is configuration-dependent.

## A.5 ACE interface signals

The following table shows the GIC-600 ACE signal set.

**Table A-5 ACE interface signals**

| Signal name | Type | Description |
|---|---|---|
| Write address channel signals → Slave | | |
| There are multiple versions of this bus. Buses that have **_its[_<num>]** are dedicated ITS slave ports for GITS_TRANSLATER only. There is always one port that has no **_its** suffix that is used for all registers except GITS_TRANSLATER. This port is used for all registers in monolithic configurations. | | |
| **awuser_[its[_<num>]]_s[did_width:0]**[ak] | Input | Optional user-defined signal in the write address channel. Supported only in AXI4.<br>Indicates the *DeviceID* of writes to GITS_TRANSLATER if MSI_64 is not configured. |
| **awaddr_[its[_<num>]]_s[variable:0]**[ak] | Input | The write address gives the address of the first transfer in a write burst transaction. |
| **awid_[its[_<num>]]_s[variable:0]**[ak] | Input | This signal is the identification tag for the write address group of signals. |
| **awlen_[its[_<num>]]_s[7:0]** | Input | The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address. |
| **awsize_[its[_<num>]]_s[2:0]** | Input | This signal indicates the size of each transfer in the burst. |
| **awburst_[its[_<num>]]_s[1:0]** | Input | The burst type and the size information, determine how the address for each transfer within the burst is calculated. |
| **awprot_[its[_<num>]]_s[2:0]** | Input | This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access. |
| **awvalid_[its[_<num>]]_s** | Input | This signal indicates that the channel is signaling valid write address and control information. |
| **awready_[its[_<num>]]_s** | Output | This signal indicates that the slave is ready to accept an address and associated control signals. |
| **awcache_[its[_<num>]]_s[3:0]** | Input | This signal indicates how transactions are required to progress through a system. |
| **awdomain_[its[_<num>]]_s[1:0]** | Input | This signal indicates the shareability domain of a write transaction. |
| **awsnoop_[its[_<num>]]_s[2:0]** | Input | This signal indicates the transaction type for Shareable write transactions. |
| **awbar_[its[_<num>]]_s[1:0]** | Input | This signal indicates a write barrier transaction. |
| Write data channel signals → Slave | | |
| **wstrb_[its[_<num>]]_s[variable:0]**[ak] | Input | This signal indicates which byte lanes hold valid data. There is one write strobe bit for every eight bits of the write data bus. |
| **wdata_[its[_<num>]]_s[variable:0]**[ak] | Input | Write data. |
| **wvalid_[its[_<num>]]_s** | Input | This signal indicates that valid write data and strobes are available. |

**Table A-5 ACE interface signals (continued)**

| Signal name | Type | Description |
|---|---|---|
| **wready_[its[_<num>]]_s** | Output | This signal indicates that the slave can accept the write data. |
| **wlast_[its[_<num>]]_s** | Input | This signal indicates the last transfer in a write burst. |
| Write response channel signals → Slave | | |
| **bid_[its[_<num>]]_s[variable:0]**[ak] | Output | This signal is the ID tag of the write response. |
| **bvalid_[its[_<num>]]_s** | Output | This signal indicates that the channel is signaling a valid write response. |
| **bready_[its[_<num>]]_s** | Input | This signal indicates that the master can accept a write response. |
| **bresp_[its[_<num>]]_s[1:0]** | Output | This signal indicates the status of the write transaction. |
| Read address channel signals → Slave | | |
| **arcache_[its[_<num>]]_s[3:0]** | Input | This signal indicates how transactions are required to progress through a system. |
| **arbar_[its[_<num>]]_s[1:0]** | Input | This signal indicates a read barrier transaction. |
| **arsnoop_[its[_<num>]]_s[3:0]** | Input | This signal indicates the transaction type for Shareable read transactions. |
| **ardomain_[its[_<num>]]_s[1:0]** | Input | This signal indicates the shareability domain of a read transaction. |
| **araddr_[its[_<num>]]_s[variable:0]**[ak] | Input | The read address gives the address of the first transfer in a read burst transaction. |
| **arid_[its[_<num>]]_s[variable:0]**[ak] | Input | This signal is the identification tag for the read address group of signals. |
| **arlen_[its[_<num>]]_s[7:0]** | Input | This signal indicates the exact number of transfers in a burst. This changes between AXI3 and AXI4. |
| **arsize_[its[_<num>]]_s[2:0]** | Input | This signal indicates the size of each transfer in the burst. |
| **aruser_[its[_<num>]]_s[2:0]** | Input | This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or instruction access. |
| **arburst_[its[_<num>]]_s[1:0]** | Input | The burst type and the size information determine how the address for each transfer within the burst is calculated. |
| **arprot_[its[_<num>]]_s[2:0]** | Input | This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access. |
| **arvalid_[its[_<num>]]_s** | Input | This signal indicates that the channel is signaling valid read address and control information. |
| **arready_[its[_<num>]]_s** | Output | This signal indicates that the slave is ready to accept an address and associated control signals. |
| Read data channel signals → Slave | | |
| **rid_[its[_<num>]]_s[variable:0]**[ak] | Output | This signal is the identification tag for the read data group of signals generated by the slave. |
| **rdata_[its[_<num>]]_s[variable:0]**[ak] | Output | Read data. |

**Table A-5 ACE interface signals (continued)**

| Signal name | Type | Description |
|---|---|---|
| **rresp_[its[_<num>]]_s[1:0]** | Output | This signal indicates the status of the read transfer. |
| **rlast_[its[_<num>]]_s** | Output | This signal indicates the last transfer in a read burst. |
| **rvalid_[its[_<num>]]_s** | Output | This signal indicates that the channel is signaling the required read data. |
| **rready_[its[_<num>]]_s** | Input | This signal indicates that the master can accept the read data and response information. |
| Write address channel signals → Master. Only present if LPI support is configured. | | |
| Buses containing **_its[_<num>]** are used by the specific ITS for read/writes to the private tables and Command queue. Buses without an **_its** suffix are used for accesses to the LPI Pending and Property tables. This port performs all accesses in monolithic configurations. | | |
| **awaddr_[its[_<num>]]_m[variable:0]**[ak] | Output | The write address gives the address of the first transfer in a write burst transaction. |
| **awid_[its[_<num>]]_m[variable:0]**[ak] | Output | This signal is the identification tag for the write address group of signals. |
| **awlen_[its[_<num>]]_m[7:0]** | Output | The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address. |
| **awsize_[its[_<num>]]_m[2:0]** | Output | This signal indicates the size of each transfer in the burst. |
| **awburst_[its[_<num>]]_m[1:0]** | Output | The burst type and size information determine how the address for each transfer within the burst is calculated. |
| **awprot_[its[_<num>]]_m[2:0]** | Output | This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access. |
| **awvalid_[its[_<num>]]_m** | Output | This signal indicates that the channel is signaling valid write address and control information. |
| **awready_[its[_<num>]]_m** | Input | This signal indicates that the channel is signaling valid write address and control information. |
| **awcache_[its[_<num>]]_m[3:0]** | Output | This signal indicates how transactions are required to progress through a system. |
| **awdomain_[its[_<num>]]_m[1:0]** | Output | This signal indicates the shareability domain of a write transaction. |
| **awsnoop_[its[_<num>]]_m[2:0]** | Output | This signal indicates the transaction type for Shareable write transactions. |
| **awbar_[its[_<num>]]_m[1:0]** | Output | This signal indicates a write barrier transaction. |
| **awuser_m[variable:0]**[ak] | Output | Optional user-defined signal in the write address channel. |
| Write data channel signals → Master. Only present if LPI support is configured. | | |
| **wstrb_[its[_<num>]]_m[variable:0]**[ak] | Output | This signal indicates which byte lanes hold valid data. There is one write strobe bit for every eight bits of the write data bus. |
| **wdata_[its[_<num>]]_m[variable:0]**[ak] | Output | Write data. |
| **wvalid_[its[_<num>]]_m** | Output | This signal indicates that valid write data and strobes are available. |

**Table A-5 ACE interface signals (continued)**

| Signal name | Type | Description |
|---|---|---|
| **wready_[its[_<num>]]_m** | Input | This signal indicates that the slave can accept the write data. |
| **wlast_[its[_<num>]]_m** | Output | This signal indicates the last transfer in a write burst. |
| Write response channel signals → Master. Only present if LPI support is configured. | | |
| **bid_[its[_<num>]]_m[variable:0]**[ak] | Input | This signal is the ID tag of the write response. |
| **bvalid_[its[_<num>]]_m** | Input | This signal indicates that valid write data and strobes are available. |
| **bready_[its[_<num>]]_m** | Output | This signal indicates that the channel is signaling a valid write response. |
| **bresp_[its[_<num>]]_m[1:0]** | Input | This signal indicates the status of the write transaction. |
| Read address channel signals → Master. Only present if LPI support is configured. | | |
| **araddr_[its[_<num>]]_m[variable:0]**[ak] | Output | The read address gives the address of the first transfer in a read burst transaction. |
| **arid_[its[_<num>]]_m[variable:0]**[ak] | Output | This signal is the identification tag for the read address group of signals. |
| **arlen_[its[_<num>]]_m[7:0]** | Output | This signal indicates the exact number of transfers in a burst. This changes between AXI3 and AXI4. |
| **arsize_[its[_<num>]]_m[2:0]** | Output | This signal indicates the size of each transfer in the burst. |
| **arburst_[its[_<num>]]_m[1:0]** | Input | The burst type and the size information determine how the address for each transfer within the burst is calculated. |
| **arprot_[its[_<num>]]_m[2:0]** | Output | This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access. |
| **arvalid_[its[_<num>]]_m** | Output | The signal indicates that the channel is signaling valid read address and control information. |
| **arready_[its[_<num>]]_m** | Input | This signal indicates that the slave is ready to accept an address and associated control signals. |
| **arcache_[its[_<num>]]_m[3:0]** | Output | This signal indicates how transactions are required to progress through a system. |
| **ardomain_[its[_<num>]]_m[1:0]** | Output | This signal indicates the shareability domain of a read transaction. |
| **arsnoop_[its[_<num>]]_m[3:0]** | Output | This signal indicates the transaction type for Shareable read transactions. |
| **arbar_[its[_<num>]]_m[1:0]** | Output | This signal indicates a read barrier transaction. |
| **aruser_[its[_<num>]]_m[variable:0]**[ak] | Output | Optional user-defined signal in the read address channel. Supported only in AXI4. |
| Read data channel signals → Master. Only present if LPI support is configured. | | |
| **rid_[its[_<num>]]_m[variable:0]**[ak] | Input | This signal is the identification tag for the read data group of signals generated by the slave. |
| **rada_[its[_<num>]]_m[variable:0]**[ak] | Input | Read data. |
| **rresp_[its[_<num>]]_m[1:0]** | Input | This signal indicates the status of the read transfer. |

**Table A-5 ACE interface signals (continued)**

| Signal name | Type | Description |
|---|---|---|
| **rlast_[its[_<num>]]_m** | Input | This signal indicates the last transfer in a read burst. |
| **rvalid_[its[_<num>]]_m** | Input | This signal indicates that the channel is signaling the required read data. |
| **rready_[its[_<num>]]_m** | Output | This signal indicates that the master can accept the read data and response information. |

[ak] The variable is configuration-dependent.

## A.6 Miscellaneous signals

The following table shows the GIC-600 miscellaneous signals.

**Table A-6 Miscellaneous signals**

| Signal name | Type | Source or destination | Description |
|---|---|---|---|
| **ppi_id[15:0]** | Input | Interrupt source | RedistributorID number that is used for system identification only. Software can read the GICR_CFGID0 register to access the value of this signal. |
| **its_id[7:0]** | Input | ITS block | ID number that is used for system identification only. Software can read the GITS_CFGID register to access the value of this signal. |
| **sample_req** | Input | Debug controller | Request to sample PMU. Equivalent to writing to the GICP_CAPR register. |
| **sample_ack** | Output | Debug controller | **sample_req** accept. |
| **fault_int** | Output | System control processor | Fault handling interrupt. |
| **err_int** | Output | System control processor | Error handling interrupt. |
| **pmu_int** | Output | Debug controller | PMU overflow length. |
| **gict_allow_ns** | Input | Tie-off | Allow Non-secure access to error record registers. |
| **gicp_allow_ns** | Input | Tie-off | Allow Non-secure access to PMU registers. |

***Related reference***

*4.5.4 Configuration ID0 Register, GICR_CFGID0* on page 4-138
*4.6.6 Configuration ID Register, GITS_CFGID* on page 4-148
*4.9.11 Counter Shadow Value Capture Register, GICP_CAPR* on page 4-180

## A.7 Interblock signals

The following table shows the GIC-600 interblock signals.

**Table A-7 Interblock signals**

| Signal name | Forward or reverse | Source or destination | Description |
|---|---|---|---|
| **icdptready** | Reverse | Redistributor → Distributor | AXI4-Stream compliant bus for communication between the Distributor and a Redistributor. It is fully credited and must never backpressure. It can be sent over any free-flowing interconnect. |
| **icdptvalid** | Forward | Distributor → Redistributor | |
| **icdptdata[variable:0]**[al] | Forward | | |
| **icdptlast** | Forward | | |
| **icdptwakeup** | Forward | | Registered wake signal to indicate that a message is arriving or is about to arrive on the **icdp** bus. Signals **icdptvalid** and **icdptready** control data transfer. |
| **icdptdest** | Forward | | Specifies the destination Redistributor block.<br>This signal is only present on the Distributor.<br>See *AXI4-Stream interfaces* in *Functional integration guidelines* of the *Arm® CoreLink™ GIC-600 Generic Interrupt Controller Configuration and Integration Manual* for more information. |
| **icdptkeep** | Forward | | Indicates the data bytes that must be transferred.<br>This signal is only present on the Distributor. |
| **icpdtready** | Reverse | | AXI4-Stream compliant bus for communication between the Redistributor and the Distributor. It is fully credited and can be sent over any free-flowing interconnect. |
| **icpdtvalid** | Forward | Redistributor → Distributor | |
| **icpdtdata[variable:0]**[al] | Forward | | |
| **icpdtlast** | Forward | | |
| **icpdtwakeup** | Forward | | Registered wake signal to indicate that a message is arriving or is about to arrive on the **icpd** bus. Signals **icpdtvalid** and **icpdtready** control data transfer. |
| **icpdtid** | Forward | | Specifies the source Redistributor block.<br>This signal is only present on the Distributor.<br>See *AXI4-Stream interfaces* in *Functional integration guidelines* of the *Arm® CoreLink™ GIC-600 Generic Interrupt Controller Configuration and Integration Manual* for more information. |
| **icpdtkeep** | Forward | | Indicates the data bytes that must be transferred.<br>This signal is only present on the Redistributor. |

[al] The variable is configuration-dependent.

**Table A-7 Interblock signals (continued)**

| Signal name | Forward or reverse | Source or destination | Description |
|---|---|---|---|
| **icditready** | Reverse | ITS → Distributor | AXI4-Stream compliant bus for communication from the Distributor to the ITS. It is fully credited and can be sent over any free-flowing interconnect. |
| **icditvalid** | Forward | Distributor → ITS | |
| **icditdata[variable:0]**[al] | Forward | | |
| **icditlast** | Forward | | |
| **icditwakeup** | Forward | | Indicates that a message is arriving or is about to arrive on the **icdi** bus. Signals **icditvalid** and **icditready** control data transfer. |
| **icditdest** | Forward | | Specifies the destination ITS block.<br>This signal is only present on the Distributor.<br>See *AXI4-Stream interfaces* in *Functional integration guidelines* of the *Arm® CoreLink™ GIC-600 Generic Interrupt Controller Configuration and Integration Manual* for more information. |
| **icditkeep** | Forward | | Indicates the data bytes that must be transferred.<br>This signal is only present on the Distributor. |
| **icidtready** | Reverse | | AXI4-Stream compliant bus for communication from the ITS to the Distributor. It is fully credited and can be sent over any free-flowing interconnect. |
| **icidtvalid** | Forward | ITS → Distributor | |
| **icidtdata[variable:0]**[al] | Forward | | |
| **icidtkeep[variable:0]**[al] | Forward | | |
| **icidtlast** | Forward | | |
| **icidtdid** | Forward | | Specifies the source ITS.<br>This signal is only present on the Distributor.<br>See *AXI4-Stream interfaces* in *Functional integration guidelines* of the *Arm® CoreLink™ GIC-600 Generic Interrupt Controller Configuration and Integration Manual* for more information. |
| **icidtkeep** | Forward | | Indicates the data bytes that must be transferred.<br>This signal is only present on the ITS. |
| **icidtwakeup** | Forward | Registered wake signal | Indicates that a message is arriving or is about to arrive on the **icid** bus. Signals **icidtvalid** and **icidtready** control data transfer. |
| **icdwtready** | Reverse | Wake Request → Distributor | AXI4-Stream compliant bus for communication from the Distributor to the Wake Request block.<br>It is fully credited and can be sent over any free-flowing interconnect.<br>This bus is not exposed when the top level is stitched. |
| **icdwtvalid** | Forward | Distributor → Wake Request | |
| **icdwtdata[15:0]** | Forward | | |
| **icdwtwakeup** | Forward | | Registered wake signal to indicate that a message is arriving or is about to arrive on the **icdw** bus. Signals **icdwtvalid** and **icdwtready** control data transfer.<br>This signal is not exposed when the top level is stitched. |

**Table A-7 Interblock signals (continued)**

| Signal name | Forward or reverse | Source or destination | Description |
|---|---|---|---|
| **icdctready** | Reverse | SPI Collator → Distributor | AXI4-Stream compliant bus for communication between the Distributor and the SPI Collator. It is fully credited and must never backpressure. It can be sent over any free-flowing interconnect. |
| **icdctvalid** | Forward | Distributor → SPI Collator | |
| **icdctdata[15:0]** | Forward | | |
| **icdctlast** | Forward | | |
| **icdctwakeup** | Forward | | Registered wake signal to indicate that a message is arriving or is about to arrive on the **icdc** bus. Signals **icdctvalid** and **icdctready** control data transfer. |
| **iccdtdest** | Forward | | Indicates that the collator number is always 0. |
| **iccdtready** | Reverse | | AXI4-Stream compliant bus for communication between the SPI Collator and the Distributor. It is fully credited and must never backpressure. It can be sent over any free-flowing interconnect. |
| **iccdtvalid** | Forward | SPI Collator → Distributor | |
| **iccdtdata[15:0]** | Forward | | |
| **iccdtlast** | Forward | | |
| **iccdtwakeup** | Forward | | Registered wake signal to indicate that a message is arriving or is about to arrive on the **iccd** bus. Signals **iccdtvalid** and **iccdtready** control data transfer. |
| **iccdtid** | Forward | | Indicates that the collator number must be tied to 0.<br>This signal is only present on the Distributor. |

## A.8 Interdomain signals

Interdomain signals are routed between domains.

The following table shows the interdomain signals.

**Table A-8 Interdomain signals**

| Signal name |
|---|
| **wakeup_sm_*** |
| **wakeup_ms_*** |
| ***_async_*** |

If you instantiate domain levels, you must ensure that matching input and output pairs of interdomain signals connect together directly, and are not separated by synchronizers.

## A.9 Interchip signals

The following table shows the GIC-600 interchip signals.

**Table A-9 Interchip signals**

| Signal name | Forward or reverse | Source or destination | Description |
|---|---|---|---|
| **icdrtready** | Reverse | Remote chip → Distributor | AXI4-Stream compliant bus for communication between the Distributor and a remote chip. It is fully credited and must never backpressure. It can be sent over any free-flowing interconnect. |
| **icdrtvalid** | Forward | Distributor → Remote chip | |
| **icdrtdata[63:0]** | Forward | | |
| **icdrtlast** | Forward | | |
| **icdrtwakeup** | Forward | | Registered wake signal to indicate that a message is arriving or is about to arrive on the **icdr** bus. Signals **icdrtvalid** and **icdrtready** control data transfer. |
| **icdrtdest[variable:0]**[a m] | Forward | | Specifies the destination remote chip.<br>This signal is only present on the Distributor.<br>See *AXI4-Stream interfaces* in *Functional integration guidelines* of the *Arm® CoreLink™ GIC-600 Generic Interrupt Controller Configuration and Integration Manual* for more information. |
| **icdrtkeep** | Forward | | Indicates the data bytes that must be transferred.<br>This signal is only present on the Distributor. |
| **icdrtready** | Reverse | | AXI4-Stream compliant bus for communication between the remote chip and the Distributor. It is fully credited and can be sent over any free-flowing interconnect. |
| **icdrtvalid** | Forward | Remote chip → Distributor | |
| **icdrtdata[63:0]** | Forward | | |
| **icdrtlast** | Forward | | |
| **icdrtwakeup** | Forward | | Registered wake signal to indicate that a message is arriving or is about to arrive on the **icrd** bus. Signals **icdrtvalid** and **icdrtready** control data transfer. |

[am] The variable is configuration-dependent.

# Appendix B
# **Implementation-defined features**

Read this for a description of the features that are IMPLEMENTATION-DEFINED.

It contains the following section:


- *B.1 Implementation-defined features reference* on page Appx-B-202.

## B.1 Implementation-defined features reference

The GIC-600 implements features that are defined in the GICv3 Architecture. Many of these features also have options in the GICv3 Architecture, which determine behavior that is specific to the GIC-600. These features and options are configurable at build time.

The following table summarizes features in the *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0* that are used by the GIC-600, and which have options that are IMPLEMENTATION-DEFINED. The table also gives references to sections within this manual that provide information about IMPLEMENTATION-DEFINED behavior that is specific to the GIC-600.

**Table B-1 Declared implementation-defined features**

| GICv3 Architecture feature | Architectural specification reference | | Description |
|---|---|---|---|
| | Chapter | Section | |
| 1 of N model | Introduction | Models for handling interrupts | See *3.5 1 of N SPI interrupt selection* on page 3-56. |
| Direct LPI support | GIC partitioning | The GIC logical components | Direct LPI support is by configuration if there are no ITS blocks in the system. |
| ITS to Redistributor communications | Locality-specific peripheral interrupts and the ITS | LPIs | This is done over a fully credited AXI4-Stream. |
| INTIDs | Distribution and routing of interrupts | INTIDs | 16-bit width when supporting LPIs, otherwise the width is set to support the number of SPIs and SGIs. |
| All error cases | - | Pseudocode throughout the document | All errors are reported through error records, see *3.16 Reliability, Accessibility, and Serviceability* on page 3-75. |
| Message-based SPIs | Physical interrupt handling and prioritization | Shared peripheral interrupts | Pending bits for level sensitive SPIs that are set by writes to GICD_SETSPI_* or GICA_SETSPI_* are not affected by writes to GICD_ICPENDRn. Writes to GICD_CLRSPI_* or GICA_CLRSPI_* have no effect on pending bits set by GICD_ISPENDRn. |
| Interrupt grouping | Physical interrupt handling and prioritization | Interrupt grouping | All implemented SPIs, SGIs, and PPIs have programmable groups. |
| Interrupt enables | Physical interrupt handling and prioritization | Enabling individual interrupts | All SGIs have a programmable enable. |
| Interrupt prioritization | Physical interrupt handling and prioritization | Interaction of group and individual interrupt enables | Interrupts that are disabled through the GICC_CTLR register or the ICC_CTLR_* registers are not considered in the selection of the highest pending interrupt and do not block fully enabled interrupts of a lower priority. |
| | | Interrupt prioritization | GIC-600 supports 32 priority levels, 16 for LPIs that are always Non-secure. |
| Effects of disabling interrupts | Physical interrupt handling and prioritization | Effect of disabling interrupts | Interrupts are set pending irrespective of the GICD_CTLR.EnableGrp* settings. |

**Table B-1 Declared implementation-defined features (continued)**

| GICv3 Architecture feature | Architectural specification reference | | Description |
|---|---|---|---|
| | Chapter | Section | |
| Changing priority | Physical interrupt handling and prioritization | Interrupt prioritization.<br>Changing the priority of enabled PPIs, SGIs, and SPIs. | Reprogramming a IPRIORITYRn register does not change the priority of an active interrupt but causes a pending and not active interrupt to be recalled from the CPU interface so that the new value can be applied. |
| Direct LPI registers | Locality-specific peripheral interrupts and the ITS | LPIs | The GICR_SETLPIR, GICR_CLRLPIR, GICR_INVLPIR, GICR_INVALLR, and GICR_SYNCR are supported in configurations that support LPIs but have no ITS anywhere in the system. If there is an ITS, these registers, and their locations, are RAZ/WI. |
| LPI caching | Locality-specific peripheral interrupts and the ITS | LPIs | See *3.11 LPI caching* on page 3-68 and *3.10 Interrupt translation service (ITS)* on page 3-65. |
| LPI configuration tables | Locality-specific peripheral interrupts and the ITS | LPI configuration tables | The GIC-600 has one GICR_PROPBASER register for all cores on a chip and therefore points at a single table. Each chip in a multichip configuration can point to a copy of the table in local memory. See *CommonLPIAff* in *Table 4-23 GICR_TYPER bit assignments* on page 4-129 for more information.<br>When interrupts are sent between chips, they keep the properties associated with them until the next invalidate. All property fetches are always from the offset specified in the GICR_PROPBASER of the issuing chip. |
| LPI Pending tables | Locality-specific peripheral interrupts and the ITS | LPI Pending tables | Refer to the GICv3 Architecture description. |

# Appendix C
# **Revisions**

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

It contains the following section:


- *C.1 Revisions* on page Appx-C-205.

## C.1 Revisions

This appendix describes changes between released issues of this book.

**Table C-1 Issue 0000-00**

| Change | Location | Affects |
|---|---|---|
| First release | - | - |

**Table C-2 Differences between issue 0000-00 and issue 0000-01**

| Change | Location | Affects |
|---|---|---|
| Added note | *1.1 About the GIC-600* on page 1-12 | r0p0 |
| Added note | *Top level* on page 1-14 | r0p0 |
| Updated section | *1.4 Features* on page 1-19 | r0p0 |
| Updated section | *2.1.1 Distributor AXI4-Stream interfaces* on page 2-25 | r0p0 |
| Updated section | *2.1.2 Distributor ACE-Lite slave interface* on page 2-25 | r0p0 |
| Updated section | *2.1.3 Distributor ACE-Lite master interface* on page 2-26 | r0p0 |
| Updated section | *2.1.4 Distributor Q-Channels* on page 2-27 | r0p0 |
| Updated table | *Table 2-6 Configurable options for the Distributor* on page 2-28 | r0p0 |
| Updated section | *2.2.2 Redistributor GIC Stream protocol interface* on page 2-31 | r0p0 |
| Updated section | *2.2.3 Redistributor Q-Channel* on page 2-31 | r0p0 |
| Updated table | *Table 2-8 Redistributor miscellaneous input signals* on page 2-32 | r0p0 |
| Updated section | *2.3.1 ITS ACE-Lite slave interface* on page 2-35 | r0p0 |
| Updated section | *2.3.3 ITS AXI4-Stream interface* on page 2-37 | r0p0 |
| Updated table | *Table 2-11 ITS miscellaneous signals* on page 2-38 | r0p0 |
| Updated table | *Table 2-12 Configurable options for the ITS* on page 2-38 | r0p0 |
| Updated section | *2.4.1 MSI-64 ACE-Lite interfaces* on page 2-39 | r0p0 |
| Updated table | *Table 2-14 MSI-64 miscellaneous signals* on page 2-40 | r0p0 |
| Updated table | *Table 2-15 Configurable options for the MSI-64 Encapsulator* on page 2-40 | r0p0 |
| Updated section | *2.5.2 SPI Collator wires* on page 2-42 | r0p0 |
| Updated section | *2.6.2 Wake Request miscellaneous signals* on page 2-44 | r0p0 |
| Updated section | *2.6.3 Wake Request configuration* on page 2-44 | r0p0 |
| Updated section | *2.7.1 Interconnect configuration* on page 2-45 | r0p0 |
| Changed note | *3.1.4 LPIs* on page 3-50 | r0p0 |
| Updated section | *3.1.5 Choosing between LPIs and SPIs* on page 3-50 | r0p0 |
| Updated section | *3.3 Physical interrupt signals (PPIs and SPIs)* on page 3-53 | r0p0 |
| Updated section | *3.4 Affinity routing and assignment* on page 3-54 | r0p0 |
| Added section | *3.5 1 of N SPI interrupt selection* on page 3-56 | r0p0 |
| Updated section | *3.9 Backwards compatibility* on page 3-64 | r0p0 |

**Table C-2 Differences between issue 0000-00 and issue 0000-01 (continued)**

| Change | Location | Affects |
|---|---|---|
| Updated section | *3.11 LPI caching* on page 3-68 | r0p0 |
| Updated section | *3.12 Memory access and attributes* on page 3-69 | r0p0 |
| Updated section | *3.13 MSI-64* on page 3-71 | r0p0 |
| Updated section | *3.14 RAM* on page 3-72 | r0p0 |
| Updated section | *3.15 Performance Monitoring Unit* on page 3-73 | r0p0 |
| Updated table | *Table 3-6 ECC error reporting* on page 3-76 | r0p0 |
| Updated section | *1.1 About the GIC-600* on page 1-12 | r0p0 |
| Added section | *4.1.1 Register map pages* on page 4-104 | r0p0 |
| Updated section | *4.1.3 GIC-600 register access and banking* on page 4-105 | r0p0 |
| Updated table | *Table 4-2 Distributor registers (GICD/GICDA) summary* on page 4-106 | r0p0 |
| Updated table | *Table 4-4 GICD_TYPER bit assignments* on page 4-110 | r0p0 |
| Updated table | *Table 4-6 GICD_FCTLR bit assignments* on page 4-113 | r0p0 |
| Updated table | *Table 4-8 GICD_SAC bit assignments* on page 4-115 | r0p0 |
| Updated table | *Table 4-20 Distributor registers (GICA) for message-based SPIs summary* on page 4-125 | r0p0 |
| Updated table | *Table 4-21 Redistributor registers for control and physical LPIs summary* on page 4-126 | r0p0 |
| Updated table | *Table 4-22 GICR_IIDR bit assignments* on page 4-128 | r0p0 |
| Updated table | *Table 4-23 GICR_TYPER bit assignments* on page 4-129 | r0p0 |
| Updated table | *Table 4-25 GICR_FCTLR bit assignments* on page 4-131 | r0p0 |
| Updated table | *Table 4-26 GICR_PWRR bit assignments* on page 4-132 | r0p0 |
| Updated table | *Table 4-30 GICR_MISCSTATUSR bit assignments* on page 4-137 | r0p0 |
| Updated table | *Table 4-34 GICR_CFGID1 bit assignments* on page 4-140 | r0p0 |
| Updated table | *Table 4-35 ITS control register summary* on page 4-141 | r0p0 |
| Updated table | *Table 4-39 GITS_OPR bit assignments* on page 4-147 | r0p0 |
| Updated table | *Table 4-44 GICT register summary* on page 4-152 | r0p0 |
| Updated table | *Table 4-47 GICT_ERR<n>CTLR bit assignments* on page 4-156 | r0p0 |
| Updated table | *Table 4-48 GICT_ERR<n>STATUS bit assignments* on page 4-157 | r0p0 |
| Updated table | *Table 4-50 GICT_ERR<n>MISC0 bit assignments* on page 4-159 | r0p0 |
| Updated table | *Table 4-51 Data field encoding* on page 4-160 | r0p0 |
| Added section | *4.8.10 Peripheral ID2 Register, GICT_PIDR2* on page 4-168 | r0p0 |
| Updated table | *Table 4-57 GICP register summary* on page 4-170 | r0p0 |
| Updated table | *Table 4-59 GICP_EVTYPERn bit assignments* on page 4-172 | r0p0 |
| Updated table | *Table 4-60 EVENT field encoding* on page 4-173 | r0p0 |
| Updated table | *Table 4-63 GICP_CNTENSET0 bit assignments* on page 4-177 | r0p0 |
| Updated table | *Table 4-64 GICP_CNTENCLR0 bit assignments* on page 4-177 | r0p0 |
| Updated table | *Table 4-65 GICP_INTENSET0 bit assignments* on page 4-178 | r0p0 |

**Table C-2 Differences between issue 0000-00 and issue 0000-01 (continued)**

| Change | Location | Affects |
|---|---|---|
| Added section | *4.9.15 Peripheral ID2 Register, GICP_PIDR2* on page 4-182 | r0p0 |
| Updated section | *A.2 Power control signals* on page Appx-A-187 | r0p0 |
| Updated section | *A.3 Interrupt signals* on page Appx-A-188 | r0p0 |
| Updated section | *A.4 CPU interface signals* on page Appx-A-189 | r0p0 |
| Updated section | *A.7 Interblock signals* on page Appx-A-196 | r0p0 |
| Updated section | *B.1 Implementation-defined features reference* on page Appx-B-202 | r0p0 |

**Table C-3 Differences between issue 0000-01 and issue 0002-00**

| Change | Location | Affects |
|---|---|---|
| Updated fourth paragraph | *1.1 About the GIC-600* on page 1-12 | All releases |
| Updated section | *1.2 Components* on page 1-13 | All releases |
| Updated section | *1.3 Compliance* on page 1-18 | All releases |
| Updated section | *1.4 Features* on page 1-19 | All releases |
| Updated Figure | *Figure 2-1 GIC-600 Distributor* on page 2-24 | All releases |
| Updated section | *2.1.1 Distributor AXI4-Stream interfaces* on page 2-25 | All releases |
| Updated section | *2.1.2 Distributor ACE-Lite slave interface* on page 2-25 | All releases |
| Updated section | *2.1.3 Distributor ACE-Lite master interface* on page 2-26 | All releases |
| Updated section | *2.1.4 Distributor Q-Channels* on page 2-27 | All releases |
| Updated table | *2.1.6 Distributor miscellaneous signals* on page 2-28 | All releases |
| Updated table | *Table 2-6 Configurable options for the Distributor* on page 2-28 | All releases |
| Updated section | *2.2 Redistributor* on page 2-30 | All releases |
| Updated section | *2.2.2 Redistributor GIC Stream protocol interface* on page 2-31 | All releases |
| Added table | *Table 2-7 GIC Stream protocol interface signals* on page 2-31 | All releases |
| Updated section | *2.2.4 Redistributor PPI signals* on page 2-31 | All releases |
| Updated section | *2.2.5 Redistributor miscellaneous input signals* on page 2-32 | All releases |
| Updated section | *2.3 ITS* on page 2-34 | All releases |
| Updated section | *2.3.1 ITS ACE-Lite slave interface* on page 2-35 | All releases |
| Updated section | *2.3.4 ITS Q-Channel* on page 2-37 | All releases |
| Updated table | *Table 2-11 ITS miscellaneous signals* on page 2-38 | All releases |
| Updated section | *2.4 MSI-64 Encapsulator* on page 2-39 | All releases |
| Updated section | *2.4.2 MSI-64 miscellaneous signals* on page 2-40 | All releases |
| Updated section | *2.4.3 MSI-64 Encapsulator configuration* on page 2-40 | All releases |
| Updated section | *2.5 SPI Collator* on page 2-42 | All releases |
| Updated section | *2.5.1 SPI Collator AXI4-Stream interface* on page 2-42 | All releases |
| Updated section | *2.5.2 SPI Collator wires* on page 2-42 | All releases |

**Table C-3 Differences between issue 0000-01 and issue 0002-00 (continued)**

| Change | Location | Affects |
|---|---|---|
| Updated section | *2.5.3 SPI Collator power Q-Channel* on page 2-42 | All releases |
| Updated section | *2.6 Wake Request* on page 2-44 | All releases |
| Updated section | *2.6.3 Wake Request configuration* on page 2-44 | All releases |
| Updated figure | *Figure 2-8 GIC top-level structure options* on page 2-47 | All releases |
| Renamed chapter 3 | *Chapter 3 Operation* on page 3-48 | All releases |
| Updated section | *3.1.1 SGIs* on page 3-49 | All releases |
| Updated section | *3.1.2 PPIs* on page 3-49 | All releases |
| Updated section | *3.1.3 SPIs* on page 3-49 | All releases |
| Updated section | *3.1.4 LPIs* on page 3-50 | All releases |
| Updated section | *3.2 Interrupt groups* on page 3-52 | All releases |
| Updated section | *3.3 Physical interrupt signals (PPIs and SPIs)* on page 3-53 | All releases |
| Updated section | *3.4 Affinity routing and assignment* on page 3-54 | All releases |
| Updated section | *3.5 1 of N SPI interrupt selection* on page 3-56 | All releases |
| Updated section | *3.6 Power management* on page 3-58 | All releases |
| Updated section | *3.8 Security* on page 3-62 | All releases |
| Updated section | *3.9 Backwards compatibility* on page 3-64 | All releases |
| Updated section | *3.10 Interrupt translation service (ITS)* on page 3-65 | All releases |
| Updated section | *3.12 Memory access and attributes* on page 3-69 | All releases |
| Updated table headings | *Table 3-5 Cacheability values* on page 3-69 | All releases |
| Updated section | *3.13 MSI-64* on page 3-71 | All releases |
| Updated section | *3.14 RAM* on page 3-72 | All releases |
| Updated section | *3.16 Reliability, Accessibility, and Serviceability* on page 3-75 | All releases |
| Updated section | *4.1.2 Discovery* on page 4-105 | All releases |
| Updated section | *4.2 Distributor registers (GICD/GICDA) summary* on page 4-106 | All releases |
| Updated section | *4.2.9 Interrupt Class Registers, GICD_ICLARn* on page 4-118 | All releases |
| Updated table | *Table 4-33 GICR_CFGID0 bit assignments* on page 4-139 | All releases |
| Updated table | *Table 4-34 GICR_CFGID1 bit assignments* on page 4-140 | All releases |
| Updated table | *Table 4-38 GITS_FCTLR bit assignments* on page 4-145 | All releases |
| Updated section | *4.6.5 Operation Status Register, GITS_OPSR* on page 4-147 | All releases |
| Updated table | *Table 4-40 GITS_OPSR bit assignments* on page 4-148 | All releases |
| Updated section | *4.7 ITS translation register summary* on page 4-151 | All releases |
| Updated table | *Table 4-48 GICT_ERR<n>STATUS bit assignments* on page 4-157 | All releases |
| Updated section | *4.8.4 Error Record Address Register, GICT_ERR<n>ADDR* on page 4-158 | All releases |
| Updated section | *4.8.5 Error Record Miscellaneous Register 0, GICT_ERR<n>MISC0* on page 4-159 | All releases |
| Updated table | *Table 4-50 GICT_ERR<n>MISC0 bit assignments* on page 4-159 | All releases |

**Table C-3 Differences between issue 0000-01 and issue 0002-00 (continued)**

| Change | Location | Affects |
|---|---|---|
| Updated table | *Table 4-51 Data field encoding* on page 4-160 | All releases |
| Updated section | *4.8.6 Error Record Miscellaneous Register 1, GICT_ERR<n>MISC1* on page 4-166 | All releases |
| Updated table | *Table 4-52 GICT_ERR10MISC1 bit assignments* on page 4-166 | All releases |
| Updated section | *4.9.2 Event Type Configuration Registers, GICP_EVTYPERn* on page 4-172 | All releases |
| Updated tables | *Table 4-59 GICP_EVTYPERn bit assignments* on page 4-172 and *Table 4-60 EVENT field encoding* on page 4-173 | All releases |
| Updated section | *A.2 Power control signals* on page Appx-A-187 | All releases |
| Updated section | *A.4 CPU interface signals* on page Appx-A-189 | All releases |
| Updated section | *A.5 ACE interface signals* on page Appx-A-190 | All releases |
| Updated section | *A.6 Miscellaneous signals* on page Appx-A-195 | All releases |
| Updated section | *A.7 Interblock signals* on page Appx-A-196 | All releases |
| Added section | *A.8 Interdomain signals* on page Appx-A-199 | All releases |
| Updated section | *B.1 Implementation-defined features reference* on page Appx-B-202 | All releases |

**Table C-4 Differences between issue 0002-00 and issue 0002-01**

| Change | Location | Affects |
|---|---|---|
| No technical changes | - | - |

**Table C-5 Differences between issue 0002-01 and issue 0003-00**

| Change | Location | Affects |
|---|---|---|
| Updated note about microarchitecture, and references to multichip | *1.1 About the GIC-600* on page 1-12 | r0p<n> |
| Updated note and tables | *Note* on page 2-25, *Table 2-1 AXI4-Stream input interface descriptions* on page 2-25, and *Table 2-2 AXI4-Stream output interface descriptions* on page 2-25. | All releases |
| Updated section | *2.1.4 Distributor Q-Channels* on page 2-27 | All releases |
| Updated note in table | *Table 2-8 Redistributor miscellaneous input signals* on page 2-32 | All releases |
| Updated the range of cores downstream | *2.2.6 Redistributor configuration* on page 2-32 | All releases |
| Updated text following table | *Table 2-10 ITS ACE-Lite slave interface acceptance capabilities* on page 2-36 | All releases |
| Clarified text | *2.3.2 ITS ACE-Lite master interface* on page 2-36 | All releases |
| Clarified text in all subsections | *2.5 SPI Collator* on page 2-42 | All releases |
| Clarified text | *2.6 Wake Request* on page 2-44 | All releases |
| Removed key requirements of the interconnect | *2.7 Interconnect* on page 2-45 | All releases |
| Added information | *3.9 Backwards compatibility* on page 3-64 | All releases |
| Added paragraph to introduction | *3.12 Memory access and attributes* on page 3-69 | All releases |

**Table C-5 Differences between issue 0002-01 and issue 0003-00 (continued)**

| Change | Location | Affects |
|---|---|---|
| Clarified text | *3.16.2 Scrub* on page 3-75 | All releases |
| Added information | *3.16.3 Error record classification* on page 3-75 | All releases |
| Clarified text for ITS caches and SPI | *Table 3-6 ECC error reporting* on page 3-76 | All releases |
| Records 5 and 6 changed to reserved | *Table 3-7 Error handling records* on page 3-77 | All releases |
| Updated text for syndromes `0x18` and `0x19` | *Table 3-8 Software errors, record 0* on page 3-78 | All releases |
| Error record name changed | *ITS command and translation error records 13+* on page 3-88 | All releases |
| Updated text in column Offset[x:16] | *Table 4-1 Register map pages* on page 4-104 | All releases |
| Added Architecture defined column and various text changes throughout table | *Table 4-2 Distributor registers (GICD/GICDA) summary* on page 4-106 | All releases |
| Updated section | *4.2.1 Distributor Control Register, GICD_CTLR* on page 4-109 | All releases |
| Updated section | *4.2.4 Function Control Register, GICD_FCTLR* on page 4-112 | All releases |
| Added text to table | *Table 4-13 GICD_IERRRn bit assignments* on page 4-120 | All releases |
| Added PEW field and updated SPIS function | *Table 4-14 GICD_CFGID bit assignments* on page 4-121 | All releases |
| Added sections | *4.2.12 Peripheral ID4 register, GICD_PIDR4* on page 4-121, *4.2.13 Peripheral ID3 register, GICD_PIDR3* on page 4-122, *4.2.16 Peripheral ID0 register, GICD_PIDR0* on page 4-124, *4.2.15 Peripheral ID1 register, GICD_PIDR1* on page 4-123 | All releases |
| Added columns Architecture defined, and Width | *Table 4-20 Distributor registers (GICA) for message-based SPIs summary* on page 4-125 | All releases |
| Added Architecture defined column | *Table 4-21 Redistributor registers for control and physical LPIs summary* on page 4-126, *Table 4-29 Redistributor registers for SGIs and PPIs summary* on page 4-135, and *Table 4-35 ITS control register summary* on page 4-141 | All releases |
| Updated descriptions of bits UEE, CEE, and LTE | *Table 4-38 GITS_FCTLR bit assignments* on page 4-145 | All releases |
| Updated section | *4.6.6 Configuration ID Register, GITS_CFGID* on page 4-148 | All releases |
| Added Architecture defined column | *4.7 ITS translation register summary* on page 4-151, and *Table 4-44 GICT register summary* on page 4-152 | All releases |
| Records 5 and 6 changed to reserved | *Table 4-45 Error records* on page 4-153 | All releases |
| Added information fields CE, IERR, SERR | *Table 4-48 GICT_ERR<n>STATUS bit assignments* on page 4-157 | All releases |
| Updated text at syndromes `0x4` and `0x19` | *Table 4-51 Data field encoding* on page 4-160 | All releases |
| Added section | *4.8.7 Error Group Status Register, GICT_ERRGSR* on page 4-167 | All releases |
| Added Architecture defined column, and registers GICP_IRQCR, GICP_PMAUTHSTATUS, and GICP_PMDEVARCH | *Table 4-57 GICP register summary* on page 4-170 | All releases |
| Added section | *4.9.14 Interrupt Configuration Register, GICP_IRQCR* on page 4-182 | All releases |
| Corrected signal names | *A.4 CPU interface signals* on page Appx-A-189 | All releases |
| Clarified introduction text | *Appendix B Implementation-defined features* on page Appx-B-201 | All releases |

**Table C-6 Differences between issue 0003-00 and issue 0102-00**

| Change | Location | Affects |
|---|---|---|
| Updated section, Distributor | *Distributor* on page 1-13 | r1p2 |
| Added figures | *Figure 1-1 GIC-600 with free-flowing interconnect in an example system* on page 1-15, *Figure 1-2 GIC-600 with interconnect in an example system* on page 1-16, and *Figure 1-3 Monolithic GIC-600 with interconnect in an example system* on page 1-17 | r1p2 |
| Added note following Figure 1-2 | *Top level* on page 1-14 | r1p2 |
| Updated paragraph following Figure 1-3 | *Top level* on page 1-14 | All releases |
| Added bullet point | *Registers and programming:* on page 1-19 | r1p2 |
| Updated section | *Error correction:* on page 1-19 | All releases |
| Updated section | *1.7 Product revisions* on page 1-22 | r1p2 |
| Updated note and tables | *Note* on page 2-25, *Table 2-1 AXI4-Stream input interface descriptions* on page 2-25, and *Table 2-2 AXI4-Stream output interface descriptions* on page 2-25. | All releases |
| Updated paragraph following Table | *Table 2-3 Distributor ACE-Lite slave interface acceptance capabilities* on page 2-26 | All releases |
| Updated Note | *2.1.3 Distributor ACE-Lite master interface* on page 2-26 | All releases |
| Updated section | *2.1.4 Distributor Q-Channels* on page 2-27 | All releases |
| Added section | *2.1.5 P-Channel* on page 2-27 | r1p2 |
| Updated table | *Table 2-6 Configurable options for the Distributor* on page 2-28 | r1p2 |
| Updated table | *Table 2-8 Redistributor miscellaneous input signals* on page 2-32 | All releases |
| Updated table | *2.2.6 Redistributor configuration* on page 2-32 | All releases |
| Updated table for combined acceptance capability attribute, write acceptance capability, and updated text following table | *Table 2-10 ITS ACE-Lite slave interface acceptance capabilities* on page 2-36 | All releases |
| Updated section | *2.3.2 ITS ACE-Lite master interface* on page 2-36 | All releases |
| Updated section | *2.5 SPI Collator* on page 2-42 | All releases |
| Updated section | *2.5.1 SPI Collator AXI4-Stream interface* on page 2-42 | All releases |
| Updated section and changed section title from SPI Collator Q-Channel | *2.5.3 SPI Collator power Q-Channel* on page 2-42 | All releases |
| Updated table | *2.5.4 SPI Collator configuration* on page 2-43 | r1p2 |
| Updated section | *2.5.5 SPI Collator clock Q-Channel* on page 2-43 | All releases |
| Updated section | *2.6 Wake Request* on page 2-44 | All releases |
| Deleted key requirements of the interconnect | *2.7 Interconnect* on page 2-45 | All releases |
| Added text to section | *2.8 Hierarchy* on page 2-46 | All releases |
| Updated introduction | *3.1.2 PPIs* on page 3-49 | All releases |
| Updated section | *3.1.3 SPIs* on page 3-49 | All releases |

**Table C-6 Differences between issue 0003-00 and issue 0102-00 (continued)**

| Change | Location | Affects |
|---|---|---|
| Updated section | *3.9 Backwards compatibility* on page 3-64 | All releases |
| Added paragraph to introduction | *3.12 Memory access and attributes* on page 3-69 | All releases |
| Updated first sentence to subsection "Snapshot" | *Snapshot* on page 3-74 | All releases |
| Updated introduction | *3.16 Reliability, Accessibility, and Serviceability* on page 3-75 | All releases |
| Updated section | *3.16.2 Scrub* on page 3-75 | All releases |
| Updated section | *3.16.3 Error record classification* on page 3-75 | All releases |
| Added Target cache to table | *Table 3-6 ECC error reporting* on page 3-76 | All releases |
| Updated records 5 and 6 | *Table 3-7 Error handling records* on page 3-77 | All releases |
| Updated cells `0x18` and `0x19` | *Table 3-8 Software errors, record 0* on page 3-78 | All releases |
| Updated name of error record, and added text to introduction | *ITS command and translation error records 13+* on page 3-88 | All releases |
| Added section for multichip operation | *3.17 Multichip operation* on page 3-97 | r1p2 |
| Updated table | *Table 4-1 Register map pages* on page 4-104 | All releases |
| Updated table | *Table 4-2 Distributor registers (GICD/GICDA) summary* on page 4-106 | All releases |
| Updated section | *4.2.1 Distributor Control Register, GICD_CTLR* on page 4-109 | All releases |
| Updated section | *4.2.4 Function Control Register, GICD_FCTLR* on page 4-112 | All releases |
| Added section | *4.2.6 Chip Status Register, GICD_CHIPSR* on page 4-115 | r1p2 |
| Added section | *4.2.7 Default Chip Register, GICD_DCHIPR* on page 4-116 | r1p2 |
| Added section | *4.2.8 Chip Registers, GICD_CHIPR<n>* on page 4-117 | r1p2 |
| Updated table | *Table 4-13 GICD_IERRRn bit assignments* on page 4-120 | All releases |
| Added bitfield PEW | *Table 4-14 GICD_CFGID bit assignments* on page 4-121 | All releases |
| Added sections | *4.2.12 Peripheral ID4 register, GICD_PIDR4* on page 4-121, *4.2.13 Peripheral ID3 register, GICD_PIDR3* on page 4-122, *4.2.16 Peripheral ID0 register, GICD_PIDR0* on page 4-124, *4.2.15 Peripheral ID1 register, GICD_PIDR1* on page 4-123 | All releases |
| Updated table | *Table 4-20 Distributor registers (GICA) for message-based SPIs summary* on page 4-125 | All releases |
| Updated table | *Table 4-21 Redistributor registers for control and physical LPIs summary* on page 4-126 | All releases |
| Added note to bitfield CGO function description | *Table 4-25 GICR_FCTLR bit assignments* on page 4-131 | All releases |
| Updated table | *Table 4-29 Redistributor registers for SGIs and PPIs summary* on page 4-135 | All releases |
| Updated table | *Table 4-35 ITS control register summary* on page 4-141 | All releases |
| Updated descriptions of bits LTE, UEE, CEE, and CGO | *Table 4-38 GITS_FCTLR bit assignments* on page 4-145 | All releases |
| Updated section | *4.6.6 Configuration ID Register, GITS_CFGID* on page 4-148 | All releases |
| Updated table | *4.7 ITS translation register summary* on page 4-151 | All releases |

**Table C-6 Differences between issue 0003-00 and issue 0102-00 (continued)**

| Change | Location | Affects |
|---|---|---|
| Updated table | *Table 4-44 GICT register summary* on page 4-152 | All releases |
| Updated table | *Table 4-45 Error records* on page 4-153 | All releases |
| Updated table | *Table 4-48 GICT_ERR<n>STATUS bit assignments* on page 4-157 | All releases |
| Updated table | *Table 4-51 Data field encoding* on page 4-160 | All releases |
| Added section | *4.8.7 Error Group Status Register, GICT_ERRGSR* on page 4-167 | All releases |
| Added section | *4.8.8 Error Interrupt Configuration Registers, GICT_ERRIRQCR<n>* on page 4-167 | All releases |
| Updated table | *Table 4-57 GICP register summary* on page 4-170 | All releases |
| Added section | *4.9.14 Interrupt Configuration Register, GICP_IRQCR* on page 4-182 | All releases |
| Added power P-Channel signals **preq**, **pstate**, **paccept**, **pdeny**, **pactive** | *A.2 Power control signals* on page Appx-A-187 | r1p2 |
| Updated signal names | *A.4 CPU interface signals* on page Appx-A-189 | All releases |
| Added signals **icdptdest**, **icdptkeep**, **icpdtid**, **icpdtkeep**, **icditdest**, **icditkeep**, **icidtdid**, **icidtkeep**, **iccdtdest**, and **iccdtid**. | *A.7 Interblock signals* on page Appx-A-196 | All releases |
| Added section for interchip signals | *A.9 Interchip signals* on page Appx-A-200 | r1p2 |
| Updated introduction | *Appendix B Implementation-defined features* on page Appx-B-201 | All releases |

**Table C-7 Differences between issue 0102-00 and issue 0103-00**

| Change | Location | Affects |
|---|---|---|
| Added pseudocode | *3.6.1 Redistributor power management* on page 3-58 | All revisions |
| Added information about single operation of the Routing table | *3.17.1 About multichip operation* on page 3-97 | All revisions |
| Updated the procedure | *3.17.2 Connecting the chips* on page 3-98 | All revisions |
| Updated the description | *3.17.4 SPI ownership* on page 3-99 | All revisions |
| Added information about the Routing table during power up. | *3.17.6 Isolating a chip from the system* on page 3-100 | All revisions |
| Added an r1p3 entry to the Version field description. | *Table 4-34 GICR_CFGID1 bit assignments* on page 4-140 | r1p3 |
| Corrected the UE bit description | *Table 4-47 GICT_ERR<n>CTLR bit assignments* on page 4-156 | All revisions |
| Added what function GICT_ERRIRQCR0 controls. Updated the SPIID description. | *4.8.8 Error Interrupt Configuration Registers, GICT_ERRIRQCR<n>* on page 4-167 | All revisions |
| Corrected the description of the ACC event | *Table 4-60 EVENT field encoding* on page 4-173 | All revisions |
| Updated the SPIID description. | *4.9.14 Interrupt Configuration Register, GICP_IRQCR* on page 4-182 | All revisions |