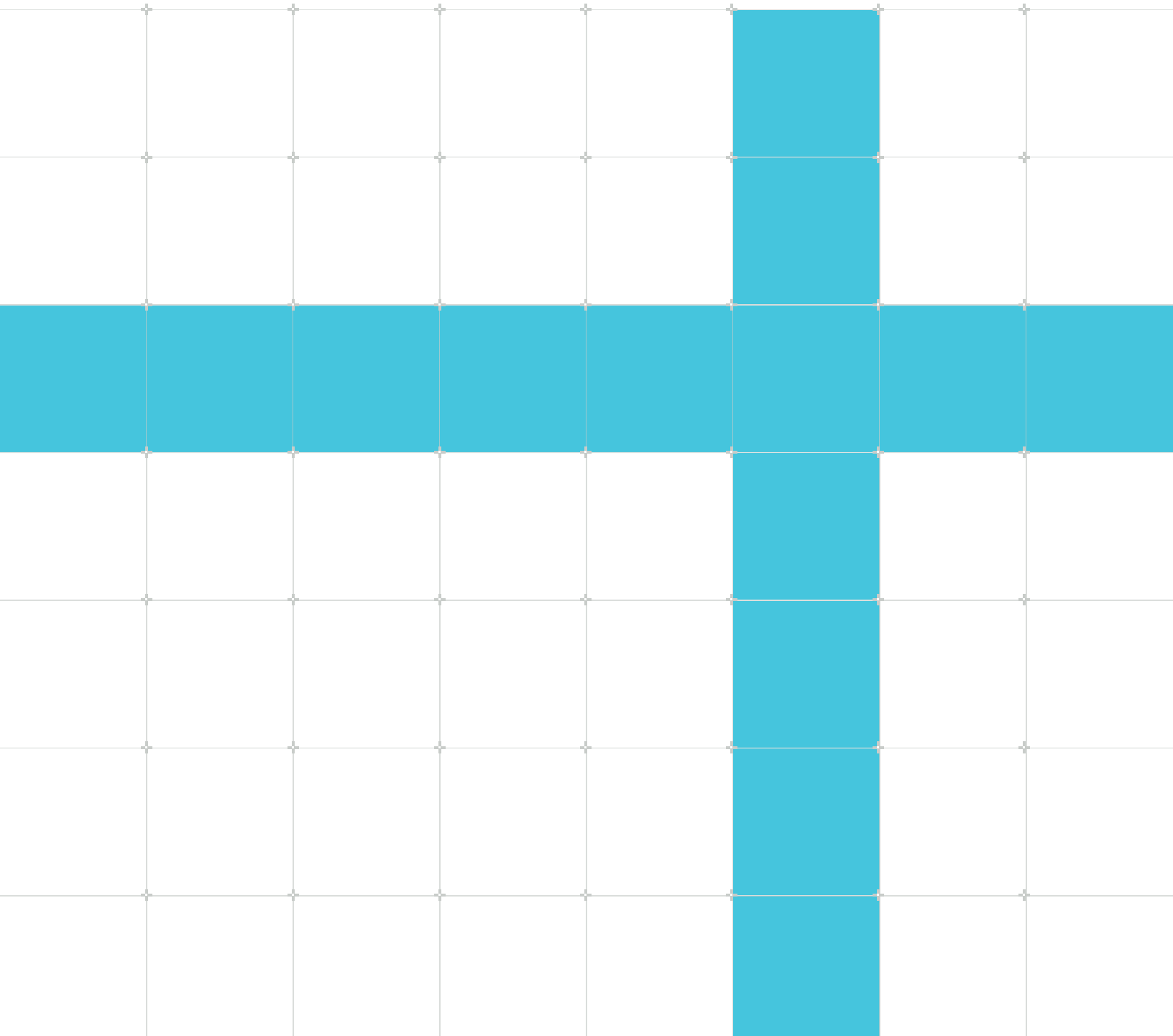
Arm® CoreLink™ GIC-600AE Generic Interrupt Controller

Revision: r0p3

Technical Reference Manual

Non-Confidential

Issue 04
101206_0003_04_en



Arm® CoreLink™ GIC-600AE Generic Interrupt Controller

Technical Reference Manual

Release Information

Document history

Issue	Date	Confidentiality	Change
0000-00	31 July 2018	Confidential	First beta release for r0p0
0000-01	9 November 2018	Non-Confidential	First early access release for r0p0
0001-02	30 August 2019	Non-Confidential	First early access release for r0p1
0002-03	24 April 2020	Non-Confidential	First release for r0p2
0003-04	16 September 2022	Non-Confidential	First release for r0p3

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.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com.

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

Previous issues of this document included language that can be offensive. We have replaced this language. See C. Revisions on page 265.

To report offensive language in this document, email terms@arm.com.

Contents

1. Introduction

1.1 Product revision status

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

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

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

1.2 Intended audience

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

1.3 Conventions

The following subsections describe conventions used in Arm documents.

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: developer.arm.com/glossary.

Convention	Use
italic	Citations.
bold	Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<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>`

Convention	Use
SMALL CAPITALS	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.

Recommendations. Not following these recommendations might lead to system failure or damage.

Requirements for the system. Not following these requirements might result in system failure or damage.

Requirements for the system. Not following these requirements will result in system failure or damage.

An important piece of information that needs your attention.

A useful tip that might make it easier, better or faster to perform a task.

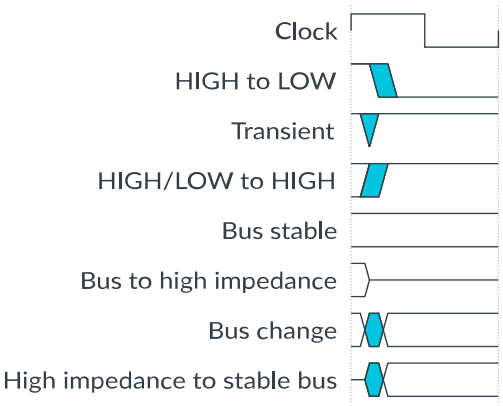
A reminder of something important that relates to the information you are reading.

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 1-1: Key to timing diagram conventions



Signals

The signal conventions are:

Signal level

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

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

Lowercase n

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

1.4 Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Arm product resources	Document ID	Confidentiality
Arm® CoreLink™ ADB-400 AMBA® Domain Bridge User Guide	DUI 0615	Confidential
Arm® CoreLink™ CMN-600 Coherent Mesh Network Technical Reference Manual	100180	Non-Confidential
Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Configuration and Integration Manual	101207	Confidential
Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Development Interface Report	101209	Confidential
Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Safety Manual	101208	Confidential

Arm product resources	Document ID	Confidentiality
GIC-600AE Dependent Failure Analysis Report	PJDOC-1779577084-15964	Confidential
GIC-600AE FMEDA Report	PJDOC-1779577084-15988	Confidential

Arm architecture and specifications	Document ID	Confidentiality
AMBA® AXI and ACE Protocol Specification	IHI 0022F	Non-Confidential
AMBA® 4 AXI4-Stream Protocol Specification	IHI 0051A	Non-Confidential
AMBA® Low Power Interface Specification	IHI 0068D	Non-Confidential
Arm® Architecture Reference Manual Armv8, for A-profile architecture	DDI 0487G.b	Non-Confidential
Arm® Architecture Reference Manual Supplement Reliability, Availability, and Serviceability (RAS), for Armv8-A	DDI 0587D.c	Non-Confidential
Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4	IHI 0069E	Non-Confidential
GICv3 and GICv4 Software Overview	DAI 0492	Non-Confidential

Non-Arm resources	Document ID	Organization
Standard Manufacturer's Identification Code	JEP106	JEDEC

Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at http://www.adobe.com

2. About the GIC-600AE

The GIC-600AE is a *Functional Safety* (FuSa) variant of the GIC-600. The GIC-600AE is a *Generic Interrupt Controller* (GIC) that handles interrupts from peripherals to the cores and between cores. The GIC-600AE supports a distributed microarchitecture containing several individual blocks that are used to provide a flexible GIC implementation.

The GIC-600AE supports the GICv3 architecture. For more information, see the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.

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

All the GIC-600AE blocks communicate through fully credited AXI4-Stream interface channels. This means that the interface only exerts transient backpressure 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 when all transactions on that channel complete without a non-transient dependency on any other transaction.

The GIC-600AE 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.


Note

GIC-600 information is unchanged, and information about the FuSa features available in GIC-600AE can be found in 6. Functional Safety on page 193.

2.1 Components

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

The GIC-600AE 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.

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

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-600AE Redistributor. However, the command and control pages for all cores on a chip are contained in the Distributor.

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

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-600AE supports up to eight ITS blocks per chip.

For more information about the ITS, see the 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.

SPI Collator

The GIC-600AE 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.

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.

GIC interconnect

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

Top level

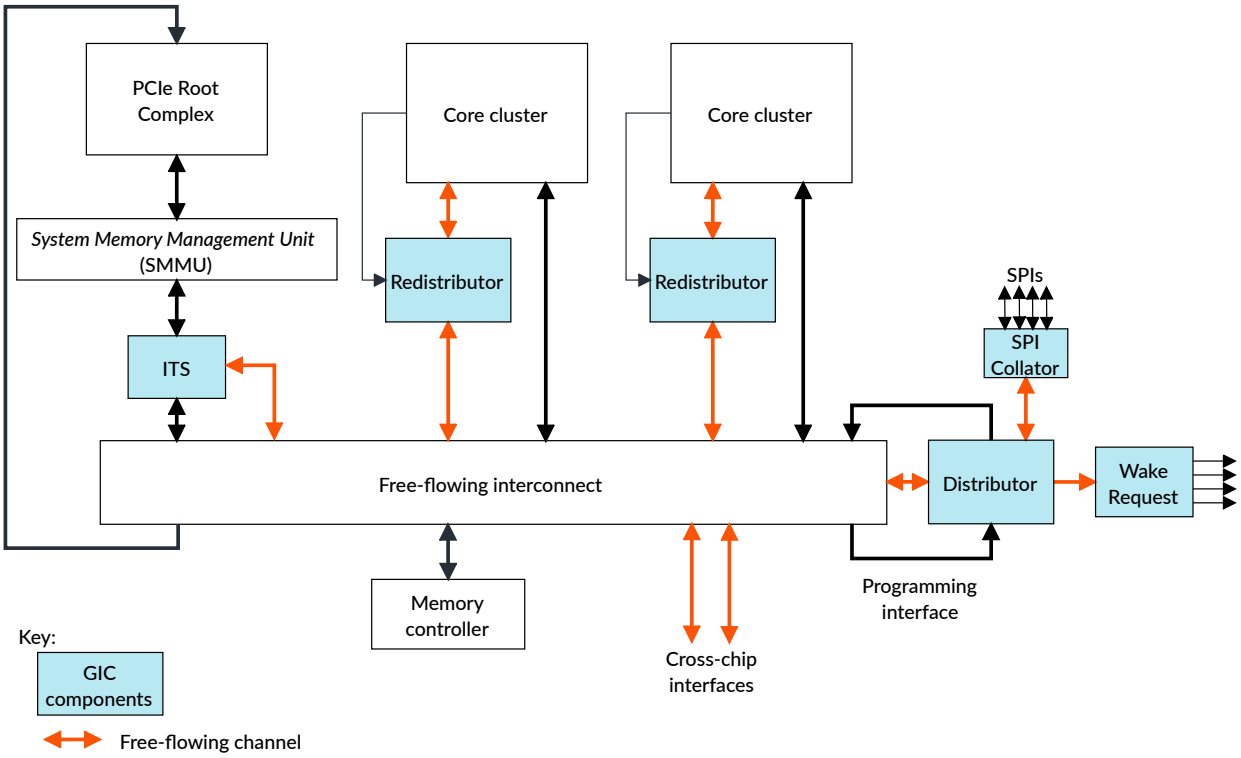
The top level has no specific interfaces but combines the interfaces of other blocks within the clock or power domain to reduce the number of domain bridges. The GIC-600AE build scripts enable you to build the GIC from a single combined block or a set of individual blocks that are interconnected using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

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-600AE includes a narrow, 16-bit, AXI4-Stream interconnect that can be used for routing internal traffic.

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

Figure 2-1: GIC-600AE with free-flowing interconnect in an example system

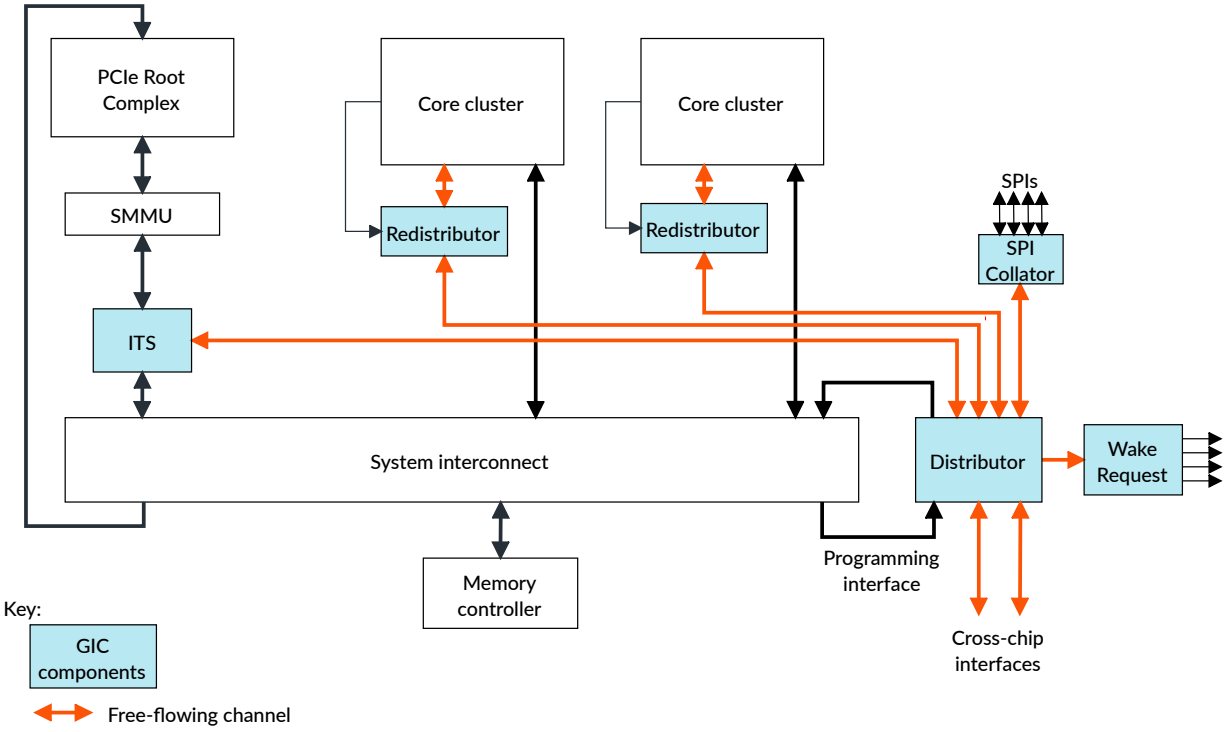


Note

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

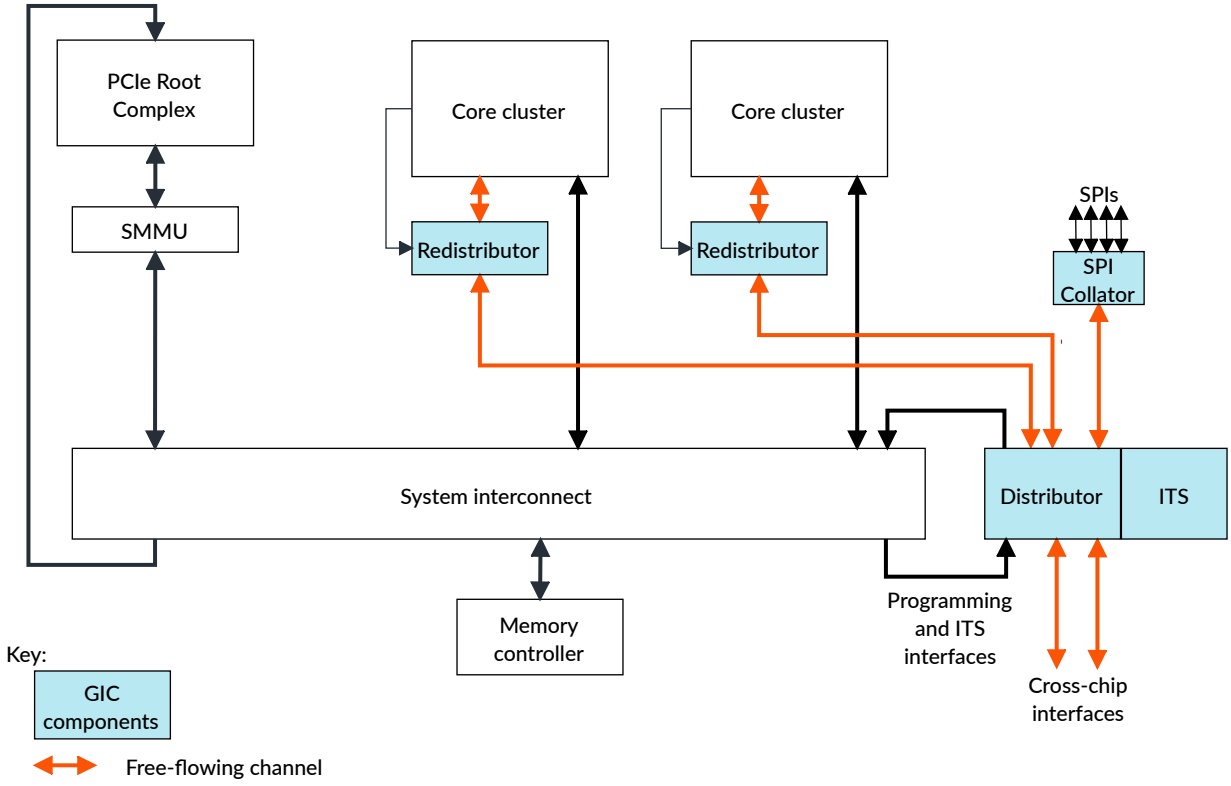
The following figure shows a GIC-600AE with interconnect in an example system. The cross-chip interfaces enable communication between cores in a multichip configuration.

Figure 2-2: GIC-600AE with interconnect in an example system



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

Figure 2-3: Monolithic GIC-600AE with interconnect in an example system



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-600AE Generic Interrupt Controller Configuration and Integration Manual*.

The GIC-600AE supports cores that implement only the Armv8.0-A architecture, and later versions such as Armv8.2-A. The cores must also support the GIC CPU interface with the standard GIC AXI4-Stream protocol interface. The GIC-600AE implements version 3.0 of the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.

Related information

Components and configuration on page 25

2.2 Compliance

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

The GIC-600AE is compliant with:

- Version 3.0 of the Arm GIC architecture specification. See the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.
- The AMBA® ACE-Lite protocol. See the AMBA® AXI and ACE Protocol Specification.

- The AMBA® AXI4-Stream protocol. See the AMBA® 4 AXI4-Stream Protocol Specification.
- The GIC Stream protocol. See the *GIC Stream Protocol interface* appendix in the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.

2.3 Features

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

Interrupt services and masking

The GIC-600AE provides the following interrupt services and masking features:

- 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-600AE. See 3.1.6 Distributor configuration on page 30.
  - Up to 960 SPIs in groups of 32. See 3.1.6 Distributor configuration on page 30.
  - Up to 16 PPIs that are independent for each core and can be programmed to support either edge-triggered or level-sensitive interrupts. See 3.2.5 Redistributor configuration on page 33.
  - Up to 16 SGIs that are generated through the GIC CPU interface of a core.
- Up to eight 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

The GIC-600AE provides the following programming features:

- Flexible affinity routing, using the *Multiprocessor Identification Register* (MPIDR) addresses, including support for all four affinity levels.
- Single ACE-Lite subordinate 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 subordinate port for programming the GITS_TRANSLATER register.
- Coherent view of SPI register data across multiple chips.

Security

The GIC-600AE provides the following security features:

- 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 4.2 Interrupt groups and security on page 50 for more information about security and groupings.


Note

For more information about Exception levels, see the Arm® Architecture Reference Manual Armv8, for A-profile architecture.

Performance monitoring

The GIC-600AE provides the following performance monitoring features:

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

Error correction

The GIC-600AE provides the following error correction features:

- Armv8.2 *Reliability Accessibility Serviceability* (RAS) architecture-compliant error reporting for:
  - Software access errors
  - ITS command and translation errors
  - *Error Correcting Code* (ECC) errors

2.4 Test features

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

Related information

Common control signals on page 250

2.5 Product documentation

Documentation that is provided with this product includes a *Technical Reference Manual* (TRM), a *Configuration and Integration Manual* (CIM), a *Safety Manual*, and a *Development Interface Report*.

For relevant protocol and architectural information that relates to this product, see 1.4 Useful resources on page 14.

The GIC-600AE documentation is as follows:

Technical Reference Manual

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

- The implementer to determine:
  - The build configuration of the implementation
  - What integration, if any, was performed before implementing the GIC-600AE
- 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-600AE into an SoC
- How to implement the GIC-600AE 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 document that is only available to licensees.

Safety Manual

The SM provides additional information on specific features of the GIC-600AE that are relevant to Functional Safety. This information is important for SoC integrators whose final designs target applications where Functional Safety is a concern.
The SM is a confidential document that is only available to licensees.

Development Interface Report

The DIR describes the activities conducted by Arm that are related to the safety architecture of the GIC-600AE.
The DIR is a confidential document that is only available to licensees.

2.6 Product revisions

This section describes the differences in functionality between product revisions.

r0p0 First release

r0p0-r0p1 Functional changes are:

- Bug fixes
- Writing `0b10` to FMU_ERR<n>STATUS.CE, clears the CE field
- The FMU pready signal is gated when in Q-Channel low-power state and no faults have been reported to FMU.

r0p1-r0p2 Functional changes are:

- Bug fixes
- To align with GICv2m, the GICA page supports the GICA_TYPER, GICA_IIDR, and GICA_PIDR*, and GICA_CIDR* registers. See 5.3 Distributor registers (GICM) for message-based SPIs summary on page 117.

r0p2-r0p3 Functional changes are:

- Bug fixes
- The GICA registers are renamed to GICM registers. See 5.3 Distributor registers (GICM) for message-based SPIs summary on page 117.

3. Components and configuration

The GIC-600AE contains several major components that use an internal GIC interconnect to route the AXI4-Stream interfaces between the different components. A configuration parameter controls the hierarchy of the GIC components.

The components are:

- Distributor
- Redistributor
- *Interrupt Translation Service* (ITS)
- MSI-64 Encapsulator
- SPI Collator
- Wake Request
- GIC interconnect

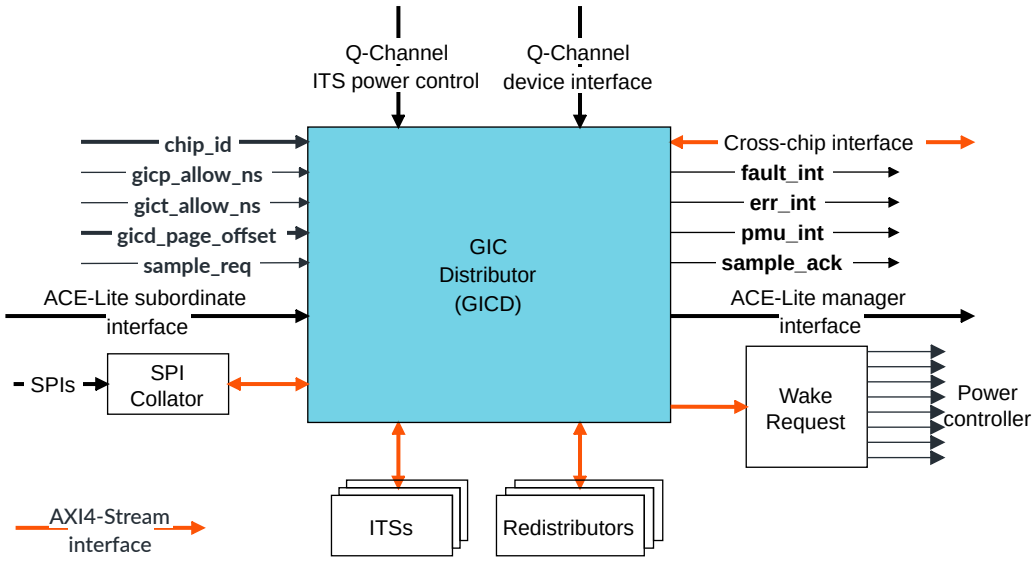
The hierarchy of the GIC components can be a single combined block that uses a dedicated AXI4-Stream interconnect or a set of individual blocks that are interconnected using your own transport layer.

3.1 Distributor

The Distributor is the main communication point between all GIC-600AE 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 3-1: GIC-600AE 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 4.1.3 SPIs on page 49
- Caching and forwarding of LPIs, see 4.1.4 LPIs on page 49
- SGI routing and forwarding
- Register programming of all registers apart from GITS_TRANSLATER
- Power control of cores and Redistributor blocks

3.1.1 Distributor AXI4-Stream interfaces

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

These interfaces are:

- fully credited
- ic<xy>tready. Where xy can be cd, dc, pd, dp, id, di, rd, dr, or dw.

Irrespective of the interconnect that is used, packets must not be reordered between endpoints, for example, between the Distributor and a single Redistributor block. Packets must never be interleaved.

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

The following table lists the AXI4-Stream input interfaces.

Table 3-1: 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 3-2: 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 3.1.4 Distributor Q-Channels on page 29.

3.1.2 Distributor ACE-Lite subordinate interface

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

The GIC-600AE only accepts single beat accesses of the sizes for each register that are shown in the programmers model, see 5. Programmers model on page 95. 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-600AE is a monolithic configuration without MSI-64 support, the Distributor and ITS both share an ACE-Lite subordinate port, and the DeviceID for the ITS translation is taken from the awuser_s[did_width+2:3] signal. The value of the `did_width` parameter is set during silicon integration. For more information about the ITS, see 3.3 Interrupt Translation Service on page 34.

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 subordinate interface.

Table 3-3: Distributor ACE-Lite subordinate interface acceptance capabilities

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

The GIC-600AE uses awatop_s, a<x>cache_s, a<x>domain_s, a<x>snoop_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-600AE also ignores other Cacheability, Shareability, and protection settings, except for the a<x>prot_s[1] security signal.

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

The GIC-600AE 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 signal) to ensure that the GIC does not request to be woken up due to incoming signal glitches.

The GIC-600AE address map has multiple pages. The number of pages and the address aliasing depends on your configuration. See 5.1 Register map pages on page 95.

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.

Related information

Register map pages on page 95

3.1.3 Distributor ACE-Lite manager interface

The GICD uses the AMBA® ACE-Lite manager interface to access all pending, property, and translation tables that are allocated to the GIC. 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 manager interface.

Table 3-4: Distributor ACE-Lite manager 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

Attribute	Capability		
	Read	Write	Combined
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 manager interface cannot issue barriers or *Cache Maintenance Operations* (CMOs). However, it can issue shareable, ReadOnce and WriteUnique, transactions if programmed to do so.

See 4.11 Memory access and attributes on page 64 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.

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

3.1.4 Distributor Q-Channels

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

The Distributor also has a separate Q-Channel that enables power control for each configured ITS. The GIC only accepts a low-power request when GITS_CTLR.Quiescent is set. If the Quiescent bit is set, the Q-Channel qacceptn_its_<n> 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, then when the qactive_gicd signal asserts on the corresponding ITS, the system must re-enable the bus and program the GICD so that it is ready to receive LPIs. The system must route the qactive_gicd signal to a power controller that implements the following sequence:

1. Power up the GICD
2. Restore the GICD program state
3. Turn on the associated ITS Q-Channel on the GICD, which allows the ITS to proceed

The qreqn* signals are synchronized internally, and can be driven asynchronously. See A.2 Power control signals on page 251.

As the qactive output signal includes combinatorial and asynchronous inputs, then you must consider qactive as an asynchronous output.

For more information, see the AMBA® Low Power Interface Specification.

3.1.5 Distributor P-Channel

The P-Channel is used for power control of the GIC-600AE 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 information

Power management on page 56

3.1.6 Distributor configuration

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

Table 3-5: 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	0-4
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
Number of SPI wires on chip for wire-based SPIs	0-960
Security support	Options include: • Security support programmable. Resets to support security. • Security support always present • Security support not present See *Security model* in the GICv3 and GICv4 Software Overview for information about the implications of setting security support to not present.

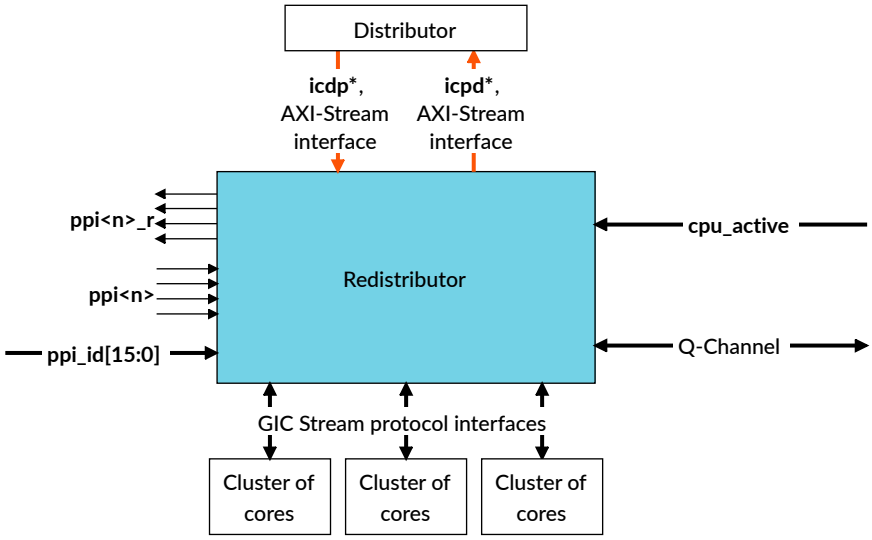
For more information, see the *Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Configuration and Integration Manual*.

3.2 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 3-2: GIC-600AE 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-600AE Generic Interrupt Controller Configuration and Integration Manual* for more information.


Note

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

Related information

Interrupt types on page 48

3.2.1 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.

3.2.2 Redistributor GIC Stream Protocol interface

The GIC-600AE 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, that is, 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.

See the *GIC Stream Protocol interface* appendix in the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4 for more information.

Table 3-6: GIC Stream Protocol interface signals

Signal	Description
iri	The iri prefix identifies the names of the downstream interface signals. These signals are sent by the GIC Stream transmitter. On this interface, the Redistributor is the transmitter and the CPU interface is the receiver.
icc	The icc prefix identifies the names of the upstream interface signals. These signals are sent by the GIC Stream receiver. On this interface, the CPU interface is the transmitter and the Redistributor is the receiver.
iritdest	The Redistributor uses this signal to direct packets to one core within the cluster
icctid	The cluster uses this signal to determine which core within the cluster sent a packet
iritwakeup	The Redistributor uses this signal to indicate that it wants to send a message to a CPU interface in the cluster
icctwakeup	The cluster uses this signal to indicate that it wants to send a message to the Redistributor

Both the iritdest and icctid signals can support 64 cores that use packed binary encoding, as opposed to one-hot encoding.

3.2.3 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 upstream or downstream, the Q-Channel denies a quiescence request that it receives on the qreqn signal, by asserting the qdeny signal. For more information, see the AMBA® Low Power Interface Specification.

The qreqn input signal is synchronized inside the Redistributor. The qactive signal is connected to the PPI wires directly, and must be considered as an asynchronous output.

Related information

Power control signals on page 251

3.2.4 Redistributor PPI signals

GIC-600AE 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 the following build-time parameters:

- `GIC600AE_<usrcfg>_PPI<ppi_id>_<cpu_number>_<ppi_number>_<INV>`
- `GIC600AE_<usrcfg>_PPI<ppi_id>_<cpu_number>_<ppi_number>_<SYNC>`
  Where `<usrcfg>` is user-defined text that is assigned when the GIC is configured, which can help with identifying a GIC configuration.

Every ppi<n> signal has a corresponding ppi<n>_r signal, from after the synchronizer or capture flop. These ppi<n>_r signals can be used to create pulse extenders for edge-triggered interrupts that cross clock domains.

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

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

3.2.5 Redistributor configuration

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

Table 3-7: Configurable options for the Redistributor

Feature	Range of options
Number of cores downstream	1-64
PPIs per core	8, 12, 16
ECC support. See 4.15 Reliability, Accessibility, and Serviceability on page 68 for more information.	True, False
Bus data width	16, 32
GIC Stream bus structure	Flexible buses and domains

For more information, see the *Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Configuration and Integration Manual*.

3.3 Interrupt Translation Service

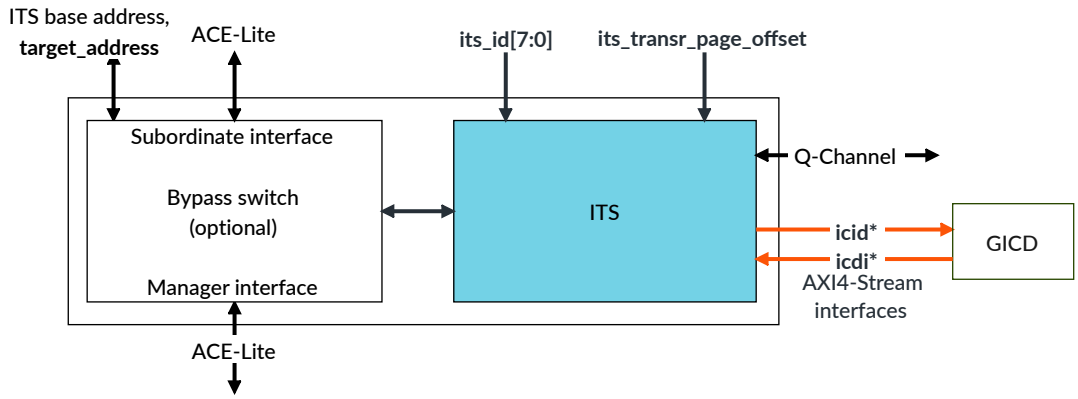
The ITS provides a software mechanism for translating message-based interrupts into LPIs. The ITS is supported optionally in configurations that support LPIs.

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 that is 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 3-3: 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 and version 4. 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 subordinate and manager ports connect to the ITS directly. See 3.3.1 ITS ACE-Lite subordinate interface on page 36 and 3.3.2 ITS ACE-Lite manager interface on page 37.

In accordance with PCIe dependency rules, read responses on a PCIe Root Complex subordinate port must be ordered against completion of posted writes on a Root Complex manager port.

This behavior means that writes must always make forward progress. The functionality of the ITS means that there is a dependency between writes to the GITS_TRANSLATER register and reads to memory, so one of the following conditions must be true:

- The interconnect must allow forward progress of reads under all circumstances.
- The GIC parameter `dgi_mem_support` must be set.

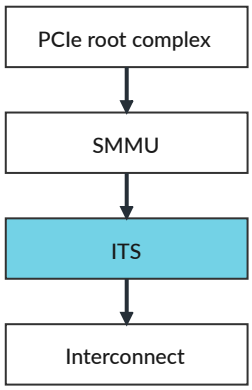
- This option provides support for routing all ITS translation-dependent traffic through the ACE-Lite manager port on the Distributor which must have free flowing access to memory. After this feature is configured, it must be enabled at boot time. To enable this feature, write to GITS_FCTLR.DMA to route the traffic through the Distributor.
- If `dgi_mem_support` is set, the ITS uses its ACE-Lite manager interface to access the Command queue, and uses the Distributor ACE-Lite manager interface to access tables.
- The ITS manager interface sends one single outstanding read and one single outstanding write at a time to access the Command queue. The ARID signal is `0x4` for the single outstanding read. The AWID signal is `0x0` for the single outstanding write.
- Setting `dgi_mem_support` = 1 increases the width of the AxID signals on the Distributor manager interface.

If neither condition is true, you must not use the configuration that Figure 3-3: ITS block on page 34 shows. This condition 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 subordinate 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 manager ports.

For more information, see 4.9 ITS on page 60.

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

Figure 3-4: 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 3.4 MSI-64 Encapsulator on page 40.

If the ITS is placed downstream of an ACE interconnect, care must be taken to avoid system deadlock. For more information, see *Functional integration* in the *Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Configuration and Integration Manual*.

For more information about each inner block, see 4.9 ITS on page 60.

3.3.1 ITS ACE-Lite subordinate interface

The ITS AMBA® ACE-Lite subordinate interface has a configurable data width of 64 bits, 128 bits, or 256 bits. The address and data widths between the subordinate and manager must match.

The ITS ACE-Lite subordinate port contains only the GITS_TRANSLATER register. See the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4 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 target_address[ADDR_WIDTH−17:0] ITS base address tie-off. 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. Writes to the ITS subordinate interface must set the awaddr[16:0] signal to `0x0040`, irrespective of whether the bypass switch is selected.

The ACE-Lite subordinate interface ignores all awatop, a<x>snoop, a<x>cache, a<x>domain, and a<x>prot signals information other than to filter non-memory transactions such as atomics and cache maintenance operations, to ensure that it replies in a protocol-compliant manner.

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

The GIC-600AE supports two different methods for deriving the DeviceID with the ACE-Lite subordinate interface:

- When using the MSI-64 configuration 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_s[did_width+2:3] signal during the address (AW) phase of the register access. In this case, burst length 1, 32-bit or 16-bit writes are accepted.

These two modes cannot be mixed on a single ITS. The DeviceID must be transferred using a method that malicious software cannot spoof.

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` parameter:

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 subordinate port contains only write-only registers, so the read channel always uses a simple transaction tracker that only allows transactions to one destination at a time.
- If the bypass switch is configured, the subordinate and manager ports must both have the same data width and the same address width.
- If the Distributor and ITS both share the ACE-Lite subordinate port, the port properties match those of the Distributor ACE-Lite subordinate port, which 3.1.2 Distributor ACE-Lite subordinate interface on page 27 describes.

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

Table 3-8: ITS ACE-Lite subordinate 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	`bypass_max_outstanding`, but not exceeding 256	2

The ITS ACE-Lite subordinate 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, the awakeup signal must be driven from a registered version of the awvalid and arvalid signals. To prevent spurious wake events, ensure that the awakeup signal is registered cleanly.

3.3.2 ITS ACE-Lite manager interface

The ITS AMBA® ACE-Lite manager 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 manager port issues accesses to the ITS private tables and Command queue. If the bypass switch is configured, the port also forwards transactions from the subordinate interface. The

ACE-Lite bus can issue I/O coherent transactions, therefore you can place these tables in shared memory if necessary. 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 subordinate port and reads on its manager port. You must ensure that any writes that back up to the subordinate 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 subordinate port. If the write channel is blocked, and the snoop is prevented from completing its task, a potential deadlock can result.

We recommend 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 subordinate 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 the a<x>id[0] signal.

Note

The ITS issues only one outstanding transaction per ID. This behavior gives a maximum of one outstanding write and five outstanding reads, excluding any transactions from the subordinate port. If this port is combined with the Distributor ACE-Lite manager port, some of these properties are changed. See Figure 3-8: GIC-600AE top-level structure options on page 47 for more information.

For more information, see the GICv3 and GICv4 Software Overview.

3.3.3 ITS AXI4-Stream interface

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

We expect that a typical distributed system is 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.

3.3.4 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 the qactive signal.

The qreqn input signal is synchronized to the ITS.

Related information

Power control signals on page 251

3.3.5 ITS configuration

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

Table 3-9: Configurable options for the ITS

Feature	Range of options
DeviceID width	3-20
EventID width	1-16
CollectionID width	2-14
Inclusion of a bypass port	True or False
MSI-64 support, which controls whether the DeviceID is sent using the awuser signals or on bits[63:32] that are written to GITS_TRANSLATER. See 4.12 MSI-64 on page 65.	True or False
The number of credits for supporting transfer of LPIs using non-locked translations to the Distributor.	1-16
ACE-Lite subordinate interface address width	20-48
ACE-Lite subordinate interface data width	64, 128, 256
ACE-Lite subordinate interface read ID width	1-32
ACE-Lite subordinate interface write ID width	1-32
AXI4-Stream data width	16, 64
ECC support for the caches. For more information, see 4.15 Reliability, Accessibility, and Serviceability on page 68.	True or False
Collection cache depth, or cache entries ÷ 2	2, 4, 8, 16, 32, 64, 128, 256, 512
Device cache depth, or cache entries ÷ 2	2, 4, 8, 16, 32, 64, 128
Event cache depth, or cache entries ÷ 2. The number of Device and EventID pairs that are cached in the ITS.	2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048
Domain name. For more information, see Figure 3-8: GIC-600AE top-level structure options on page 47.	Any legal domain identifier

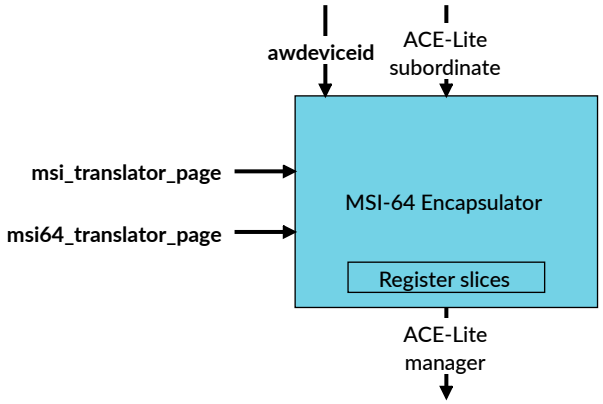
For more information, see the *Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Configuration and Integration Manual*.

3.4 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 3-5: MSI-64 Encapsulator



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

See 4.12 MSI-64 on page 65 for more information.

3.4.1 MSI-64 ACE-Lite interfaces

The MSI-64 Encapsulator has an ACE-Lite subordinate interface and an ACE-Lite manager interface.

MSI-64 ACE-Lite subordinate interface with awdeviceid

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

MSI-64 ACE-Lite manager interface

This interface is a full ACE-Lite manager port.

The following table shows the transaction acceptance capabilities of both subordinate and manager ports.

Table 3-10: Transaction acceptance

Transaction type	Maximum number of transactions allowed
Read	Unlimited
Write	Unlimited
Combined	Unlimited

Any leading wdata signal is registered and held until the awaddr signal arrives. These signals are described in A.5 ACE-Lite interface signals on page 254.


Note

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

3.4.2 MSI-64 Encapsulator configuration

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

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

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

Table 3-11: Configurable options for the MSI-64 Encapsulator

RTL parameter	Function	Range of options
`DATA_WIDTH`	Specifies the width of rdata and wdata data signals	64, 128, 256
`ADDR_WIDTH`	Specifies the width of araddr and awaddr address signals	17-48
`AWUSER_WIDTH`	Specifies the width of awuser signal	1-128
`ARUSER_WIDTH`	Specifies the width of aruser signal	1-128
`RUSER_WIDTH`	Specifies the width of ruser signal	1-128
`WUSER_WIDTH`	Specifies the width of wuser signal	1-128
`BUSER_WIDTH`	Specifies the width of buser signal	1-128
`DID_WIDTH`	Specifies the width of the DeviceID	3-20
`WID_WIDTH`	Specifies the width of wid signal	1-32
`RID_WIDTH`	Specifies the width of rid signal	1-32
`FWD_REG_TYPE`	Register slice type on forward AW, AR, and W channels	**0** None **1** Reverse **2** Forward **3** Full

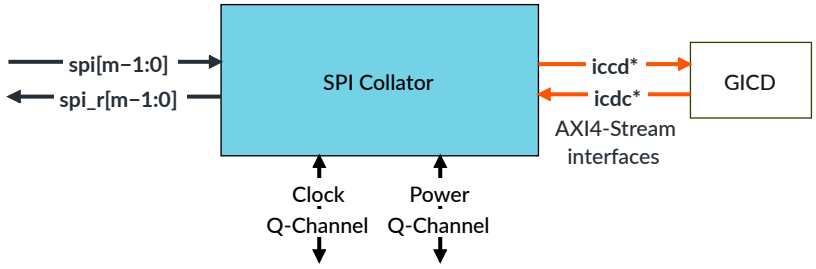
RTL parameter	Function	Range of options
REV_REG_TYPE	Register slice type on B and R channels	**0** None **1** Reverse **2** Forward **3** Full

3.5 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 3-6: 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 Distributor can be clock gated hierarchically.

3.5.1 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.

3.5.2 SPI Collator wires

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

By default, the asserted level of an SPI is active-HIGH, as with previous Arm GIC implementations. However, each SPI can be either inverted, synchronized, or both, using the parameters `SPI_INV[n]` and `SPI_SYNC[n]`, where:

- `SPI_INV[n]` == 1 indicates that the inverter is enabled
- `SPI_SYNC[n]` == 1 indicates that the synchronizer is enabled

- `[n]` = SPI_ID – 32

Each SPI input wire has a corresponding spi_r wire after the synchronizer or capture flop that can be used to create pulse extenders for edge-triggered interrupts that cross clock domains. If `SPI_INV[n]` is set to 1, then the wire after the synchronizer is inverted with respect to the input.

3.5.3 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 the qactive_col signal is LOW, it indicates that all SPIs to the SPI Collator are in their idle state of either 0 (active-HIGH) or 1 (active-LOW), so all messages are sent to the Distributor.

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

If the qreqn_col signal 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 4.6 Power management on page 56 before the Distributor can be powered down.

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 SPI Collator clock if all edge-triggered interrupts into the SPI Collator are pulse extended so that edges are not missed.

3.5.4 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 the qactive_col_clk signal 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 the qactive_col_clk signal is HIGH, the SPI Collator rejects any attempt to enter a low-power mode.

If the qreqn_col_clk signal 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, the qactive_col_clk signal is asserted.

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

3.5.5 SPI Collator configuration

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

Table 3-12: Configurable options for the SPI Collator

Feature	Range of options
The number of SPI wires	0-960, in multiples of 32
SPI_INV is a wide vector of one bit for each SPI, indicating whether to invert the interrupt	True, False
SPI_SYNC is a wide vector of one bit for each SPI, indicating whether to synchronize the interrupt	True, False

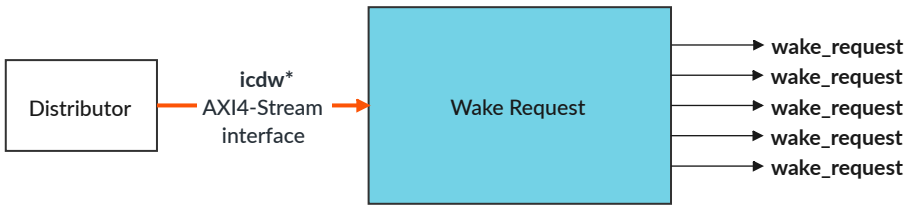
For more information, see the *Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Configuration and Integration Manual*.

3.6 Wake Request

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

The following figure shows the Wake Request block.

Figure 3-7: Wake Request



A wake_request signal wakes a powered-down core when one of the following conditions is true:

- An interrupt that targets only that specific core is pending
- GICD_CTLR.E1NWF is set, and a 1-of-N SPI has selected that core as its target.

The GIC-600AE does not know whether a core is powered up or down. It only knows whether software has enabled sending transactions on the AXI4-Stream interface. Therefore, a wake_request signal remains asserted after a core has powered up. A wake_request signal deasserts when software clears GICR_WAKER.ProcessorSleep and the GIC-600AE clears the GICR_WAKER.ChildrenAsleep bit.

If there are pending interrupts, either targeted or 1-of-N when GICR_WAKER.ProcessorSleep is set, the wake_request signal might assert during the powerdown sequence. The power controller must ignore the wake_request signal until the core is powered down.

Each wake_request signal is protected with odd parity. The parity signals are wake_request_chk[<cpus>−1:0].

The level of the asserted wake_request[<cpus>−1:0] 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.

3.6.1 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.

3.6.2 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.

3.7 Interconnect

The GIC-600AE 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-600AE provides an AXI4-Stream interconnect for transporting messages. However, messages can be sent over an existing interconnect provided the interconnect is free-flowing.

3.7.1 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* (CDCs).

The Arm internal scripts limit a single interconnect crossbar to 16 destinations. To work around this limitation, you can use domains in the config file. For example, instead of 32 Redistributors in one domain, you can use two domains that each contain 16.

3.8 Hierarchy

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

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

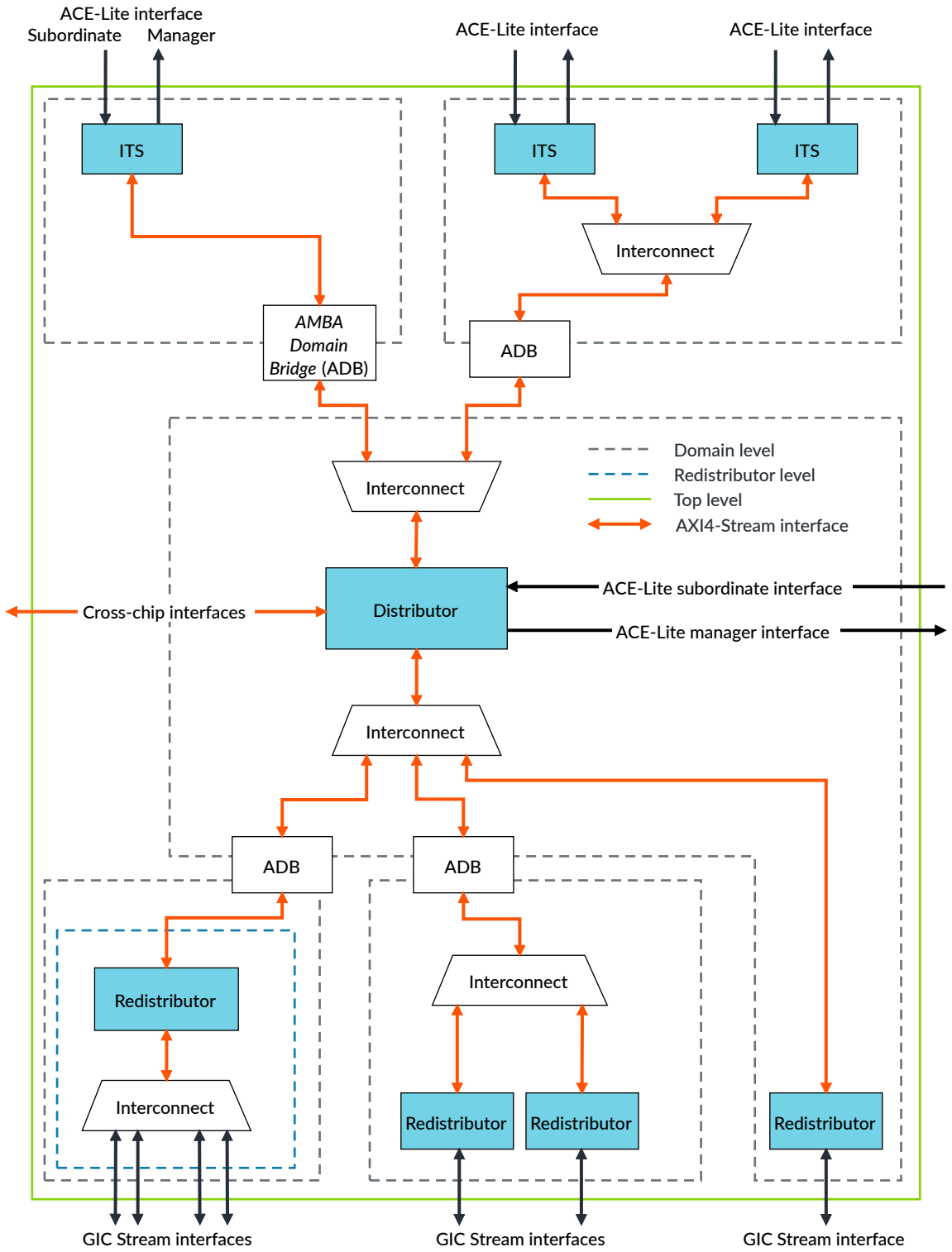
- The Redistributor is wrapped with interconnect components between the Redistributor and the cores. The components that are wrapped at this level are shown within the blue dashed lines in the following figure. If the core is in a different clock domain, in accordance with the domain tags, then half of the CoreLink™ ADB-400 domain bridge is included in a stitched file that is named `gic600ae_ppi_wrap_<n>_<usrcfg>.v`.
- If a bypass switch is selected as shown in Figure 3-3: ITS block on page 34, the ITS block is wrapped in a file that is named `gic600ae_its_wrap_<n>_<usrcfg>.v`.
- If the GIC-600AE is configured to share ACE-Lite ports between the ITS and GICD (configuration parameter `monolithic==1`), the ITS and GICD are stitched together in a file that is named `gic600ae_gicd_wrap_<usrcfg>.v`.

domain All blocks and wrapped components that are in the same domain are stitched together in a file that is named `gic600ae_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-600AE file called `gic600ae_<usrcfg>.v`.

The following figure shows the top-level options.

Figure 3-8: GIC-600AE top-level structure options



4. Operation

This chapter provides an operational description of the GIC-600AE.

4.1 Interrupt types

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

4.1.1 SGIs

Software Generated Interrupts (SGIs) 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. SGIs are edge triggered.

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

4.1.2 PPIs

A *Private Peripheral Interrupt* (PPI) 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 connect 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 ID16-ID31 range.

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 and version 4 for more information.

The GIC-600AE provides an option, through parameters, to include one or both a synchronizer and inverter on each PPI interrupt signal. See 3.2.4 Redistributor PPI signals on page 33 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.

4.1.3 SPIs

A *Shared Peripheral Interrupt* (SPI) is generated by a peripheral that is accessible across the whole system such as a USB receiver, and which can connect 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 an SPI on one core activates the SPI for all cores. That is, the GIC-600AE allows at most one core to activate an 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 subordinate programming interface. The GIC-600AE can support up to 960 SPIs, corresponding to the external spi input signals on the SPI Collator. The number of SPIs available depends on the implemented configuration. The permitted values are ID32-ID991, in steps of 32. The first SPI has an ID number of 32.

During configuration of the GIC, you can apportion some or all SPIs to be message-based or you can set all SPIs to be physical spi signals. If an SPI ID is allocated as a physical spi input signal, then software can still use that SPI ID as a message-based SPI, provided that the hardware ensures that the spi signal is held to a logic level that represents the inactive state.

You can configure whether each SPI is triggered on a rising edge or is active-HIGH level-sensitive. The GIC-600AE provides an option, through a parameter, to include one or both of a synchronizer or inverter for each SPI interrupt wire.

The GIC-600AE 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 3.5 SPI Collator on page 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 trigger a valid SPI by using the GICD_SETSPI_NSR or GICD_SETSPI_SR registers, see the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.

4.1.4 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.

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-600AE to generate and control LPIs. For more information, see GICR_SETLPIR register in the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.

4.1.5 Choosing between LPIs and SPIs

Message-based interrupts can be either LPIs or SPIs.

The decision by software to use an LPI or SPI for an interrupt, depends on whether there are message-based SPIs available and if the GIC-600AE has LPI support. The allocation of message-based SPIs is set during the GIC configuration process. Also, if the hardware ensures that an spi signal is held to a logic level that represents the inactive state, then software can use that SPI ID as a message-based SPI.

The interrupt type can be selected by either making the peripheral write to a different GIC-600AE 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 ITS 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 a virtual machine owns. This is because the hypervisor can let the virtual machine program the peripheral directly, and the ITS converts the virtual machine interrupt IDs 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-600AE attempts to automatically balance the interrupt load to cores that are active but not handling other interrupts.
- The GIC-600AE can provide more 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 difference is because SPIs have all their settings stored internally to the GIC-600AE, 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, we recommend using SPIs for any latency-sensitive interrupts that are expected to occur infrequently.

For more information, see the GICv3 and GICv4 Software Overview.

4.2 Interrupt groups and security

The GIC-600AE 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.

The following registers control to what group each interrupt is assigned:

- GICD_IGROUPRn
- GICD_IGRPMODRn
- GICR_IGROUPR0
- GICR_IGRPMODR0

The groups are:

- 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 core interfaces.

The GIC-600AE supports the three interrupt groups that the following table shows.

Table 4-1: 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 4-2: 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

The `ds_value` configuration parameter controls the GIC-600AE security, as the GIC exits reset.

0	Security enabled (fixed)
1	Security disabled (fixed)
P	Security is programmable by software during the boot sequence using GICD_CTLR.DS.

Setting the *Disable Security* (DS) bit to 1 in the GICD_CTLR register removes the security support of the GIC-600AE. It can be set by Secure software during the boot sequence or configured to be always set when you configure the design using the `ds_value` parameter. 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 and version 4.


Note

We recommend 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 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-600AE 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. When set, only a hardware reset can clear the DSL bit.

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

For more information, see the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4 and the GICv3 and GICv4 Software Overview.

4.3 Physical interrupt signals (PPIs and SPIs)

The GIC-600AE 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 4.1 Interrupt types on page 48 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-600AE 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 3.5.2 SPI Collator wires on page 42.

For more information, see the GICv3 and GICv4 Software Overview and the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.

4.4 Affinity routing and assignment

The GIC-600AE 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.

The GIC-600AE 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-bit fields 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 connects to the interrupt controller.

The GIC-600AE allows fully flexible allocation of MPIDR. However, it has two built-in default assignments that are based on the `aff0_thread` configuration parameter:

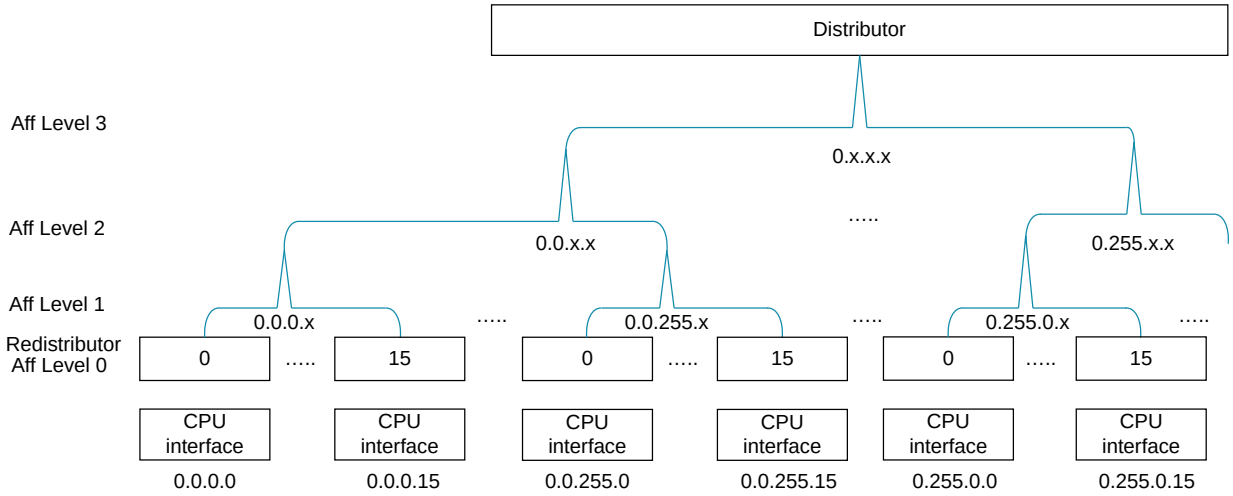
`aff0_thread == 1`	The four fields map to 0.<cluster>.<core>.<thread>

aff0_thread == 0	The four fields map to 0.0.<cluster>.<core>

See the *Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Configuration and Integration Manual* for information about the `aff0_thread` configuration parameter and how to build affinity schemes that include heterogenous clusters and multithreaded cores.

The following figure shows the affinity hierarchical structure.

Figure 4-1: 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 GICv3 and GICv4 Software Overview and the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.

4.5 SPI routing and 1 of N selection

The GIC-600AE supports 1 of N selection of SPI interrupts. You can program an SPI to target several cores, and the GIC-600AE can select which cores receive an SPI.

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

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

The GIC-600AE only sends an SPI to cores that are powered up and have the relevant interrupt group enabled. The GIC-600AE prioritizes cores that are considered active, but if there are no active cores, it selects inactive cores.

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

cpu_active signal

A cpu_active signal is an input to a Redistributor that corresponds to a particular core. When a cpu_active signal is LOW, 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 an SPI if there are no other options possible.
Ideally, the cores that are in retention are not woken without explicit software intervention, so that cores spend more time in retention. To ensure that this behavior is usually the case, use the following guidelines:

- Cores in retention must drive their corresponding cpu_active signal LOW.
- Powered-up cores that are not in retention must drive their cpu_active signal HIGH.

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

- When a core is powered down, the value of its cpu_active signal is irrelevant. This irrelevancy is because the software programming requirements for the GIC ensure that it knows when cores are powered up or down.
- The cpu_active signal 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.

Processor and GICD group enables and GICR_WAKER.ProcessorSleep

A 1 of N interrupt is not sent to a core if one of the following is true:

- The core is asleep, as indicated by GICR_WAKER.ProcessorSleep.
- The interrupt group is disabled by either the processor or the GICD_CTLR group enables.

Interrupt class

This is an implementation-defined feature that the GIC-600AE provides. Each core can be assigned to either class 0 or class 1 by writing to the relevant GICR_CLASSR register. An 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 an Arm® 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.E1NWF bit controls whether the GIC-600AE 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. Therefore, the GIC uses the GICR_CTLR.DPG and GICR_CLASSR.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.

We strongly recommend that if you use GICD_CTLR.E1NWF, you must also set the GICR_CTLR.DPGx bits 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-600AE 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 its cpu_active signal 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.

4.6 Power management

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

4.6.1 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-600AE 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)
```

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

4.6.2 Processor core power management

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

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-600AE receives an interrupt that targets only that core, the Wake Request block asserts the wake_request signal that corresponds to that core. The wake_request signal must connect to the system power controller. See 3.6 Wake Request on page 44.

You must not set the GICR_WAKER.ProcessorSleep bit to 1, unless the core enters a power state where the GIC-600AE uses the power controller to wake the core instead of the GIC Stream interface. For example, with Arm® Cortex®-A53 and Cortex®-A57 processors, if a 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 A-profile architecture. The system integrator can use a 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.

Interrupts that target only one core are unaffected by the cpu_active signal 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 and version 4 for information about power management, and about wakeup signals and their relation to the core outputs.

4.6.3 Other power management

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

When powering down the GIC-600AE, software must preserve the state of the GIC-600AE, 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 and version 4.

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.

- 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.


Note

- 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.

We recommend 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-600AE without deadlock.

When the GICR_WAKER.Quiescent bit is set, it is safe to power down the GIC-600AE 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-600AE if any interrupts arrive that must not be ignored.

When the GIC-600AE 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, we recommend that you speed up the initialization of the GIC-600AE as follows:

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


Note

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

For more information, see the GICv3 and GICv4 Software Overview.

4.7 Getting started

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

Each Redistributor must be powered on using its GICR_PWRR register to enable the Redistributors to be accessed, see 4.6.1 Redistributor power management on page 56 for more information.

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

4.8 Backwards compatibility

The GIC-600AE does not support legacy operation.

Legacy operation is indicated by GICD_CTLR.ARE_S or GICD_CTLR.ARE_NS == 0.

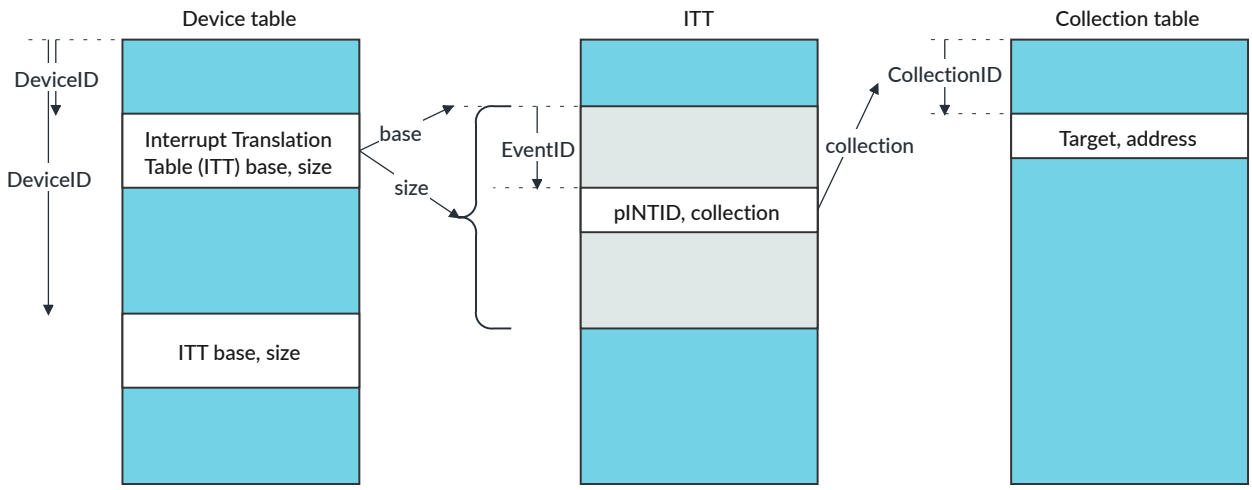
Therefore, SGIs and PPIs can be programmed only through the GICR register space, and SGIs are not banked by the source core.

4.9 ITS

The GIC-600AE supports up to 16 *Interrupt Translation Service* (ITS) blocks in the system with a limit of eight 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 4-2: ITS process



To reduce memory traffic and keep interrupt latency to a minimum, GIC-600AE 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 that is implemented as an array of flops, and to reduce RAM area, you can remove ECC from each RAM individually. See the *Arm® CoreLink™ GIC-600AE 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-600AE 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 and version 4 for more information.

The GIC-600AE 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, software must pre-clear these tables to 0, apart from pointers to cleared level-two Device tables, unless the tables were previously populated by GIC-600AE.

The GIC-600AE ITS supports all GICv3 commands that the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4 describes.

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 4.15 Reliability, Accessibility, and Serviceability on page 68.

For information about how to program and use the ITS, see the GICv3 and GICv4 Software Overview.

4.9.1 ITS cache control, locking, and test

The GIC-600AE can lock certain interrupt translations in the EventID cache.

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

The GIC-600AE can lock certain translations into the ITS cache, 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, when software provides the DEVICE_ID, EVENT_ID, and the correct GITS_OPR.LOCK_TYPE (ITS lock = 2). The GIC attempts to perform the lock, and reports the status in GITS_OPSR. If the lock succeeds, GITS_OPSR.REQUEST_COMPLETE == 1 and GITS_OPSR.REQUEST_PASS == 1.

Each cache set is 2-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_OPR register to unlock entries that are locked.

The GITS_OPR 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 translation data in GITS_OPSR. The GIC also reports whether 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 the GIC reports 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, we recommend that the GITS_OPR value is read after writing. You can abort Track operation by writing GITS_OPR.LOCK_TYPE == Track abort.

4.9.2 ITS commands and errors

Each ITS detects a wide range of command errors and translation errors, and reports them in Armv8.2 RAS-compliant error records in the Distributor.

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 4-3: ITS record error syndrome groups

Group	Control
ACE-Lite subordinate interface write translation errors. Only when the ITS has a separate ACE-Lite subordinate port.	GITS_FCTLR.AEE (Access Error Enable)
Translation errors on incoming writes to GITS_TRANSLATER	GITS_FCTLR.UEE (Unmapped Error Enable)

Group	Control
Errors during commands	GITS_FCTLR.CEE (Command Error Enable)
Other errors such as memory system, or memory allocation errors	None

See 4.15.6.7 ITS command and translation error records 13 and beyond on page 81 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 4.7 Getting started on page 59 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 that the GITS_CREADR identifies and then 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 completes, you can either:

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

If you add an `INT` command, then we recommend that you enable GITS_FCTLR.CEE and that you configure the fault handling interrupt or error recovery interrupt to be delivered to a core that can resolve Command queue issues. See 4.15.5 Error recovery and fault handling interrupts on page 70 for more information.

4.10 LPI caching

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

The LPI cache is 2-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-600AE 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 behavior 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. The GIC does not accept changes in the LPI Property table, until the relevant INV and SYNC commands are executed through an ITS, GICR_INVLPIR, or GICR_INVALLR.

The GIC-600AE 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 information

Distributor configuration on page 30

4.11 Memory access and attributes

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

We recommend that all tables are placed in Normal memory. All ITS tables are private, and after allocation, only the GIC accesses them. 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 signal mappings for the memory transactions that the GIC generates.

Table 4-4: Memory access registers

Access type	Register	Mapping control bit[1]
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 the following table shows.

Table 4-5: Cacheability values

Main Cacheability value (*BASER*.OuterCache)	Other Cacheability value (*BASER*.InnerCache)	arcache signal	awcache signal	arcache signal (DCC = 1)	awcache signal (DCC = 1)
0b000, Device-nGnRnE	-	0b0010	0b0010	0b0010	0b0010

[1] The mappings are designed for the Armv8 and Armv8.2 generation of cores. However, setting this bit converts the GIC-600AE to full featured mapping.

Main Cacheability value (*BASER*.OuterCache)	Other Cacheability value (*BASER*.InnerCache)	arcache signal	awcache signal	arcache signal (DCC = 1)	awcache signal (DCC = 1)
0b001, Normal Non-cacheable	Match	0b0011	0b0011	0b0011	0b0011
	No match	0b0011	0b0011	0b0011	0b0011
0b010, Normal Cacheable RA Write-Through	Match	0b0011	0b0011	0b1110	0b0110
	No match	0b0011	0b0011	0b1110	0b0110
0b011, Normal Cacheable RA Write-Back	Match	0b1111	0b0111	0b1111	0b0111
	No match	0b0011	0b0011	0b1111	0b0111
0b100, Normal Cacheable WA Write-Through	Match	0b0011	0b0011	0b1010	0b1110
	No match	0b0011	0b0011	0b1010	0b1110
0b101, Normal Cacheable WA Write-Back	Match	0b1011	0b1111	0b1011	0b1111
	No match	0b0011	0b0011	0b1011	0b1111
0b110, Normal Cacheable WA RA Write-Through	Match	0b0011	0b0011	0b1110	0b1110
	No match	0b0011	0b0011	0b1110	0b1110
0b111, Normal Cacheable WA RA Write-Back	Match	0b1111	0b1111	0b1111	0b1111
	No match	0b0011	0b0011	0b1111	0b1111

The a<x>domain signal 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 AMBA® AXI and ACE Protocol Specification.

4.12 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-600AE ITS supports two mechanisms:

awuser_*_s signal

The DeviceID arrives on sideband User signals. You 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, we recommend that the GITS_TRANSLATER register is moved to an arbitrary page that is protected by the hypervisor.

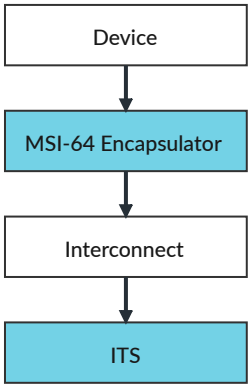
The GIC-600AE uses two methods to support this:

- The MSI-64 Encapsulator modifies the page address of accesses to the architectural GITS_TRANSLATER address, set by the msi_translator_page tie-off signal, to the system-defined page set by the msi64_translator_page signal.
- When the ITS shares an ACE-Lite subordinate port, the its_transr_page_offset tie-off signal allows the GITS_TRANSLATER register page to be moved to anywhere in the address map, to match the msi64_translator_page signal value that is independent of the GICD address map reset.
  The msi64_translator_page and its_transr_page_offset signals, 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 3.4 MSI-64 Encapsulator on page 40 for more information.

  The following figure shows an example of how to integrate the MSI-64 Encapsulator in a system. The MSI-64 Encapsulator connects upstream of the interconnect and targets an ITS downstream of the interconnect. In this scenario, the DeviceID is transported on the data channels of the interconnect to the ITS. This topology benefits those systems where the width of the awuser signal on the interconnect is too narrow to transport the DeviceID.

Figure 4-3: MSI-64 Encapsulator with DeviceID sent in the data[63:32] bits



4.13 RAMs and ECC

The GIC-600AE uses multiple RAMs to store a range of states for all types of interrupt. In typical operation, the RAMs are transparent to software.

Each 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 4.15 Reliability, Accessibility, and Serviceability on page 68 for more information.

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

4.14 Performance Monitoring Unit

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


Note

The PMU does not track Redistributor events. Software can count the delivery of PPI and SGI interrupts by recording calls to the core interrupt service routine.

The GIC events are described in Table 5-60: GICP_EVTYPERn.EVENT field encoding on page 165.

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

Secure and Non-secure interrupts are counted together, so Non-secure software cannot, by default, access the GICP (PMU) register space. However, Secure software can decide to allow access. Non-secure software can be given access to the GICP (PMU) register space by either:

- Software programming the GICD_SAC.GICPNS bit to 1
- Setting the gicp_allow_ns tie-off signal HIGH, during silicon integration

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 value 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

Software can enable the overflow interrupt on a per counter basis by setting the relevant bit of GICP_INTENSET0. For example, bit[0] enables GICP_EVCNTR0 and bit[1] enables GICP_EVCNTR1. Similarly, software can disable the overflow interrupt enable 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 registers can be used for save and restore operations and for testing the correct integration of the pmu_int interrupt signal.

The pmu_int signal 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 signal SPI is delivered internally in accordance with normal SPI programming.

The GICP_IRQCR.SPIID field 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 or out of range, or for multichip configurations, not owned on chip, the register updates to 0 and no internal delivery occurs.

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

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

4.15 Reliability, Accessibility, and Serviceability

The GIC-600AE uses a range of RAS features for all RAMs, which include *Single Error Correction and Double Error Detection* (SECDED), and Scrub, software and bus error reporting.

The GIC makes all necessary information available to software through Armv8.2 RAS architecture-compliant register space.

4.15.1 Non-secure access

You can control whether Non-secure software has access to the RAS architecture-compliant register space by using GICD_SAC.GICTNS. The gict_allow_ns tie-off signal sets the reset value of the GICTNS bit.

If there is an error, and if 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.

4.15.2 Scrub

The GIC-600AE holds significant programming and interrupt states in RAM, which is protected by *Single Error Correction and Double Error Detection* (SECDED).

However, some RAM content might be static for a long duration, and there is a potential for errors to accumulate if a particular address is not periodically accessed. To prevent this occurring, software can periodically trigger a low-priority scrub of a RAM, by setting the GITS_FCTLR.SIP, GICR_FCTLR.SIP, and GICD_FCTLR.SIP bits. 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.

4.15.3 Error record classification

The GIC reports errors in Armv8.2 RAS architecture-compliant error records, which are accessible through the ACE-Lite subordinate programming interface.

The classes of error records are:

- 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.

4.15.4 ECC error reporting and recovery

When an ECC error is detected, the GIC-600AE 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 4-6: 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_ICERRRn register. The corrupted RAM contents can be read until the error is cleared by writing to GICD_ICERRRn. SPIs that are in the error state can also be determined by reading the GICD_ICERRRn 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. **Note:** Repeated double errors in the LPI cache cause an overflow of the error record, which means subsequent information is lost. We recommend 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. You can determine the interrupts that are in error by reading the GICR_IERRVR register. **Note:** Because the group is unknown, it is assumed to be Secure, and so interrupt deactivates can be ignored. Software must consider this as part of the recovery sequence. It is also possible for a GenerateSGI packet to become corrupted. In this case, the GenerateSGI is reported as bad. For more information about Pending and Active PPI states, see the 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 managers from raising Secure interrupts incorrectly.

4.15.5 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, so the interrupt only fires when the counter in the associated GICT_ERR<n>MISC0 register overflows. You can preset the counter to any value by writing to GICT_ERR<n>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 signal, by setting GICT_ERR<n>CTLR.FI, or to the error recovery interrupt, err_int signal, by setting GICT_ERR<n>CTLR.UI. The interrupt fires on every uncorrectable interrupt occurrence irrespective of the counter value.

You can route the fault_int and err_int signals 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 not owned on chip for multichip configurations, the register updates to 0 and no internal delivery occurs.

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

The err_int and fault_int signals 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 nonexistent register.

4.15.6 Error handling records

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

The following table lists the GIC-600AE error handling records. The Type column uses the following acronyms:

CE Correctable error
UEO Restartable error and contained
UER Recoverable error

Table 4-7: Error handling records

Record	Description	Type	Description, events, and recovery sequences
0	Software error in GICD programming	UEO	Table 4-8: Software errors, record 0 on page 72
1	Correctable SPI RAM errors	CE	Table 4-9: SPI RAM errors, records 1-2 on page 76. GICT_ERR<n>STATUS.SERR == 7, data value from associative memory.
2	Uncorrectable SPI RAM errors	UER	
3	Correctable SGI RAM errors	CE	Table 4-10: SGI RAM errors, records 3-4 on page 78. GICT_ERR<n>STATUS.SERR == 7, control value from associative memory.
4	Uncorrectable SGI RAM errors	UER	
5	Reserved	-	-
6	Reserved	-	-
7	Correctable PPI RAM errors	CE	Table 4-11: PPI RAM errors, records 7-8 on page 79. GICT_ERR<n>STATUS.SERR == 7, control value from associative memory.
8	Uncorrectable PPI RAM errors	UER	
9	Correctable LPI RAM errors	CE	Table 4-12: LPI RAM errors, records 9-10 on page 80. GICT_ERR<n>STATUS.SERR == 7, control value from associative memory. Records 9-10 are not present if there is no LPI support.
10	Uncorrectable LPI RAM errors	UER	
11	Correctable error from ITS RAM	CE	Table 4-13: ITS RAM errors, records 11-12 on page 80. GICT_ERR<n>STATUS.SERR == 6, data value from associative memory. Records 11-12 are not present if an ITS is not present.
12	Uncorrectable error from ITS RAM	UEO	
13+	ITS command and translation errors	UER	4.15.6.7 ITS command and translation error records 13 and beyond on page 81. GICT_ERR<n>STATUS.SERR == 14, illegal access. One record per ITS on the chip. Records 13+ are not present if an ITS is not present.

4.15.6.1 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-600AE. 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.

We recommend that record 0 is connected to a high priority interrupt. This connection prevents the record from overflowing if it receives more errors than it is able to process with the possible loss of information that is required for recovery. See 4.15.5 Error recovery and fault handling interrupts on page 70 for more information.

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

Table 4-8: Software errors, record 0

GICT_ERR<n>STATUS.IERR (Syndrome)	GICT_ERR<n>STATUS .SERR	GICT_ERR<n>MISC0. Data description (other bits RES0) Always packed from 0 (lowest = 0)	Recovery, prevention
0x0, SYN_ACE_BAD Illegal ACE-Lite subordinate access.	0xE	AccessRnW, bit[12] AccessSparse, bit[11] AccessSize, bits[10:8] AccessLength, bits[7:0]	Repeat illegal access, with appropriate size and properties. Full access address is given in GICT_ERR0ADDR.
0x1, SYN_PPI_PWRDWN Attempt to access a powered down Redistributor.	0xF	Redistributor, bits[24:16] Core, bits[8:0]	Ensure that the Redistributor is powered up before accessing. See GICR_PWRR. Attempt was made by the core reported in MISC0.
0x2, SYN_PPI_PWRCHANGE Attempt to power down Redistributor rejected.	0xF	Redistributor, bits[24:16] 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 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 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. When 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. See 4.6.3 Other power management on page 58.

GICT_ERR<n>STATUS.IERR (Syndrome)	GICT_ERR<n>STATUS .SERR	GICT_ERR<n>MISC0. Data description (other bits RES0) Always packed from 0 (lowest = 0)	Recovery, prevention
0x5, SYN_PENDBASE_ACC Attempt to reprogram PENDBASE registers to a value that is not accepted because another value is already in use.	0xF	Core, bits[8:0]	When 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. See 4.6.3 Other power management on page 58. Otherwise, repeat the register access using the current global values.
0x6, SYN_LPI_CLR Attempt to reprogram ENABLE_LPI when not enabled and not asleep.	0xF	Core, bits[8:0]	We recommend 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. See 4.6.3 Other power management on page 58.
0x7, SYN_WAKER_CHANGE Attempt to change GICR_WAKER abandoned due to handshake rules.	0xF	Core, bits[8:0]	GICR_WAKER.ProcessorSleep and GICR_WAKER.ChildrenAsleep form a 4-phase handshake. The attempt to change state must be repeated when the previous transition has completed.
0x8, SYN_SLEEP_FAIL 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 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 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.
0x10, SYN_SGI_NO_TGT 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. If this level of RAS functionality is required, the software must track generated SGIs externally.
0x11, SYN_SGI_CORRUPTED 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 error, it must use an external record of the generated SGI.
0x12, SYN_GICR_CORRUPTED 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 that is stored in SGI RAM or PPI RAM. Check records 4 and 8, and perform a recovery sequence for those interrupts.

GICT_ERR<n>STATUS.IERR (Syndrome)	GICT_ERR<n>STATUS .SERR	GICT_ERR<n>MISC0. Data description (other bits RES0) Always packed from 0 (lowest = 0)	Recovery, prevention
0x13, SYN_GICD_CORRUPTED Data was read from GICD register space that encountered an uncorrectable error.	0x6	GICT_ERR0ADDR is populated	Software has tried to read corrupted data that is stored in SPI RAM. Check record 2 and perform a recovery sequence for those interrupts.
0x14, SYN_ITS_OFF Data was read from an ITS that is powered down.	0xF	GICT_ERR0ADDR is populated	Ensure that the qreqn_its<x> signal power control Q-Channel is in the RUN state before accessing the relevant ITS
0x18, SYN_SPI_BLOCK Attempt to access an SPI block that is not implemented.	0xE	Block, bits[4:0]	No recovery is required. Correct the software.
0x19, SYN_SPI_OOR Attempt to access a non-implemented SPI using (SET|CLR)SPI.	0xE	ID, bits[9:0]	Reprogram the issuing device so that it sends a supported SPI ID
0x1A, SYN_SPI_NO_DEST_TGT An SPI has no legal target destinations.	0xF	ID, bits[9:0]	Before enabling the specified SPI, reprogram the SPI to target an existing core. The same SPI might repeat this error several times and cause an overflow.
0x1B, SYN_SPI_NO_DEST_1OFN A 1 of N SPI cannot be delivered due to bad GICR_CTRL.DPG<0|1NS|1S> or GICR_CLASSR 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. The same SPI might repeat this error several times and cause an overflow.
0x1C, SYN_COL_OOR 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 A `Deactivate` command to a nonexistent SPI, or with incorrect groups set. `Deactivate` commands to LPI and nonexistent PPI are not reported.	0xE	None	A `Deactivate` command occurred to a nonexistent SPI, or that SPI group prevents the deactivate occurring. Software must check the active states of SPIs.
0x1E, SYN_SPI_CHIP_OFFLINE An attempt was made to send an 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 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] Data, bits[15:0]	Software must reprogram the source device to only create legal LPI IDs
0x29, SYN_ITS_REG_CLR _OOR An attempt was made to clear an OOR interrupt. Only valid when GICR LPI injection registers are supported.	0xE	Core, bits[24:16] Data, bits[15:0]	Software must not attempt to clear nonexistent LPIs

GICT_ERR<n>STATUS.IERR (Syndrome)	GICT_ERR<n>STATUS .SERR	GICT_ERR<n>MISC0. Data description (other bits RES0) Always packed from 0 (lowest = 0)	Recovery, prevention
0x2A, SYN_ITS_REG_INV_OOR An attempt was made to invalidate an OOR interrupt. Only valid when GICR LPI injection registers are supported.	0xE	Core, bits[24:16] Data, bits[15:0]	Software must not attempt to clear nonexistent LPIs
0x2B, SYN_ITS_REG_SET_ENB 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] 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 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] 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 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] 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 An attempt was made to read properties for a single interrupt, where an error response was received with the data.	0x12	Target, bits[29:16] 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 An attempt was made to read properties for a block of interrupts, where an error response was received with the data.	0x12	Target, bits[29:16] 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.
0x42, SYN_PT_COARSE_MAP_READ_FAIL An attempt was made to read the coarse map for a target, where an error response was received with the data.	0x12	Target, bits[29:16]	No recovery is necessary because the GIC assumes that the coarse map is full
0x43, SYN_PT_COARSE_MAP_WRITE_FAIL An attempt was made to write the coarse map for a target, with an error received with the write response.	0x12	Target, bits[29:16]	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 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[29:16] ID, bits[15:0]	Software must determine the reason for the pending error read fail. The GIC uses the data that is supplied, however, it is possible for the LPI interrupt to be lost around the specified LPI.

GICT_ERR<n>STATUS.IERR (Syndrome)	GICT_ERR<n>STATUS .SERR	GICT_ERR<n>MISC0. Data description (other bits RES0) Always packed from 0 (lowest = 0)	Recovery, prevention
0x45, SYN_PT_TABLE_WRITE_FAIL 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[29:16] 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 An attempt was made to read a subblock of interrupts from a Pending table, where an error response was received with the data.	0x12	Target, bits[29:16] ID, bits[15:0]	Software must determine the reason for the pending error read fail. The GIC uses the data that is supplied, however, it is possible for the LPI interrupt to be lost around the specified LPI.
0x47, SYN_PT_TABLE_WRITE_FAIL_BYTE 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[29:16] ID, bits[15:0]	The GIC tries to continue, however, this error indicates issues with the memory system, and operation might be unpredictable

4.15.6.2 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.

The GIC-600AE 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 4.15.5 Error recovery and fault handling interrupts on page 70 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, and the RAM has repair capability.

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

Table 4-9: 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 an SPI has an uncorrectable error, GICD_ICERRRn 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_ICERRRn registers, so that you can identify the SPIs that have errors. The GICD_ICERRRn registers must be read from the Secure side.
   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, if an overflow occurs, all GICD_ICERRRn registers must be checked.
4. If necessary, 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. We recommend 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 the GICD_ICERRRn bits that step 3 on page 77 indicates are showing an SPI error. This write clears the interrupt error and reverts the corresponding GICD_IGROUPRn, GICD_IGRPMODRn, GICD_ICFGRn, and GICD_NSACRn bits to their default values.
7. Read GICD_ICERRRn, to check that the error has cleared. If the error remains, then clear all the GICD_CTLR group enables so that it forces all SPIs to return to their owner chips. When GICD_CTLR.RWP returns to 0, repeat the write to GICD_ICERRRn. When the error clear is accepted, you can re-enable the group enables.
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, we recommend 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 on page 77.

4.15.6.3 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 4.15.5 Error recovery and fault handling interrupts on page 70 for information about the error counters and interrupt generation options.

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

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

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

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

The RAM stores information for the same SGI for up to 16 cores on a single row. The corrupted SGI number is given by:

- address MOD 16 on cores (address − (address MOD 16)) to (address − (address MOD 16)) + 15

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

When an SGI is in error, the GIC operates using the values that GICR_SGIDR contains.

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, so that you can identify the SGIs or PPIs that have errors. The GICR_IERRVR registers must be read from the Secure side.
4. If necessary, 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. We recommend that intended programming is stored in memory so that this step is not required. The GICR_NSACR is overwritten when an error occurs, so the pre-error value cannot be read back at this stage.
5. Write to GICR_ICENABLER0, to disable all interrupts that have errors.
6. Write 1 to the GICR_IERRVR bits that step 3 indicates are showing an SGI or PPI error. This write clears the interrupt error and reverts the corresponding GICR_IGROUPR0, GICR_IGRPMODR0, and GICR_NSACR bits 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 the recovery sequence from step 2 on page 78.

4.15.6.4 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.

Error records 7-8 record the errors from PPI RAM that contain 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 4-11: PPI RAM errors, records 7-8

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

For uncorrectable errors, if:

Bit[7], SGI/Int == 0

Software must perform the recovery sequence that 4.15.6.3 SGI RAM error records 3-4 on page 77 describes.

Bit[7], SGI/Int == 1

The GIC did not generate the SGI because an error occurred during SGI generation. Although an SGI generation error occurs, the GIC continues to operate normally.

4.15.6.5 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 support is configured.

The LPI RAM is the main LPI cache and it stores the LPI properties and pending information.

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

Table 4-12: LPI RAM errors, records 9-10

Record	GICT_ERR<n>MISC0.Data
9 = Correctable	• Bit location, bits[14+] • Pending, bits[13:12]. These bits indicate if there were pending interrupts in the cache at the time of the corruption. • Reserved, bits[11:10]. • Address, bits[9:0]
10 = Uncorrectable	• Pending, bits[13:12] • Reserved, bits[11:10] • Address, bits[9:0]

When an uncorrectable error occurs, the data shown in the table is stored and the GICT_ERR<n>MISC1 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.

If required, software can use the data in the GICT_ERR<n>MISC1 register to check several interrupt sources, such as the corrupted INTID. This ID is never more than two bits away from the recorded ID.

4.15.6.6 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 register reports data for ITS RAM error records 11-12 shown in the following table.

Table 4-13: ITS RAM errors, records 11-12

Record	GICT_ERR<n>MISC0.Data
11 = Correctable	• Address, bits[31:x + 10] • Bit location, bits[x + 9:x + 2] • RAM, bits[x + 1:x] • ITS, bits[x − 1:0] Where x = $\log_2$(ITS)

Record	GICT_ERR<n>MISC0.Data
12 = Uncorrectable	• Address, bits[31:x + 3)] • RAM, bits[x + 2:x] • ITS, bits[x – 1:0] Where 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 4-14: 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

4.15.6.7 ITS command and translation error records 13 and beyond

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 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 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 4-15: ITS command and translation errors, records 13 and beyond

MAPD commands					
Error mnemonic	**Encoding**	**IERR**	**Stall**	**Mask**	**Description**
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 commands					
Error mnemonic	**Encoding**	**IERR**	**Stall**	**Mask**	**Description**
MAPC_COLLECTION_OOR	0x10903	0	1	CEE	A MAPC command has tried to map a CollectionID that is not supported. See GITS_TYPER.
MAPC_TGT_OOR	0x10920	1	1/0	CEE	A MAPC command has tried to map to a core that does not exist. If the core is within the maximum range that the ITS supports, the command stalls. If the command is detected in the destination Distributor, the command is ignored and the core is reported in GICT_ERR<n>MISC1. **Note:** If the value in GICT_ERR<n>MISC1 is 0, the location of the detected error is in the ITS. CEE applies to errors detected in the ITS only.
MAPC_CHIP_OFFLINE_OOR	0x10922	1	0	-	A MAPC command has targeted a core in an offline chip. Software must correct the mapping or take the target chip online.
MAPC_LPI_OFF	0x10923	1	0	-	A MAPC command has tried to map a collection to a core that does not have LPIs enabled. Software must correct the mapping, or it must first enable LPIs using GICR_CTLR.Enable_LPIs. The core is reported in GICT_ERR<n>MISC1.

MAPI commands					
Error mnemonic	**Encoding**	**IERR**	**Stall**	**Mask**	**Description**
MAPI_DEVICE_OOR	0x10B01	0	1	CEE	A MAPI has tried to map a DeviceID that is not supported. See GITS_BASER0, and for information about the supported range, see GITS_TYPER.
MAPI_COLLECTION_OOR	0x10B03	0	1	CEE	A MAPI has tried to map to a collection that is not supported. See GITS_BASER1, and for information about the supported range, see GITS_TYPER.
MAPI_UNMAPPED_DEVICE	0x10B04	0	1	CEE	A MAPI has tried to map an interrupt to a device that is not mapped
MAPI_ID_OOR	0x10B05	0	1	CEE	A MAPI has tried to map to an EventID size that is not supported. The size that is supported is reported in GITS_TYPER, but might be reduced depending on the MAPD command for the specified DeviceID.

MAPVI commands					
Error mnemonic	**Encoding**	**IERR**	**Stall**	**Mask**	**Description**
MAPVI_DEVICE_OOR	0x10A01	0	1	CEE	A MAPVI has tried to map a device supported by the ITS that is out-of-range. See GITS_BASER0, and for information about the supported range, see GITS_TYPER.
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. See GITS_BASER1, and for information about the supported range, see GITS_TYPER.
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 commands					
Error mnemonic	**Encoding**	**IERR**	**Stall**	**Mask**	**Description**
MAPVI_PHYSICALID_OOR	0x10A06	0	1	CEE	A MAPVI is received that has a physical ID outside the range supported. The supported range is >16-<8096 bits.

MOVI commands					
Error mnemonic	**Encoding**	**IERR**	**Stall**	**Mask**	**Description**
MOVI_DEVICE_OOR	0x10101	0	1	CEE	A MOVI has tried to map a device that is outside the range that the ITS supports. See GITS_BASER0, and for information about the supported range, see GITS_TYPER.
MOVI_COLLECTION_OOR	0x10103	0	1	CEE	A MOVI has tried to use a collection that is outside the range that the ITS supports. See GITS_BASER1, and for information about the supported range, see GITS_TYPER.
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
MOVI_UNMAPPED_COLLECTION	0x10109	0	1	CEE	A MOVI command has tried to operate on a collection that is not mapped

MOVALL commands					
Error mnemonic	**Encoding**	**IERR**	**Stall**	**Mask**	**Description**
MOVALL_TGT_OOR	0x10E20	1	0	-	MOVALL from a core that does not exist. Command is ignored.
MOVALL_DST_TGT_OOR	0x10E21	1	0	-	MOVALL to a core that does not exist. 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. Command is ignored.
MOVALL_ENABLE_LPI_OFF	0x10E23	1	0	-	MOVALL from a core where GICR_CTLR.Enable_LPIs is 0. 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. LPIs on MOVALL source are dropped.

DISCARD commands					
Error mnemonic	**Encoding**	**IERR**	**Stall**	**Mask**	**Description**
DISCARD_DEVICE_OOR	0x10F01	0	1	CEE	A DISCARD has tried to use a device that is outside the range that the ITS supports. See GITS_BASER0, and for information about the supported range, see GITS_TYPER.
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 commands					
Error mnemonic	**Encoding**	**IERR**	**Stall**	**Mask**	**Description**
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

CLEAR commands					
Error mnemonic	**Encoding**	**IERR**	**Stall**	**Mask**	**Description**
CLEAR_DEVICE_OOR	0x10501	0	1	CEE	A CLEAR has attempted to use a device that is outside the range that the ITS supports. See GITS_BASER0, and for information about the supported range, see GITS_TYPER.
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_TGT_OOR	0x10520	1	0	-	A CLEAR has been sent to an illegal target
CLEAR_CHIP_OFFLINE_OOR	0x10522	1	0	-	A CLEAR has been sent to a target on a chip that is offline
CLEAR_LPI_OFF	0x10523	1	0	-	A CLEAR has been sent to a target that does not have LPIs enabled
CLEAR_PHYSICALID_OOR	0x10526	1	0	-	A CLEAR has tried to drop an interrupt, which has a physical ID that is not supported by the target

INV commands					
Error mnemonic	**Encoding**	**IERR**	**Stall**	**Mask**	**Description**
INV_DEVICE_OOR	0x10C01	0	1	CEE	An INV has tried to use a device that is outside the range that the ITS supports. See GITS_BASER0, and for information about the supported range, see GITS_TYPER.
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_TGT_OOR	0x10C20	1	0	-	An INV has tried to invalidate an interrupt that is mapped to an invalid target
INV_CHIP_OFFLINE_OOR	0x10C22	1	0	-	An INV has tried to invalidate an interrupt that is mapped to a chip that is offline
INV_LPI_OFF	0x10C23	1	0	-	An INV has tried to invalidate an interrupt that is mapped to a target that does not have LPIs enabled. See GICR_CTLR.Enable_LPIs.
INV_PHYSICALID_OOR	0x10C26	1	0	-	An INV has tried to invalidate an interrupt with a physical ID that is larger than the target supports. See GICR_PROPBASER.IDbits.

INVALL commands					
Error mnemonic	**Encoding**	**IERR**	**Stall**	**Mask**	**Description**
INVALL_COLLECTION_OOR	0x10D03	0	1	CEE	An INVALL has tried to invalidate an OOR collection. See GITS_TYPER.
INVALL_UNMAPPED_COLLECTION	0x10D09	0	1	CEE	An INVALL has tried to invalidate a collection that is not mapped
INVALL_TGT_OOR	0x10D20	1	0	-	An INVALL has been sent to an illegal target
INVALL_CHIP_OFFLINE_OOR	0x10D22	1	0	-	An INVALL has tried to invalidate an interrupt from a device that is not mapped
INVALL_LPI_OFF	0x10D23	1	0	-	An INVALL has been sent to a target that has LPIs turned off

INT commands					
Error mnemonic	**Encoding**	**IERR**	**Stall**	**Mask**	**Description**
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. See GITS_BASER0, and for information about the supported range, see GITS_TYPER.
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
INT_TGT_OOR	0x10320	1	0	-	INT received for a core that does not exist. Software must correct mappings. Interrupt is dropped and the target is reported in GICT_ERR<n>MISC1.
INT_CHIP_OFFLINE_OOR	0x10322	1	0	-	INT received for a chip that is offline. Software must either correct mappings or take the chip online. Interrupt is dropped and TGT is reported in GICT_ERR<n>MISC1.
INT_LPI_OFF	0x10323	1	0	-	INT received for a target with GICR_CTLR.Enable_LPIs disabled. Software must either enable LPI or correct mappings. Target is reported in GICT_ERR<n>MISC1.
INT_PHYSICALID_OOR	0x10326	1	0	-	INT received with a physical ID that is beyond the range that is specified in GICR_PROPBASER.IDbits. Software must correct mappings. Interrupt is dropped and ID is reported in GICT_ERR<n>MISC1.

Implementation-defined feature commands					
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. See GITS_BASER0, and for information about the supported range, see GITS_TYPER.

Implementation-defined feature commands					
Error mnemonic	**Encoding**	**IERR**	**Stall**	**Mask**	**Description**
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. See GITS_BASER0, and for information about the supported range, see GITS_TYPER.
OPR_UNMAPPED_DEVICE	0x10A04	1	-	-	Software has tried to lock an interrupt from a device that is not mapped through GITS_OPR. The GITS_OPSR register reports a fail.
OPR_ID_OOR	0x10A05	1	-	-	Software has tried to lock an interrupt using an EventID that is larger than the specified device supports. The GITS_OPSR register reports a fail.
OPR_UNMAPPED_INTERRUPT	0x10A07	1	-	-	Software has tried to lock an interrupt that is not mapped through GITS_OPR. The GITS_OPSR register 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. The GITS_OPSR register reports a fail.
ACE_LITE_ACCESS_FAILURE	0x10B01	1	-	-	An access that the ITS issues, receives an SLVERR or DECODE error. The address is given in GICT_ERR<n>MISC1. This error can occur from multiple sources. Software must determine whether the Command queue is stalled, by checking GITS_CREADR.Stalled. If the Command queue has stalled, the command might not have occurred. See 4.9.2 ITS commands and errors on page 62.
ACE_LITE_TRANS_FAILURE	0x10B03	1	-	AEE	An unknown source in the system has written to the subordinate port with an access that is not a legal GITS_TRANSLATER access. The full address of the access is given in GICT_ERR<n>MISC1. If the address matches GITS_TRANSLATER, then the size, length, strobes, or access type is wrong. Read accesses are not tracked.
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. Software must allocate and clear a new second-level table, update the first-level entry, and repeat the command.
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. Software must correct this and then resume.

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

Similarly, GICT_ERR<n>MISC0 and GICT_ERR<n>MISC1 cannot be written, except the counter fields, if the corresponding GICT_ERR<n>STATUS.MV bit is set, and GICT_ERR<n>ADDR cannot be written if GICT_ERR<n>STATUS.AV is set.

Recommended recovery sequences are described for each error record in 4.15.6.1 Software error record 0 on page 72 to 4.15.6.7 ITS command and translation error records 13 and beyond on page 81.

4.15.7 Bus errors

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

The GICT_ERR0CTLR.UE bit can be used to enable the SLVERR ACE-Lite bus error for the syndromes shown in the following table.

Table 4-16: Bus error syndromes

Syndrome	Description	Direction
SYN_ACE_BAD	ACE-Lite transactions are either bad or unrecognized	Read and write
SYN_GICD_CORRUPTED	Data read from SPI RAM is corrupted	Read-only
SYN_GICR_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

4.16 Multichip operation

During silicon configuration, the system designers can configure the GIC-600AE to support multichip operation.

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

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

GICD_CHIPR<n>

These Chip Registers define 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 the value of its chip_id signal.

GICD_DCHIPR

The Default Chip Register 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

The Chip Status Register 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 4.16.1 Connecting the chips on page 88.

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. Also:

- 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.

4.16.1 Connecting the chips

Use the following procedure to connect the chips in a multichip configuration.

Before you begin

The following restrictions apply when connecting or removing chips:

- You must consider that data that is 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. The GICD_DCHIPR.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 any chip in the system has an ITS block, parameter `its_type_support = full`, then direct injection LPI registers are not supported.
- If you are connecting together multiple different instances of the GIC-600AE, 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`)

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

About this task

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

Procedure

1. Ensure that the values of the chip_id tie-off input signals to all chips are correct.
2. Ensure that all Group enables in the GICD_CTLR register are disabled and GICD_CTLR.RWP == 0.
3. Designate a chip, chip `x`, to own the Routing table.
   You can designate a different chip later if necessary.
4. In a single register write, program GICD_CHIPRx with:
   a) GICD_CHIPRx.ADDR so that each chip can 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 signal value or a more complex identifier.

b) GICD_CHIPRx.SPI_BLOCK_MIN and GICD_CHIPRx.SPI_BLOCKS to appropriate values for the SPIs that chip `x` owns.
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

c) GICD_CHIPRx.SocketState = 1

5. To check that the writes are successful, read GICD_CHIPRx.
The writes might fail due to security settings, an overlapping or nonexistent SPI, or if another update is still in progress. If the accesses fail, then GICD_CHIPRx.SocketState == 0, indicating that the chip is offline.
6. To check that the actions of this sequence have executed correctly, read the following register fields and ensure that their values are as follows:

   a. GICD_CHIPSR.RTS == 2 (Consistent)

   b. GICD_DCHIPR.rt_owner == chip `x`

   c. 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:

7. 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 step through any chip that is already connected, or more efficiently by writing to the chip that owns the Routing table, chip_id signal value == rt_owner.
8. Poll GICD_DCHIPR until bit PUP == 0, indicating that the connection is complete.
9. To check whether the write to GICD_CHIPRy is accepted, read GICD_CHIPRy.
For each chip connection, repeat steps 7 on page 90 through 9 on page 90.

4.16.2 Changing the Routing table owner

You can change the chip that owns the Routing table at any time. However, the Routing table owner must be the last chip to be powered down.

About this task

The following procedure describes how to change the owner of the Routing table:

Procedure

1. Write to GICD_DCHIPR.rt_owner with a value that selects the appropriate chip to be the Routing table owner.
The chip_id signal sets the identification value of a chip.
2. Poll for GICD_DCHIPR.PUP == 0.

4.16.3 SPI ownership for multichip operation

The owner of an SPI block is defined by the GICD_CHIPR<n> registers.

You can remove SPI blocks from a chip and add them to another chip by reprogramming the relevant GICD_CHIPR<n> 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 an 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 an SPI block is removed from, or added to, a chip, all programming that is associated with the SPI block returns to the reset state.

You must not alter the SPI_BLOCK_MIN of an online chip because the results are unpredictable. To change SPI_BLOCK_MIN:

1. Move the chip offline by setting GICD_CHIPR<n>.SocketState = 0.
2. Alter SPI_BLOCK_MIN when the chip is brought back online.

4.16.4 Power control and P-Channel

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

The P-Channel has the following states:

RUN (pstate signal == `0x0`)
: The normal functional mode

CONFIG (pstate signal == `0x9`)
: The GIC does not send any cross-chip messages. It accepts incoming messages but does not process them.

OFF (pstate signal == `0xF`)
: 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.

The GIC can exit reset in either the 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.

4.16.5 Isolating a chip from the system

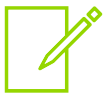
You can isolate a chip from the system.

About this task

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

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.
   Before isolating the chip, the ITSs must be powered off because the Routing table is invalid when the GIC P-Channel is in the OFF state.
3. Ensure that LPIs from other chips are not routed to this chip.
4. Attempt to enter the CONFIG state (pstate signal = `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 the pactive signal. If the pactive signal asserts while attempting to enter a lower power state, you must return to RUN (pstate signal == `0x0`).
5. When in CONFIG state, any required state can be saved.
   Writing to GICD_CHIPRn 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. Flush the LPI cache using GICR_WAKER.Sleep.
   We recommend 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 on page 92). Also, LPIs from other chips must be routed back while the chip is in the OFF state.

   If LPIs arrive after sleep is set in the CONFIG state, then the LPIs are dropped.
8. Attempt to enter the OFF state.
   If the pactive signal is HIGH, return to the CONFIG state.
9. Use the Q-Channel to put the GIC into a safe mode to reset.
   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 values 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.

4.16.6 SPI operation for multichip 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_IROUTERn register. Remote chips are targeted using either Affinity2 or Affinity3, and the affinity level can be discovered using GICD_CFGID.AFSL.

If SPIs within an SPI block are sent to multiple chips, we recommend that you do not read or write the GICD_ISACTIVERn, GICD_ICACTIVERn, GICD_ISPENDRn, and GICD_ICPENDRn registers. 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, we recommend that sources are programmed to write SPI IDs that their chip owns. Other SPI IDs are supported if these SPIs are owned somewhere in your system.

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

SPI Collators in multichip

The SPI Collator wires are always connected to the lowest owned SPIs on the chip.

For example, 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 2-chip system, each chip must not use more wires than 16 × (the number of configured SPI blocks).

SPI 1 of N

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

4.16.7 LPI multichip operation

The GIC-600AE does not use physical target addresses, so GITS_TYPER.PTA == 0. Therefore, GIC-600AE uses the value of GICR_TYPER.ProcessorNumber to route all LPIs and commands to their targets.

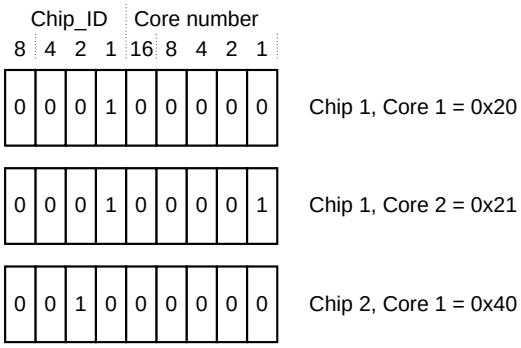
The GIC-600AE splits the GITS_TYPER.ProcessorNumber value into two fields, Chip_ID and the padded linear on-chip core number.

The width of the padded on-chip core number field is defined by the `max_pe_on_chip` configuration parameter. This parameter sets the maximum number of cores or threads 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 ceil[$\log_2$(17)] = 5 bits. 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 4-4: 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.

5. Programmers model

All the GIC-600AE 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-600AE registers:

- The GIC-600AE implements only memory-mapped registers
- The GIC-600AE 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, depending on the value of 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 Cold reset, resets all register bits to zero
- The GIC-600AE ACE-Lite subordinate interface 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 and version 4 defines the permitted sizes of access.


Note

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

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

RO	Read-only
RW	Read and write
WO	Write-only, reads return as **UNKNOWN**

5.1 Register map pages

The GIC-600AE address map has multiple pages. The number of pages and the address aliasing depends on the GIC configuration.

The following table shows the register map pages.

Table 5-1: Register map pages

Page offset	Page	Description
0	GICD	5.2 Distributor registers (GICD/GICDA) summary on page 98

Page offset	Page	Description
1	GICA	5.3 Distributor registers (GICM) for message-based SPIs summary on page 117
2	GICT	5.8 GICT register summary on page 147
3	GICP	5.9 GICP register summary on page 163
4 + 2×ITSnum	GITSn	5.6 ITS control register summary on page 136. ITSnum is the serial number of each ITS, which is from 0 to ITScount−1.
5 + 2×ITSnum	GITSn translate	5.7 ITS translation register summary on page 147
4 + 2×ITScount + 2×RDnum	GICR (LPI)	5.4 Redistributor registers for control and physical LPIs summary on page 121. ITScount is the total number of ITS.
5 + 2×ITScount + 2×RDnum	GICR (SGI)	5.5 Redistributor registers for SGIs and PPIs summary on page 130. RDnum is the serial number of each "internal Redistributor", which is from 0 to RDcount−1.
4 + 2×ITScount + 2×RDcount	GICDA	Alias to GICD (page after last GICR page). RDcount is the total number of "internal Redistributor", which equals total number of CPU cores.

For more information, see the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.

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. Allowing the GIC pages to be globally accessible might require the system interconnect to alias the page addresses.

The registers in the previous table are accessible through the ACE-Lite interface. The *Fault Management Unit* (FMU) registers are accessible through the APB interface. See:

- 5.10 FMU register summary on page 179
- 6.2.1 FMU APB4 interface on page 196

Page offset

The ACE-Lite address bits[x:16] control which GIC register page is accessed in Table 5-1: Register map pages on page 95. The value of `x` depends on the `axis_addr_width` GICD configuration parameter.

In non-monolithic configurations, the GIC-600AE ignores address bits above ceil[$\log_2$(page_count)] + 15. For example, a configuration that uses 11 pages ignores address bits above 19, so any address bits of the form `0xXXXXX00000` is accepted and it accesses the GICD page.

In monolithic configurations, where the Distributor and ITS share the ACE-Lite subordinate port, there are two address tie-off signals that control 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 A.6 Miscellaneous signals on page 258 for information about the gicd_page_offset and its_transr_page_offset tie-off signals.

5.1.1 Discovery

We recommend 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 relate to GIC registers, software can check these pointers against the discovery registers, which start at offset `0xFFD0` for each GIC page. These registers allow discovery of the architecture version and, for GIC-600AE, whether the page contains the Distributor, ITS, or Redistributor registers. For example, to discover the page type, software can:

1. Read from `0xFFE0` to determine the PIDR0.PART_0 value.
2. Read from `0xFFE4` to determine the PIDR1.PART_1 value.
3. Concatenate PART_1 (4 bits) and PART_0 (8 bits), to discover the 12-bit part number, PART_1|| PART_0. A value of:
   - `0x492` indicates that this page contains Distributor registers.
   - `0x493` indicates that this page contains Redistributor registers.
   - `0x494` indicates that this page contains ITS registers.

When this information is known, software can obtain additional information from registers that are specific to each page.

For Redistributors, we recommend 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-600AE has only two pages for each Redistributor because the GICR_TYPER.VLPIS bit indicates 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-600AE 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 signal. Therefore, the chip_id signal value must be set before the GIC exits from reset.

For more information, see the GICv3 and GICv4 Software Overview.

5.1.2 GIC-600AE register access and banking

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

For more information about the register access and banking scheme, see the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.

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).

5.2 Distributor registers (GICD/GICDA) summary

The GIC-600AE 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 document or the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.

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

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

Table 5-2: Distributor (GICD/GICDA) register summary

Offset	Name	Type	Reset	Width	Description	Architecture defined?
0x0000	GICD_CTLR	RW	Configuration dependent	32	Distributor Control Register	Yes
0x0004	GICD_TYPER	RO	Configuration dependent	32	Interrupt Controller Type Register	Yes
0x0008	GICD_IIDR	RO	0x030nn43B The nn value depends on the rxpy identifier.	32	Distributor Implementer Identification Register	Yes
0x000C-0x001C	-	-	-	-	Reserved	-
0x0020	GICD_FCTLR	RW	0x0	32	Function Control Register	No

Offset	Name	Type	Reset	Width	Description	Architecture defined?
0x0024	GICD_SAC	RW	Tie-off dependent[2]	32	Secure Access Control register	No
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[3][4]	WO	-	32	Secure SPI Set Register	Yes
0x0054	-	-	-	-	Reserved	-
0x0058	GICD_CLRSPI_SR[3][4]	WO	-	32	Secure SPI Clear Register	Yes
0x005C-0x007C	-	-	-	-	Reserved	-
0x0080-0x00FC	GICD_IGROUPRn[4]	RW	0x0	32	Interrupt Group Registers, n = 0-31, but n=0 is Reserved	Yes
0x0100-0x017C	GICD_ISENABLERn	RW	0x0	32	Interrupt Set-Enable Registers, n = 0-31, but n=0 is Reserved	Yes
0x0180-0x01FC	GICD_ICENABLERn	RW	0x0	32	Interrupt Clear-Enable Registers, n = 0-31, but n=0 is Reserved	Yes
0x0200-0x027C	GICD_ISPENDRn	RW	SPI signal dependent	32	Interrupt Set-Pending Registers, n = 0-31, but n=0 is Reserved	Yes
0x0280-0x02FC	GICD_ICPENDRn	RW	SPI signal dependent	32	Interrupt Clear-Pending Registers, n = 0-31, but n=0 is Reserved	Yes
0x0300-0x037C	GICD_ISACTIVERn	RW	0x0	32	Interrupt Set-Active Registers, n = 0-31, but n=0 is Reserved	Yes
0x0380-0x03FC	GICD_ICACTIVERn	RW	0x0	32	Interrupt Clear-Active Registers, n = 0-31, but n=0 is Reserved	Yes
0x0400-0x07FC	GICD_IPRIORITYRn	RW	Security dependent	32	Interrupt Priority Registers, n = 0-255, but n=0-7 are Reserved when affinity routing is enabled	Yes
0x0800-0x0BFC	-	-	-	-	Reserved	-
0x0C00-0x0CFC	GICD_ICFGRn	RW	0x0	32	Interrupt Configuration Registers, n = 0-63, but n=0-1 are Reserved	Yes
0x0D00-0x0D7C	GICD_IGRPMODRn	RW	0x0	32	Interrupt Group Modifier Registers, n = 0-31, but n=0 is Reserved. If GICD_CTLR.DS == 1, then this register is RAZ/WI.	Yes
0x0D80-0x0DFC	-	-	-	-	Reserved	-

[2] 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.
[3] The existence of this register depends on the configuration of the GIC-600AE. If Security support is not included, then this register does not exist.
[4] This register is only accessible from a Secure access.

Offset	Name	Type	Reset	Width	Description	Architecture defined?
0x0E00-0x0EFC	GICD_NSACRn[3]	RW	0x0	32	Non-secure Access Control Registers, n = 0-63, but n=0-1 are Reserved when affinity routing is enabled	Yes
0x0F00-0x5FFC	-	-	-	-	Reserved	-
0x6000-0x7FF8	GICD_IROUTERn	RW	0x0080000000	64	Interrupt Routing Registers, n = 0-991, but n=0-31 are Reserved when affinity routing is enabled. See the GICv3 and GICv4 Software Overview. All SPIs are reset with Interrupt_Routing_Mode == 1. The first register is GICD_IROUTER32.	Yes
0xC000	GICD_CHIPSR	RO	P-Channel dependent	32	Chip Status Register. Reserved in single-chip configurations.	No
0xC004	GICD_DCHIPR	RW	0x0	32	Default Chip Register. Reserved in single-chip configurations.	No
0xC008-0xC080	GICD_CHIPRn	RW	0x0	64	Chip Registers, n = 0-15. Reserved in single-chip configurations.	No
0xC088-0xDFFC	-	-	-	-	Reserved	-
0xE000-0xE0FC	GICD_ICLARn	RW	0x0	32	Interrupt Class Registers, n = 0-63, but n=0-1 are Reserved	No
0xE100-0xE17C	GICD_ICERRRn	RW	0x0	32	Interrupt Clear Error Registers, n = 0-31, but n=0 is Reserved	No
0xE180-0xEFFC	-	-	-	-	Reserved	-
0xF000	GICD_CFGID	RO	Configuration dependent	64	Configuration ID Register	No
0xF008-0xFFCC	-	-	-	-	Reserved	-
0xFFD0	GICD_PIDR4	RO	0x44	32	Peripheral ID 4 Register	No
0xFFD4	GICD_PIDR5	RO	0x00	32	Peripheral ID 5 Register	No
0xFFD8	GICD_PIDR6	RO	0x00	32	Peripheral ID 6 Register	No
0xFFDC	GICD_PIDR7	RO	0x00	32	Peripheral ID 7 Register	No
0xFFE0	GICD_PIDR0	RO	0x92	32	Peripheral ID 0 Register	No
0xFFE4	GICD_PIDR1	RO	0xB4	32	Peripheral ID 1 Register	No
0xFFE8	GICD_PIDR2	RO	0x3B	32	Peripheral ID 2 Register	No
0xFFEC	GICD_PIDR3	RO	0x00	32	Peripheral ID 3 Register	No
0xFFF0	GICD_CIDR0	RO	0x0D	32	Component ID 0 Register	No
0xFFF4	GICD_CIDR1	RO	0xF0	32	Component ID 1 Register	No
0xFFF8	GICD_CIDR2	RO	0x05	32	Component ID 2 Register	No
0xFFFC	GICD_CIDR3	RO	0xB1	32	Component ID 3 Register	No

5.2.1 GICD_CTLR, Distributor Control Register

This register enables interrupts and affinity routing.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

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.

Bit descriptions

Figure 5-1: GICD_CTLR bit assignments

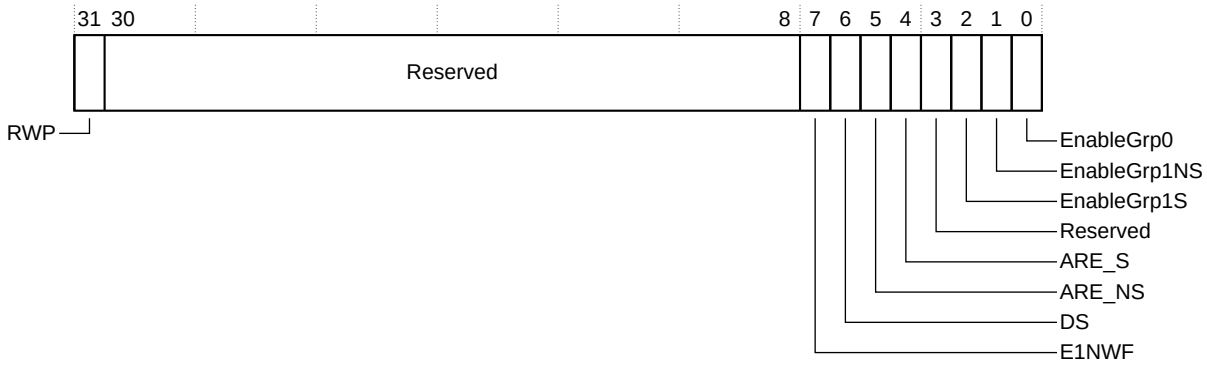


Table 5-3: GICD_CTLR bit descriptions

Bits	Name	Description	Type	Reset
[31]	RWP	Register Write Pending: **0** No register write in progress **1** Register write in progress	RO	0
[30:8]	-	Reserved	-	-
[7]	E1NWF	Enable 1 of N Wakeup Functionality	RW	0

Bits	Name	Description	Type	Reset
[6]	DS	Disable Security	RW	Resets to: • 0 when the `ds_value` configuration parameter is set to 0 or P • 1 when `ds_value` is set to 1
[5]	ARE_NS	Affinity Routing Enable, Non-secure state	RO	1
[4]	ARE_S	Affinity Routing Enable, Secure state	RO	1
[3]	-	Reserved	-	-
[2]	EnableGrp1S	Enable Secure Group 1 interrupts	RW	0
[1]	EnableGrp1NS	Enable Non-secure Group 1 interrupts	RW	0
[0]	EnableGrp0	Enable Group 0 interrupts	RW	0

5.2.2 GICD_TYPER, Interrupt Controller Type Register

This register returns information about the configuration of the GIC-600AE. 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.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-2: GICD_TYPER bit assignments

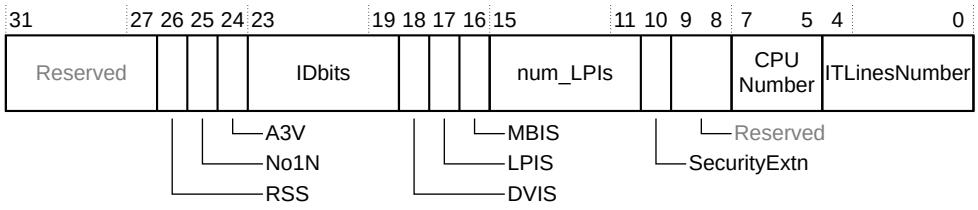


Table 5-4: GICD_TYPER bit assignments

Bits	Name	Description
[31:26]	-	Reserved, returns `0b000000`
[25]	No1N	1 of N SPI: **0** The GIC-600AE supports 1 of N SPI interrupts.
[24]	A3V	Affinity level 3 values. Depending on the configuration, returns either: **0** The GIC-600AE Distributor only supports zero values of affinity level 3. **1** The GIC-600AE Distributor supports nonzero values of affinity level 3.
[23:19]	IDbits	Interrupt identifier bits: **`0b01111`** The GIC-600AE supports 16 interrupt identifier bits.
[18]	DVIS	Direct virtual LPI injection support: **0** The GIC-600AE does not support direct virtual LPI injection. See the GICv3 and GICv4 Software Overview.
[17]	LPIS	Indicates whether the implementation supports LPIs. Depending on the configuration, returns either: **0** LPIs are not supported **1** LPIs are supported
[16]	MBIS	Message-based interrupt support: **1** The GIC-600AE supports message-based interrupts.
[15:11]	num_LPIs	Returns `0b00000` because GICD_TYPER.IDbits indicates the number of LPIs that the GIC supports.
[10]	SecurityExtn	Security state support. Depending on the configuration, returns either: **0** The GIC-600AE supports only a single Security state. **1** The GIC-600AE supports two Security states. When GICD_CTLR.DS == 1, this field is RAZ.
[9:8]	-	Reserved, returns `0b00000`
[7:5]	CPUNumber	Returns `0b000` because GICD_CTLR.ARE==1 (ARE_NS & ARE_S).
[4:0]	ITLinesNumber	Returns the maximum SPI INTID that this GIC-600AE implementation supports, and is given by 32×(ITLinesNumber + 1) − 1.

5.2.3 GICD_IIDR, Distributor Implementer Identification Register

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

Configurations

This register is available in all configurations.

Attributes

Width 32-bit

Functional group See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-3: GICD_IIDR bit assignments

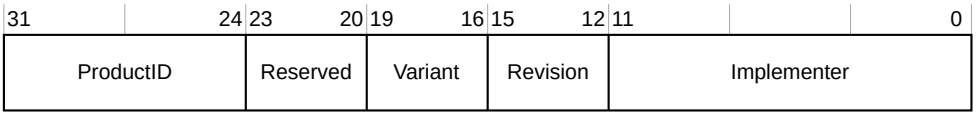


Table 5-5: GICD_IIDR bit descriptions

Bits	Name	Function
[31:24]	ProductID	Indicates the product ID: **0x03** GIC-600AE
[23:20]	-	Reserved, RAZ
[19:16]	Variant	Indicates the major revision, or variant, of the product `rmpn` identifier: **0x0** r0
[15:12]	Revision	Indicates the minor revision of the product `rmpn` identifier: **0x1** p0 **0x3** p1 **0x4** p2 **0x5** p3
[11:0]	Implementer	Identifies the implementer: **0x43B** Arm

5.2.4 GICD_FCTLR, Function Control Register

This register controls non-architectural functionality such as the scrubbing of all RAMs in the local Distributor. The register is not distributed and only acts on the local chip.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Functional group

See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

Usage constraints

Some bits are only accessible by Secure accesses.

Bit descriptions

Figure 5-4: GICD_FCTLR bit assignments

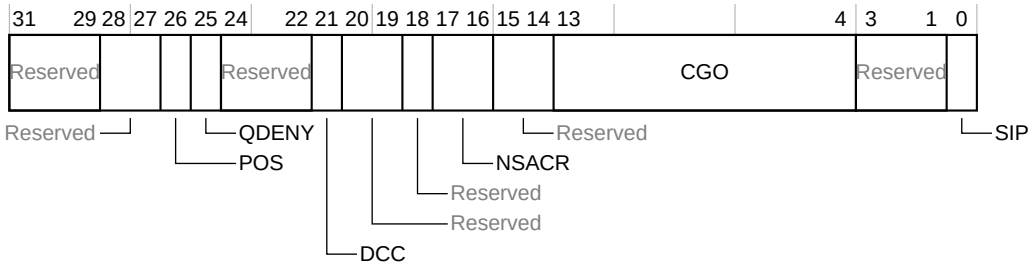


Table 5-6: GICD_FCTLR bit descriptions

Bits	Name	Description
[31:29]	-	Reserved, returns `0b00000`
[28:27]	-	Reserved, RES0
[26]	POS	Point of serialization. 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. **0** Store locally and propagate when possible **1** Propagate access to POS Resets to `0b0`.
[25]	QDENY	Q-Channel deny. Overrides the Q-Channel logic and forces the Distributor to reject powerdown requests.
[24:22]	-	Reserved, RES0
[21]	DCC	Do not correct cache. Modifies the a<x>cache output signals from the Distributor. See 4.11 Memory access and attributes on page 64.
[20:19]	-	Reserved, RES0
[18]	-	Reserved
[17:16]	NSACR	Non-secure access control. Values are as described in the GICD_NSACR register. This is the value that is used if an SPI has an error. Secure access only. Resets to `0b00`.
[15:14]	-	Reserved, returns `0b00`

Bits	Name	Description
[13:4]	CGO	Clock gate override. One bit per clock gate: **0** Use full clock gating **1** Leave clock running. If clock gates are not implemented, then you must use this value. The clock gate bit assignments are: **Bit[13], CGO[9]** Reserved **Bit[12], CGO[8]** ITS communications block **Bit[11], CGO[7]** Pending table search and control **Bit[10], CGO[6]** Trace and debug **Bit[9], CGO[5]** SGI and GICR registers **Bit[8], CGO[4]** LPI cache and search **Bit[7], CGO[3]** ACE-Lite manager interface **Bit[6], CGO[2]** ACE-Lite subordinate interface **Bit[5], CGO[1]** SPI registers and search **Bit[4], CGO[0]** CPU communications block
[3:1]	-	Reserved, returns `0b000`
[0]	SIP	Scrub in progress: **0** No scrub in progress **1** Scrub in progress This bit is read and written by software. When a scrub is complete, the GIC clears the bit to 0.

5.2.5 GICD_SAC, Secure Access Control register

This register allows Secure software to control Non-secure access to GIC-600AE Secure features by other software. It also controls whether Secure PMU events are visible to Non-secure software.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

Usage constraints

Only accessible by Secure accesses.

Bit descriptions

Figure 5-5: GICD_SAC bit assignments

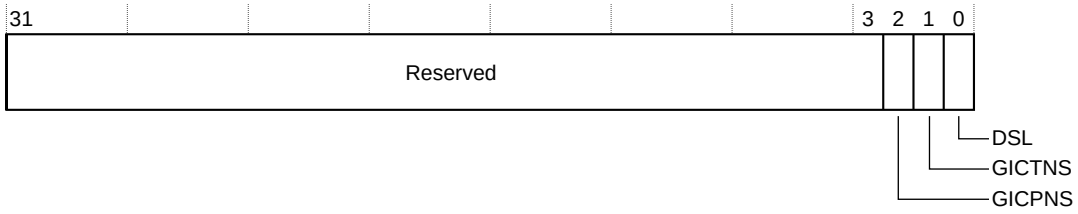


Table 5-7: GICD_SAC bit assignments

Bits	Name	Description	Type
[31:3]	-	Reserved, returns zero	-
[2]	GICPNS	Controls whether the Non-secure world can access the Secure PMU data: **0** Secure access only **1** Allow Non-secure access to the GICP registers The gicp_allow_ns tie-off signal controls the reset value on a per-chip basis.	RW
[1]	GICTNS	Controls whether the Non-secure world can access the Secure trace data: **0** Secure access only **1** Allow Non-secure access to the GICT registers The gict_allow_ns tie-off signal controls the reset value on a per-chip basis.	RW
[0]	DSL	Disable security lock. *WriteOnce* (WO): **0** No effect **1** WO bit to lock GICD_CTLR.DS to be WO at its current value When set to 1, this bit only returns to 0 when the GIC is reset.	RW

5.2.6 GICD_CHIPSR, Chip Status Register

This register returns the status of the chip in a multichip configuration. A single copy of this register exists on each chip in a multichip configuration.

Configurations

This register is available in all multichip configurations.

Attributes

Width 32-bit
Functional group See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

Usage constraints

Only accessible by Secure reads.

Bit descriptions

Figure 5-6: GICD_CHIPSR bit assignments

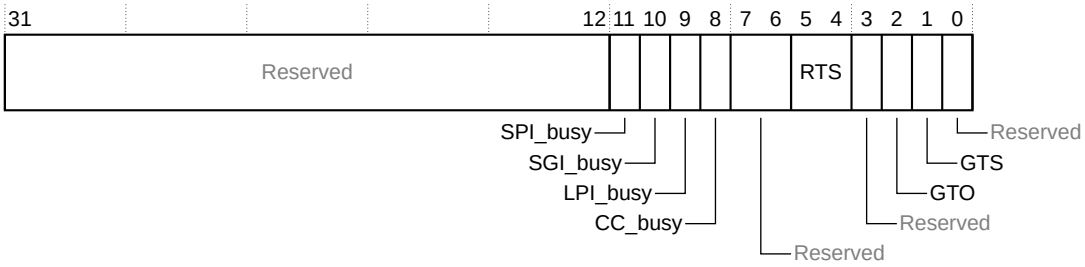


Table 5-8: GICD_CHIPSR bit descriptions

Bits	Name	Description
[31:12]	-	Reserved, RES0
[11]	SPI_busy	**0** ongoing SPI-related cross-chip traffic **1** no traffic
[10]	SGI_busy	**0** ongoing SGI-related traffic or not all cores are asleep **1** no traffic
[9]	LPI_busy	**0** ongoing LPI-related traffic **1** no traffic
[8]	CC_busy	**0** ongoing cross-chip traffic **1** no traffic
[7:6]	-	Reserved, RES0
[5:4]	RTS	Routing table status: **0b00** disconnected **0b01** updating **0b10** consistent **0b11** Reserved
[3]	-	Reserved, RES0
[2]	GTO	Gating transaction ongoing: **0** no accesses **1** accesses ongoing
[1]	GTS	Gating status: **0** not gated **1** gated
[0]	-	Reserved, RES0

5.2.7 GICD_DCHIPR, Default Chip Register

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.

Configurations

This register is available in all multichip configurations.

Attributes

Width 32-bit
Functional group See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

Usage constraints

Only accessible by Secure accesses.

Bit descriptions

Figure 5-7: GICD_DCHIPR bit assignments

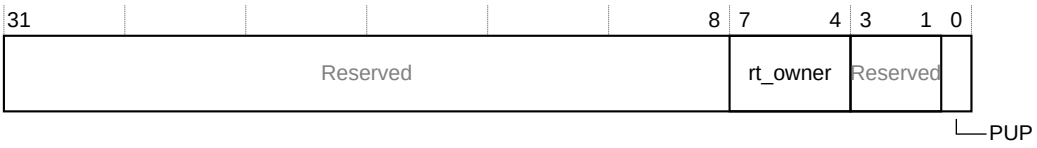


Table 5-9: GICD_DCHIPR bit assignments

Bits	Name	Description	Type
[31:8]	-	Reserved	-
[7:4]	rt_owner	Routing table owner: Value = 0-maximum chip, in the configuration	RW
[3:1]	-	Reserved	-
[0]	PUP	Power update in progress: **0** PUP not in progress **1** PUP in progress	RO

5.2.8 GICD_CHIPR<n>, Chip Registers

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.

Configurations

This register is available in all multichip configurations.

Attributes

Width 64-bit
Functional group See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

Usage constraints

Only accessible by Secure accesses.

Ignores writes if any interrupt group enable is set, that is, GICD_CTLR.EnableGrp0 == 1, or EnableGrp1NS == 1, or EnableGrp1S == 1.

Bit descriptions

Figure 5-8: GICD_CHIPR<n> bit assignments

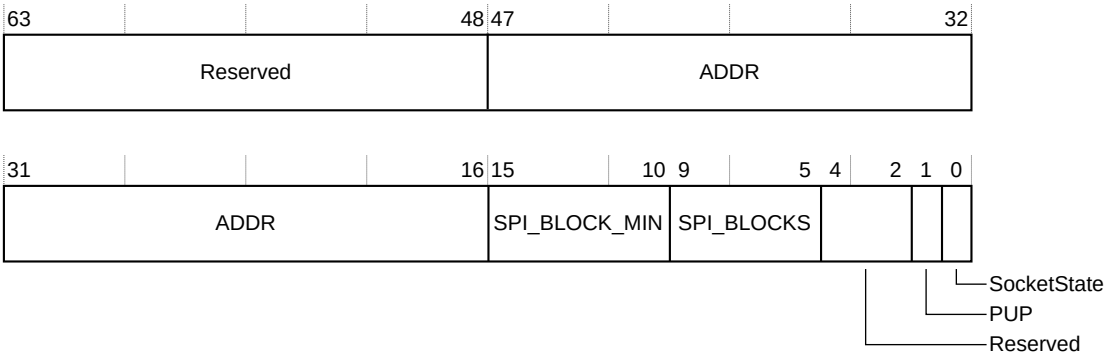


Table 5-10: GICD_CHIPR<n> bit assignments

Bits	Name	Description	Type
[63:48]	-	Reserved	-
[47:16]	ADDR	Controls the value of the icdrtdest signal, when routing messages to the remote chip. The `chip_addr_width` configuration parameter controls the width of this field, so the field spans from bit[16] upwards.	RW
[15:10]	SPI_BLOCK_MIN	Controls the minimum number of SPIs in a group (block). The permitted values are 0-31.	RW
[9:5]	SPI_BLOCKS	Controls the number of SPI blocks. The permitted values are 0-31.	RW
[4:2]	-	Reserved	-
[1]	PUP	This bit returns the power update status: **0** Power update complete **1** Power update in progress	RO
[0]	SocketState	This bit controls the state of the chip: **0** Chip is offline **1** Chip is online	RW

5.2.9 GICD_ICLARn, Interrupt Class Registers

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-600AE has 60 registers, GICD_ICLAR2-GICD_ICLAR61.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

Usage constraints

The Distributor provides up to 60 registers to support 960 SPIs. If you configure the GIC-600AE 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.

These registers are only accessible when the corresponding GICD_IROUTERn.Interrupt_Routing_Mode == 1.

Bit descriptions

Figure 5-9: GICD_ICLARn bit assignments

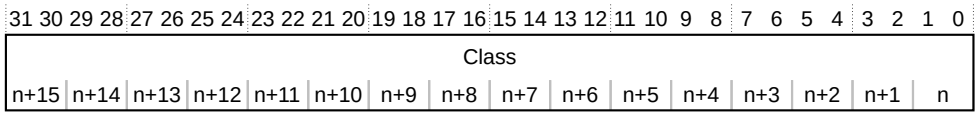


Table 5-11: GICD_ICLARn bit descriptions

Bits	Name	Description
[31:0] Bits[2x+1:2x], for x = 0 to 15	Class<x>	Controls whether the 1 of N SPI can target a core, depending on the class group that the core is assigned to: **0b00** The SPI can target a core that is assigned to class 0 or class 1 **0b01** The SPI can target a core that is assigned to class 1 **0b10** The SPI can target a core that is assigned to class 0 **0b11** The SPI cannot target a core that is assigned to class 0 or class 1 The SPI that a bit refers to, depends on its bit position and the base address offset of the GICD_ICLARn, that is, SPI = 16×n + bit[number]/2.

5.2.10 GICD_ICERRRn, Interrupt Clear Error Registers

These registers can clear the error status of an SPI or return the error status of an SPI. Each register monitors 32 SPIs and the GIC-600AE has 30 registers, GICD_ICERRR1-GICD_ICERRR30.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

Usage constraints

The Distributor provides up to 30 registers to support 960 SPIs. If you configure the GIC-600AE 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.

Bit descriptions

In earlier versions of the GIC-600AE, this register was known as the GICD_IERRRn.

Figure 5-10: GICD_ICERRRn bit assignments

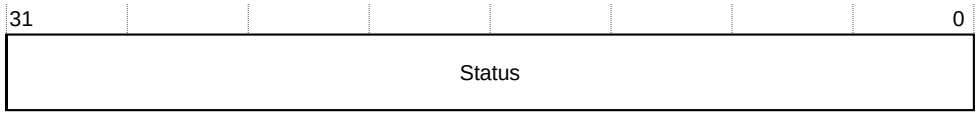


Table 5-12: GICD_ICERRRn bit descriptions

Bits	Name	Description
[31:0]	Status	Indicates whether an SPI is in an error state: **0** If read, the SPI is not in an error state and programming is valid. Writing 0 has no effect. **1** If read, the SPI is in an error state and programming is not valid. Writing 1 clears the error. The SPI that a bit refers to, depends on its bit position and the base address offset of the GICD_ICERRRn, that is, SPI = 32×n + bit[number].

5.2.11 GICD_CFGID, Configuration ID Register

This register contains information that enables test software to determine if the GIC-600AE system is compatible.

Configurations

This register is available in all configurations.

Attributes

Width 64-bit
Functional group See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-11: GICD_CFGID bit assignments

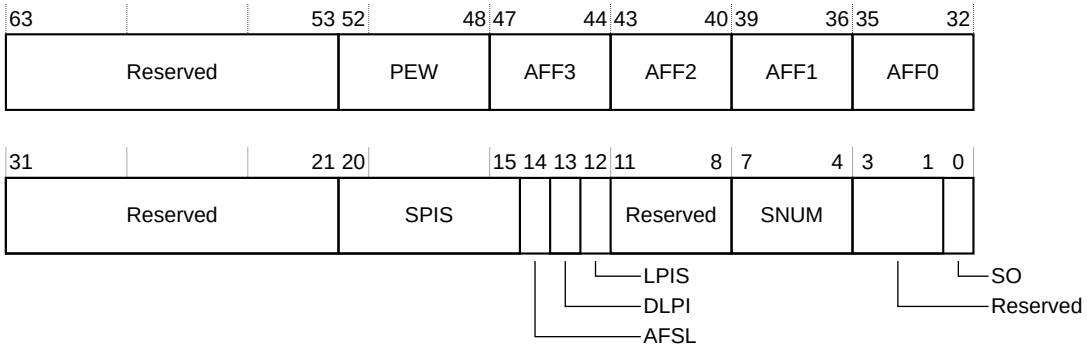


Table 5-13: GICD_CFGID bit assignments

Bits	Name	Description
[63:53]	-	Reserved, returns zero
[52:48]	PEW	Width of lower part of on-chip core number field, ceil[$\log_2$(max_pe_on_chip)]. `max_pe_on_chip` is a configuration option that is set during system integration, which defines the maximum number of cores on a single chip in the system. See 4.16.7 LPI multichip operation on page 93 for more information.
[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]	CNUM	Chip number
[3:1]	-	Reserved, returns zero
[0]	SO	Socket online status: **0** chip is offline **1** chip is online

5.2.12 GICD_PIDR4, Peripheral ID4 register

This register returns byte[4] of the peripheral ID. The GICD_PIDR4 register is part of the set of Distributor peripheral identification registers.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-12: GICD_PIDR4 bit assignments

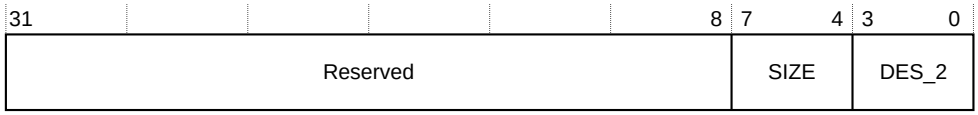


Table 5-14: GICD_PIDR4 bit descriptions

Bits	Name	Description
[31:8]	-	Reserved, RAZ
[7:4]	SIZE	Returns `0x4`, which indicates that the Distributor occupies 64KB of memory, (2^{SIZE} × 4KB).
[3:0]	DES_2	Returns `0x4`, which represents bits[10:7] of the JEDEC JEP106 identification code. Together, GICD_PIDR1.DES_0, GICD_PIDR2.DES_1, and DES_2 identify the component designer.

5.2.13 GICD_PIDR3, Peripheral ID3 register

This register returns byte[3] of the peripheral ID. The GICD_PIDR3 register is part of the set of Distributor peripheral identification registers.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit

Functional group See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-13: GICD_PIDR3 bit assignments

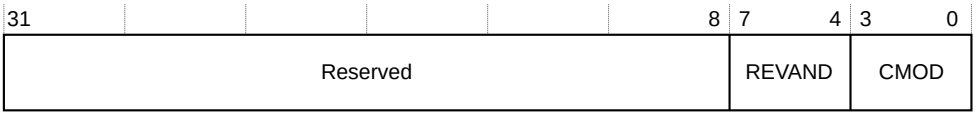


Table 5-15: GICD_PIDR3 bit descriptions

Bits	Name	Description
[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. `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. `0x0`.

5.2.14 GICD_PIDR2, Peripheral ID2 register

This register returns byte[2] of the peripheral ID. The GICD_PIDR2 register is part of the set of Distributor peripheral identification registers.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-14: GICD_PIDR2 bit assignments

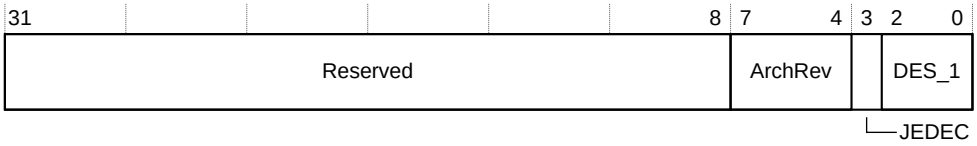


Table 5-16: GICD_PIDR2 bit descriptions

Bits	Name	Description
[31:8]	-	Reserved, RAZ
[7:4]	ArchRev	Identifies the version of the GIC architecture with which the Distributor complies: **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[7:4].

5.2.15 GICD_PIDR1, Peripheral ID1 register

This register returns byte[1] of the peripheral ID. The GICD_PIDR1 register is part of the set of Distributor peripheral identification registers.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-15: GICD_PIDR1 bit assignments

Table 5-17: GICD_PIDR1 bit descriptions

Bits	Name	Description
[31:8]	-	Reserved, RAZ
[7:4]	DES_0	Returns `0xB`, which represents bits[3:0] of the JEDEC JEP106 identification code. Together, DES_0, GICD_PIDR2.DES_1, and GICD_PIDR4.DES_2 identify the component designer.
[3:0]	PART_1	Returns `0x4`, which represents bits[11:8] of the 12-bit part number of the Distributor. Together, GICD_PIDR0.PART_0 and PART_1 field values indicate the part number of the Distributor.

5.2.16 GICD_PIDR0, Peripheral ID0 register

This register returns byte[0] of the peripheral ID. The GICD_PIDR0 register is part of the set of Distributor peripheral identification registers.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.2 Distributor registers (GICD/GICDA) summary on page 98 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-16: GICD_PIDR0 bit assignments

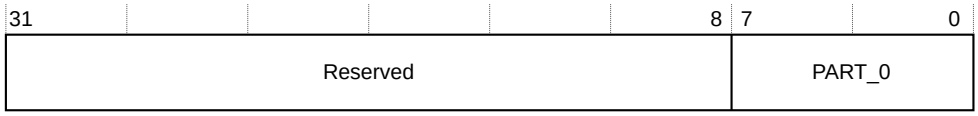


Table 5-18: GICD_PIDR0 bit descriptions

Bits	Name	Description
[31:8]	-	Reserved, RAZ
[7:0]	PART_0	Returns `0x92`, which represents bits[7:0] of the 12-bit part number of the Distributor. Together, PART_0 and GICD_PIDR1.PART_1 field values indicate the part number of the Distributor.

5.3 Distributor registers (GICM) for message-based SPIs summary

The functions for the GIC-600AE message-based SPIs are controlled through the Distributor registers identified with the prefix GICM.

This page was previously known as the GICA page.

The following table lists the message-based SPI registers in base offset order and provides a reference to the register description that is described in either this document or the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4. The WO registers allow 16-bit accesses.

Table 5-19: Distributor registers (GICM) for message-based SPIs summary

Offset	Name	Type	Reset	Width	Description	Architecture defined?
0x0000-0x0004	-	-	-	-	Reserved	-
0x0008	GICM_TYPER	RO	Configuration dependent	64	Message-based Type Register	Yes
0x0010-0x003C	-	-	-	-	Reserved	-
0x0040	GICM_SETSPI_NSR	WO	-	32	Message-based Non-secure SPI Set Register	Yes
0x0044	-	-	-	-	Reserved	-
0x0048	GICM_CLRSPI_NSR	WO	-	32	Message-based Non-secure SPI Clear Register	Yes
0x004C	-	-	-	-	Reserved	-
0x0050	GICM_SETSPI_SR[5]	WO[6]	-	32	Message-based Secure SPI Set Register	Yes
0x0054	-	-	-	-	Reserved	-
0x0058	GICM_CLRSPI_SR[5]	WO[6]	-	32	Message-based Secure SPI Clear Register	Yes
0x005C-0xFFC8	-	-	-	-	Reserved	-
0xFFCC	GICM_IIDR	RO	0x030nn43B The nn value depends on the *rxpy* identifier.	32	Message-based Distributor Implementer Identification Register	Yes
0xFFD0	GICM_PIDR4	RO	0x44	32	Peripheral ID 4 register	No
0xFFD4	GICM_PIDR5	RO	0x00	32	Peripheral ID 5 register	No
0xFFD8	GICM_PIDR6	RO	0x00	32	Peripheral ID 6 register	No
0xFFDC	GICM_PIDR7	RO	0x00	32	Peripheral ID 7 register	No
0xFFE0	GICM_PIDR0	RO	0x97	32	Peripheral ID 0 register	No
0xFFE4	GICM_PIDR1	RO	0xB4	32	Peripheral ID 1 register	No

[5] The existence of this register depends on the configuration of the GIC-600AE. If Security support is not included, this register does not exist.
[6] This register is only accessible from a Secure access.

Offset	Name	Type	Reset	Width	Description	Architecture defined?
0xFFE8	GICM_PIDR2	RO	0x3B	32	Peripheral ID 2 register	No
0xFFEC	GICM_PIDR3	RO	0x00	32	Peripheral ID 3 register	No
0xFFF0	GICM_CIDR0	RO	0x0D	32	Component ID 0 register	No
0xFFF4	GICM_CIDR1	RO	0xF0	32	Component ID 1 register	No
0xFFF8	GICM_CIDR2	RO	0x05	32	Component ID 2 register	No
0xFFFC	GICM_CIDR3	RO	0xB1	32	Component ID 3 register	No

5.3.1 GICM_TYPER, Message-based Type Register

This register returns information about the number of SPIs that are assigned to the frame.

Configurations

This register is available in all configurations.

Attributes

Width 64-bit
Functional group See 5.3 Distributor registers (GICM) for message-based SPIs summary on page 117 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-17: GICM_TYPER bit assignments

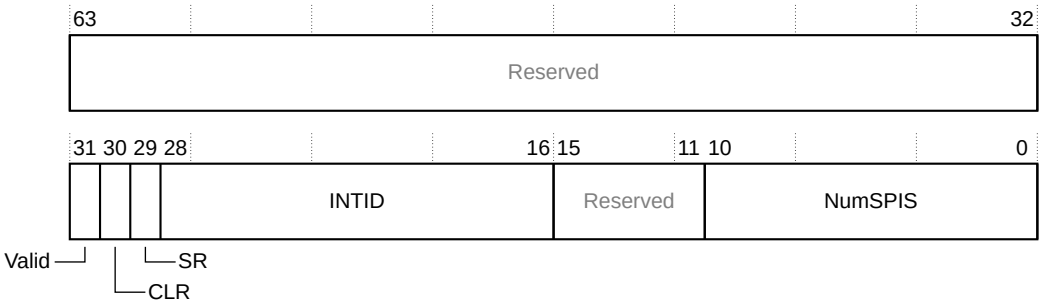


Table 5-20: GICM_TYPER bit descriptions

Bits	Name	Description
[63:32]	-	Reserved, RES0
[31]	Valid	Returns 1 to indicate that the register reports information about the capabilities of the frame
[30]	CLR	Returns 1 to indicate that the GICM_CLRSPI registers are present

Bits	Name	Description
[29]	SR	Indicates whether the GICM_CLRSPI_SR and GICM_SETSPI_SR registers are present: **0** GICM_CLRSPI_SR and GICM_SETSPI_SR registers are not present **1** GICM_CLRSPI_SR and GICM_SETSPI_SR registers are present (only permitted value when Valid==1)
[28:16]	INTID	The INTID of the lowest or first SPI that is assigned to the frame
[15:11]	-	Reserved, RES0
[10:0]	NumSPIS	Returns the number of SPIs that are assigned to the frame.

5.3.2 GICM_IIDR, Message-based Distributor Implementer Identification Register

This register provides information about the implementer and revision of the message-based Distributor page.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.3 Distributor registers (GICM) for message-based SPIs summary on page 117 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-18: GICM_IIDR bit assignments

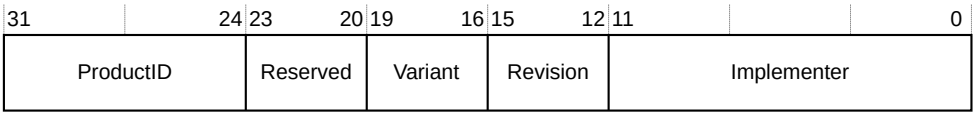


Table 5-21: GICM_IIDR bit descriptions

Bits	Name	Function
[31:24]	ProductID	Indicates the product ID: **0x03** GIC-600AE
[23:20]	-	Reserved, RAZ
[19:16]	Variant	Indicates the major revision, or variant, of the product `rmpn` identifier: **0x0** r0

Bits	Name	Function
[15:12]	Revision	Indicates the minor revision of the product `rmpn` identifier: **0x1** p0 **0x3** p1 **0x4** p2 **0x5** p3
[11:0]	Implementer	Identifies the implementer: **0x43B** Arm

5.4 Redistributor registers for control and physical LPIs summary

The functions for the GIC-600AE physical LPIs are controlled through the Redistributor registers identified with the prefix GICR. These registers start from the base address of the Redistributor.

For more information about LPIs, see the GICv3 and GICv4 Software Overview.

For descriptions of registers that are not specific to the GIC-600AE, see the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.

Table 5-22: Redistributor registers for control and physical LPIs summary

Offset	Name	Type	Reset	Width	Description	Architecture defined?
0x0000	GICR_CTLR	RW	0x0	32	Redistributor Control Register	Yes
0x0004	GICR_IIDR	RO	0x030nn43B The nn value depends on the r*x*p*y* identifier.	32	Redistributor Implementation Identification Register	Yes
0x0008	GICR_TYPER	RO	Configuration dependent	64	Redistributor Type Register	Yes
0x0010	-	-	-	-	Reserved	-
0x0014	GICR_WAKER	RW[7]	0x6	32	Power Management Control Register	[8]
0x0018-0x001C	-	-	-	-	Reserved	-
0x0020	GICR_FCTLR	RW	0x0	32	Function Control Register	No
0x0024	GICR_PWRR	RW	Configuration dependent	32	Power Register	No
0x0028	GICR_CLASSR	RW	0x0	32	Class Register	No
0x002C-0x003C	-	-	-	-	Reserved	-

[7] This register is only accessible from a Secure access.
[8] Parts of this register are architecture defined and the other parts are microarchitecture defined.

Offset	Name	Type	Reset	Width	Description	Architecture defined?
0x0040	GICR_SETLPIR[9]	WO	-	64	-	Yes
0x0048	GICR_CLRLPIR[9]	WO	-	64	-	Yes
0x0050-0x006C	-	-	-	-	Reserved	-
0x0070	GICR_PROPBASER[10]	RW	Configuration dependent	64	Redistributor Properties Base Address Register	Yes
0x0078	GICR_PENDBASER[10]	RW[7]	Configuration dependent	64	Redistributor LPI Pending Table Base Address Register We recommend that if possible, you set the GICR_PENDBASER Pending Table Zero bit to one. This setting reduces the power and time that is taken during initialization.	Yes
0x0080-0x009C	-	-	-	-	Reserved	-
0x00A0	GICR_INVLPIR[9]	WO	-	64	-	Yes
0x00A8-0x00AC	-	-	-	-	Reserved	-
0x00B0	GICR_INVALLR[9]	WO	-	64	-	Yes
0x00B8-0x00BC	-	-	-	-	Reserved	-
0x00C0	GICR_SYNCR[9]	RO	0x0	32	-	Yes
0x00C4-0xFFCC	-	-	-	-	Reserved	-
0xFFD0	GICR_PIDR4	RO	0x44	32	Peripheral ID 4 Register	No
0xFFD4	GICR_PIDR5	RO	0x00	32	Peripheral ID 5 Register	No
0xFFD8	GICR_PIDR6	RO	0x00	32	Peripheral ID 6 Register	No
0xFFDC	GICR_PIDR7	RO	0x00	32	Peripheral ID 7 Register	No
0xFFE0	GICR_PIDR0	RO	0x93	32	Peripheral ID 0 Register	No
0xFFE4	GICR_PIDR1	RO	0xB4	32	Peripheral ID 1 Register	No
0xFFE8	GICR_PIDR2	RO	0x3B	32	Peripheral ID 2 Register	No
0xFFEC	GICR_PIDR3	RO	0x00	32	Peripheral ID 3 Register	No
0xFFF0	GICR_CIDR0	RO	0x0D	32	Component ID 0 Register	No
0xFFF4	GICR_CIDR1	RO	0xF0	32	Component ID 1 Register	No
0xFFF8	GICR_CIDR2	RO	0x05	32	Component ID 2 Register	No
0xFFFC	GICR_CIDR3	RO	0xB1	32	Component ID 3 Register	No

[9] This register is present only when Direct LPI registers are configured.
[10] The existence of this register depends on the configuration of the GIC-600AE. If ITS and LPI support is not included, this register does not exist.

5.4.1 GICR_IIDR, Redistributor Implementation Identification Register

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

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.4 Redistributor registers for control and physical LPIs summary on page 121 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-19: GICR_IIDR bit assignments

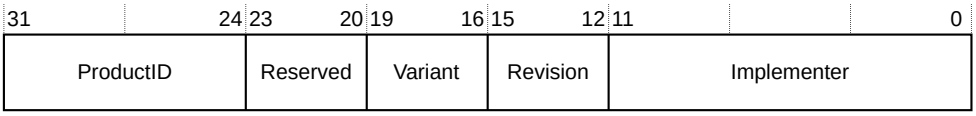


Table 5-23: GICR_IIDR bit descriptions

Bits	Name	Function
[31:24]	ProductID	Indicates the product ID: **0x03** GIC-600AE
[23:20]	-	Reserved, RAZ
[19:16]	Variant	Indicates the major revision, or variant, of the product `rmpn` identifier: **0x0** r0
[15:12]	Revision	Indicates the minor revision of the product `rmpn` identifier: **0x1** p0 **0x3** p1 **0x4** p2 **0x5** p3
[11:0]	Implementer	Identifies the implementer: **0x43B** Arm

5.4.2 GICR_TYPER, Redistributor Type Register

This register returns information about the features that this Redistributor supports.

Configurations

This register is available in all configurations.

Attributes

Width

64-bit

Functional group

See 5.4 Redistributor registers for control and physical LPIs summary on page 121 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-20: GICR_TYPER bit assignments

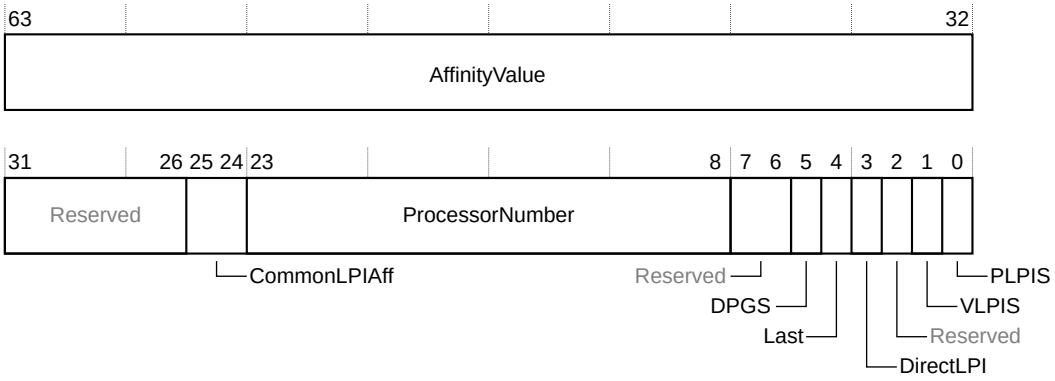


Table 5-24: GICR_TYPER bit assignments

Bits	Name	Function
[63:32]	AffinityValue	Affinity level values for this Redistributor: **Bits[63:56], AF3** The affinity level 3 value **Bits[55:48], AF2** The affinity level 2 value **Bits[47:40], AF1** The affinity level 1 value **Bits[39:32], AF0** The affinity level 0 value
[31:26]	-	Reserved, returns `0b000000`
[25:24]	CommonLPIAff	Returns: **0b00** Single chip configuration **0b01** If chip set by AF3 **0b10** If chip set by AF2 **0b11** Reserved

Bits	Name	Function
[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	Returns 1, to indicate that the GIC-600AE supports *Disable Processor Group Selections*. See GICR_CTLR.DPG1S, GICR_CTLR.DPG1NS, and GICR_CTLR.DPG0 in the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.
[4]	Last	Last Redistributor: **0** This Redistributor is not the last Redistributor on the chip **1** This Redistributor is the last Redistributor on the chip
[3]	DirectLPI	Indicates whether direct injection of physical LPIs is supported: **0** This Redistributor does not support direct injection of physical LPIs. The GICR_SETLPIR, GICR_CLRLPIR, GICR_INVLPIR, GICR_INVALLR, and GICR_SYNCR registers are not implemented. **1** This Redistributor supports direct injection of physical LPIs. The GICR_SETLPIR, GICR_CLRLPIR, GICR_INVLPIR, GICR_INVALLR, and GICR_SYNCR registers are implemented.
[2]	-	Reserved, returns 0
[1]	VLPIS	Virtual LPI support: **0** The Redistributor does not support virtual LPIs
[0]	PLPIS	Indicates whether the Redistributor supports physical LPIs: **0** The Redistributor does not support physical LPIs. This value occurs when `lpi_support` == 0. **1** The Redistributor supports physical LPIs. This value occurs when `lpi_support` == 1.

5.4.3 GICR_WAKER, Power Management Control Register

This register controls whether the GIC-600AE can be powered down.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.4 Redistributor registers for control and physical LPIs summary on page 121 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-21: GICR_WAKER bit assignments

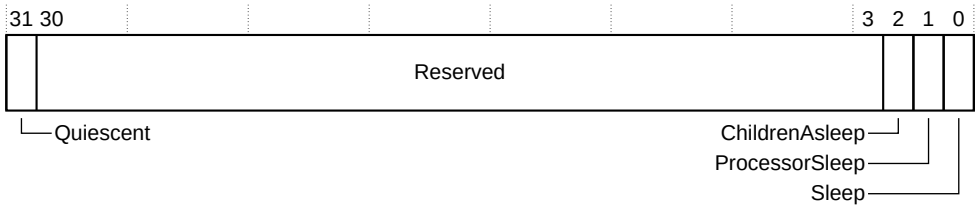


Table 5-25: GICR_WAKER bit descriptions

Bits	Name	Description
[31]	Quiescent	Indicates that the GIC-600AE 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
[1]	ProcessorSleep	Indicates: **0** This Redistributor never asserts a wake_request signal and it delivers the interrupt to the core **1** This Redistributor must assert a wake_request signal if there is a pending interrupt targeted at the connected core. See 4.6.2 Processor core power management on page 57.
[0]	Sleep	Indicates the sleep state: **0** Normal operation **1** The GIC-600AE ensures that all the caches are consistent with external memory and that it is safe to power down. See 4.6.3 Other power management on page 58.

5.4.4 GICR_FCTLR, Function Control Register

This register controls the scrubbing of all RAMs in the associated Redistributor.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.4 Redistributor registers for control and physical LPIs summary on page 121 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-22: GICR_FCTLR bit assignments

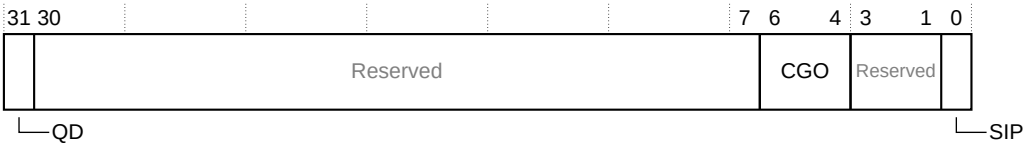


Table 5-26: GICR_FCTLR bit descriptions

Bits	Name	Description
[31]	QD	Q-Channel deny: **0** Allow Q-Channel accesses **1** Deny Q-Channel accesses
[30:7]	-	Reserved, RAZ/WI
[6:4]	CGO	Clock gate override. One bit per clock gate: **0** Use full clock gating **1** Leave clock running. If clock gates are not implemented, then you must use this value. The clock gate bit assignments are: **Bit[6], CGO[2]** Search clock gate **Bit[5], CGO[1]** Downstream message clock gate **Bit[4], CGO[0]** Upstream message clock gate
[3:1]	-	Reserved, RAZ/WI
[0]	SIP	Scrub in progress: **0** No scrub in progress **1** Scrub in progress This bit is read and written by software. When a scrub is complete, the GIC clears the bit to 0.

5.4.5 GICR_PWRR, Power Register

This register controls the powerup sequence of the Redistributors. Software must write to this register during the powerup sequence.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.4 Redistributor registers for control and physical LPIs summary on page 121 for the address offset, type, and reset value of this register.

Usage constraints

Only accessible by Secure accesses.

Bit descriptions

Figure 5-23: GICR_PWRR bit assignments

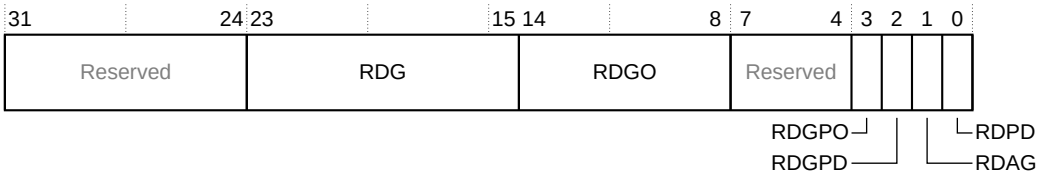


Table 5-27: GICR_PWRR bit descriptions

Bits	Name	Description	Type
[31:24]	-	Reserved, RAZ	-
[23:15]	RDG	RDGroup. This field indicates the number of this Redistributor.	RO
[14:8]	RDGO	RDGroupOffset. This field indicates the identifier of the current core within the Redistributor.	RO
[7:4]	-	Reserved, RAZ	-
[3]	RDGPO	RDGroupPoweredOff. This bit indicates: **0** Redistributor is powered up and can be accessed **1** It is safe to power down the Redistributor	RO
[2]	RDGPD	RDGroupPowerDown. This bit indicates the intentional power state of the Redistributor: **0** Intend to power up **1** Intend to power down The Redistributor has reached its intentional power state when RDGPD = RDGPO.	RO
[1]	RDAG	RDApplyGroup. Setting this bit to 1 applies the RDPD value to all Redistributors on the same Redistributor. If the RDPD value cannot be applied to all cores in the group, then the GIC ignores this request.	WO
[0]	RDPD	RDPowerDown: **0** Redistributor is powered up and can be accessed **1** The core permits the Redistributor to be powered down Writes to 1 are ignored if GICR_WAKER.ProcessorSleep != 1. Writes are ignored if RDGPD != RDGPO and changing to not match RDGPD. If all other cores in the Redistributor group have RDPD == 1, then setting this bit to 1 also sets RDGPD = 1.	RW

Related information

Redistributor power management on page 56

5.4.6 GICR_CLASSR, Class Register

This register specifies which class of 1 of N interrupt the CPU accepts.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.4 Redistributor registers for control and physical LPIs summary on page 121 for the address offset, type, and reset value of this register.

Usage constraints

Only accessible by Secure accesses.

Bit descriptions

Figure 5-24: GICR_CLASSR bit assignments

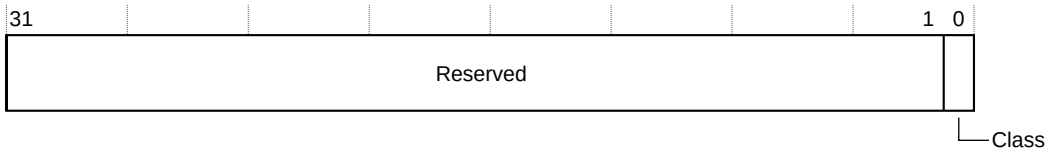


Table 5-28: GICR_CLASSR bit descriptions

Bits	Name	Description
[31:1]	-	Reserved, RAZ/WI
[0]	Class	Interrupt class: **0** Class 0 **1** Class 1

5.4.7 GICR_PIDR2, Peripheral ID2 Register

This register returns byte[2] of the peripheral ID. The GICR_PIDR2 register is part of the set of Redistributor peripheral identification registers.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.4 Redistributor registers for control and physical LPIs summary on page 121 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-25: GICR_PIDR2 bit assignments

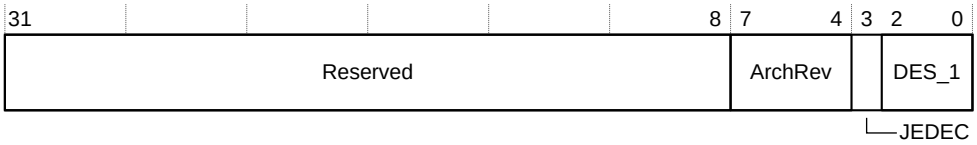


Table 5-29: GICR_PIDR2 bit descriptions

Bits	Name	Description
[31:8]	-	Reserved, RAZ
[7:4]	ArchRev	Identifies the version of the GIC architecture with which the Redistributor complies: **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[7:4].

5.5 Redistributor registers for SGIs and PPIs summary

The functions for the GIC-600AE 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-600AE, see the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.

Table 5-30: Redistributor registers for SGIs and PPIs summary

Offset	Name	Type	Reset	Width	Description	Architecture defined?
0x0000-0x007C	-	-	-	-	Reserved	-
0x0080	GICR_IGROUPR0	RW	0x0	32	Interrupt Group Register	Yes
0x0084-0x0FFC	-	-	-	-	Reserved	-
0x0100	GICR_ISENABLER0	RW	0x0	32	Interrupt Set-Enable Register	Yes
0x0104-0x017C	-	-	-	-	Reserved	-
0x0180	GICR_ICENABLER0	RW	0x0	32	Interrupt Clear-Enable Register	Yes
0x0184-0x01FC	-	-	-	-	Reserved	-
0x0200	GICR_ISPENDR0	RW	PPI wire dependent	32	Interrupt Set-Pending Register	Yes

Offset	Name	Type	Reset	Width	Description	Architecture defined?
0x0204-0x027C	-	-	-	-	Reserved	-
0x0280	GICR_ICPENDR0	RW	PPI wire dependent	32	Peripheral Clear Pending Register	Yes
0x0284-0x02FC	-	-	-	-	Reserved	-
0x0300	GICR_ISACTIVER0	RW	0x0	32	Interrupt Set-Active Register	Yes
0x0304-0x037C	-	-	-	-	Reserved	-
0x0380	GICR_ICACTIVER0	RW	0x0	32	Interrupt Clear-Active Register	Yes
0x0384-0x03FC	-	-	-	-	Reserved	-
0x0400-0x041C	GICR_IPRIORITYRn	RW	0x0	32	Interrupt Priority Registers	Yes
0x0420-0x0BFC	-	-	-	-	Reserved	-
0x0C00-0x0C04	GICR_ICFGRn	RW	0xAAAAAAAA when n == 0. 0x0 when n == 1.	32	Interrupt Configuration Registers	Yes
0x0C08-0x0CFC	-	-	-	-	Reserved	-
0x0D00	GICR_IGRPMODR0	RW	0x0	32	Interrupt Group Modifier Register	Yes
0x0D04-0x0DFC	-	-	-	-	Reserved	-
0x0E00	GICR_NSACR	RW	0x0	32	Non-secure Access Control Register	Yes
0x0E04-0xBFFC	-	-	-	-	Reserved	-
0xC000	GICR_MISCSTATUSR	RO	0x0	32	Miscellaneous Status Register	No
0xC004	-	-	-	-	Reserved	-
0xC008	GICR_IERRVR	RW	0x0	32	Interrupt Error Valid Register	No
0xC00C	-	-	-	-	Reserved	-
0xC010	GICR_SGIDR	RW	-	64	SGI Default Register	No
0xC018-0xEFFC	-	-	-	-	Reserved	-
0xF000	GICR_CFGID0	RO	Configuration dependent	32	Configuration ID0 Register	No
0xF004	GICR_CFGID1	RO	Configuration dependent	32	Configuration ID1 Register	No

5.5.1 GICR_MISCSTATUSR, Miscellaneous Status Register

Use this register to test the integration of the cpu_active and wake_request input signals. You can also use the register to debug the CPU interface enables as seen by the GIC-600AE.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.5 Redistributor registers for SGIs and PPIs summary on page 130 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-26: GICR_MISCSTATUSR bit assignments

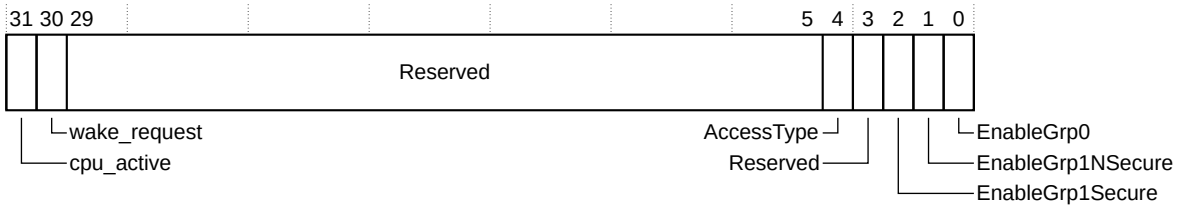


Table 5-31: GICR_MISCSTATUSR bit descriptions

Bits	Name	Description
[31]	cpu_active	Returns the status of the cpu_active signal for the core corresponding to the Redistributor whose register is being read: **0** cpu_active input signal is not active **1** cpu_active input signal is active 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	Returns the status of the wake_request signal: **0** wake_request signal is not active **1** wake_request signal is asserted
[29:5]	-	Reserved
[4]	AccessType	Returns the access type: **0** Secure access **1** Non-secure access
[3]	-	Reserved

Bits	Name	Description
[2][11]	EnableGrp1Secure	In systems that enable two Security states, when GICD_CTLR.DS == 0, then: • For Secure reads, returns the Group 1 Secure CPU interface enable. • For Non-secure reads, returns zero. In systems that only enable a single Security state, when GICD_CTLR.DS == 1, then this bit returns zero.
[1][11]	EnableGrp1NSecure	In systems that enable two Security states, when GICD_CTLR.DS == 0, then: • For Secure reads, this bit returns the Group 1 Non-secure CPU interface enable. • For Non-secure reads, when GICD_CTLR.ARE_NS == 1, this bit returns the Group 1 Non-secure CPU interface enable. • For Non-secure reads when GICD_CTLR.ARE_NS == 0, this bit returns zero. In systems that only enable a single Security state, when GICD_CTLR.DS == 1, this bit returns the Group 1 CPU interface enable.
[0][11]	EnableGrp0	In systems that enable two Security states, when GICD_CTLR.DS == 0, then: • For Secure reads, this bit returns the Group 0 CPU interface enable. • For Non-secure reads when GICD_CTLR.ARE_NS == 0, this bit returns the Group 1 Non-secure CPU interface enable. • For Non-secure reads when GICD_CTLR.ARE_NS == 1, this bit returns zero. In systems that only enable a single Security state, when GICD_CTLR.DS == 1, this bit returns the Group 0 CPU interface enable.

5.5.2 GICR_IERRVR, Interrupt Error Valid Register

This register indicates if the SGI or PPI data has been corrupted in SRAM. You can use this register to clear an error.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Functional group

See 5.5 Redistributor registers for SGIs and PPIs summary on page 130 for the address offset, type, and reset value of this register.

Usage constraints

Only accessible by Secure accesses.

[11] 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 and version 4.

Bit descriptions

Table 5-32: GICR_IERRVR bit descriptions

Bits	Name	Description
[31:16]	valid	Indicates whether a PPI is in an error state: **Bit[n] = 0** If read, PPI[n–16] is not in an error state. Writing 0 has no effect. **Bit[n] = 1** If read, PPI[n–16] is in an error state, so the interrupt is not delivered. Writing 1 clears the error on PPI[n–16].
[15:0]		Indicates whether an SGI is in an error state: **Bit[n] = 0** If read, SGIn is not in an error state. Writing 0 has no effect. **Bit[n] = 1** If read, SGIn is in an error state, so the interrupt is not delivered. Writing 1 clears the error on SGIn.

5.5.3 GICR_SGIDR, SGI Default Register

This register controls the default value of SGI settings, for use in the case of a *Double-bit Error Detect Error* (DEDERR).

Configurations

This register is available in all configurations.

Attributes

Width 64-bit
Functional group See 5.5 Redistributor registers for SGIs and PPIs summary on page 130 for the address offset, type, and reset value of this register.

Usage constraints

Only accessible by Secure accesses. This register does not support 32-bit writes.

Bit descriptions

Table 5-33: GICR_SGIDR bit descriptions

Bits	Name	Description
[3] + 4n: [63, 59, 55, 51, 47, 43, 39, 35, 31, 27, 23, 19, 15, 11, 7, 3]	-	Reserved, RES0
[2] + 4n: [62, 58, 54, 50, 46, 42, 38, 34, 30, 26, 22, 18, 14, 10, 6, 2]	GRPMOD	As GICR_IGRPMODR0 register
[1] + 4n: [61, 57, 53, 49, 45, 41, 37, 33, 29, 25, 21, 17, 13, 9, 5, 1]	GRP	As GICR_IGROUPR0 register
[0] + 4n: [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>

5.5.4 GICR_CFGID0, Configuration ID0 Register

This register returns information about the configuration of the Redistributors.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.5 Redistributor registers for SGIs and PPIs summary on page 130 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-27: GICR_CFGID0 bit assignments

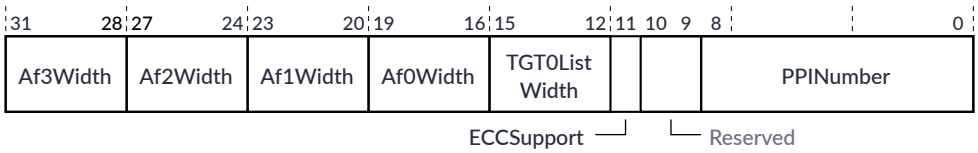


Table 5-34: GICR_CFGID0 bit descriptions

Bits	Name	Description
[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. The ppi_id[15:0] tie-off signal sets the value of the ID. Each Redistributor must have a unique ID.

Related information

Miscellaneous signals on page 258

5.5.5 GICR_CFGID1, Configuration ID1 Register

This register returns information about the configuration of the Redistributors.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.5 Redistributor registers for SGIs and PPIs summary on page 130 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-28: GICR_CFGID1 bit assignments

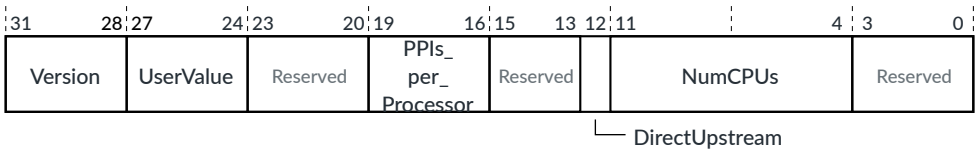


Table 5-35: GICR_CFGID1 bit descriptions

Bits	Name	Description
[31:28]	Version	Identifies the major and minor revisions of GIC-600AE: **0x1** r0p0 **0x3** r0p1 **0x4** r0p2 **0x5** r0p3
[27:24]	UserValue	Modification value that you can set. Indicates whether the customer has modified the behavior of the Redistributor. Usually, this field is `0x0`. Customers change this value when they make authorized modifications to the Redistributor.
[23:20]	-	Reserved, RAZ
[19:16]	PPIs_per_Processor	The number of PPIs for each core − 1
[15:13]	-	Reserved
[12]	DirectUpstream	Indicates a direct upstream connection
[11:4]	NumCPUs	The number of cores that this Redistributor supports. GIC-600AE supports up to 64 cores, so the maximum value of this field is `0x3F`.
[3:0]	-	Reserved, RAZ

5.6 ITS control register summary

The GIC-600AE *Interrupt Translation Service* (ITS) functions are controlled through registers that are identified with the prefix GITS.

This page does not exist in GIC-600AE configurations that do not support LPIs or that do not have an ITS.

For descriptions of registers that are not specific to the GIC-600AE, see the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.

Table 5-36: ITS control register summary

Offset	Name	Type	Reset	Width	Description	Architecture defined?
0x0000	GITS_CTLR	RW	0x80000000	32	ITS Control Register	Yes
0x0004	GITS_IIDR	RO	0x030nn43B The nn value depends on the rxpy identifier.	32	ITS Implementer Identification Register	Yes
0x0008	GITS_TYPER	RO	Configuration dependent	64	ITS Type Register	Yes
0x0010-0x001C	-	-	-	32	Reserved	-
0x0020	GITS_FCTLR	RW	0x0	32	Function Control Register	No
0x0024	-	-	-	-	Reserved	-
0x0028	GITS_OPR	RW	0x0	64	Operations Register	No
0x0030	GITS_OPSR	RO	0x0	64	Operation Status Register	No
0x0038-0x007C	-	-	-	-	Reserved	-
0x0080	GITS_CBASER	RW	0x0	64	Command Queue Control Register. See the GICv3 and GICv4 Software Overview.	Yes
0x0088	GITS_CWRITER	RW	0x0	64	Command Queue Write Pointer Register	Yes
0x0090	GITS_CREADR	RO	0x0	64	Command Queue Read Pointer Register	Yes
0x0098-0x00FC	-	-	-	-	Reserved	-
0x0100	GITS_BASER0	RW	0x0107000000000000	64	ITS Translation Table Descriptor Register0	Yes
0x0108	GITS_BASER1	RW	0x0401000000000000	64	ITS Translation Table Descriptor Register1	Yes
0x0110-0xEFFC	-	-	-	-	Reserved	-
0xF000	GITS_CFGID	RO	Configuration dependent	32	Configuration ID Register	No
0xF004-0xFFCC	-	-	-	-	Reserved	-
0xFFD0	GITS_PIDR4	RO	0x44	32	Peripheral ID 4 Register	No
0xFFD4	GITS_PIDR5	RO	0x00	32	Peripheral ID 5 Register	No

Offset	Name	Type	Reset	Width	Description	Architecture defined?
0xFFD8	GITS_PIDR6	RO	0x00	32	Peripheral ID 6 Register	No
0xFFDC	GITS_PIDR7	RO	0x00	32	Peripheral ID 7 Register	No
0xFFE0	GITS_PIDR0	RO	0x94	32	Peripheral ID 0 Register	No
0xFFE4	GITS_PIDR1	RO	0xB4	32	Peripheral ID 1 Register	No
0xFFE8	GITS_PIDR2	RO	0x3B	32	Peripheral ID 2 Register	No
0xFFEC	GITS_PIDR3	RO	0x00	32	Peripheral ID 3 Register	No
0xFFF0	GITS_CIDR0	RO	0x0D	32	Component ID 0 Register	No
0xFFF4	GITS_CIDR1	RO	0xF0	32	Component ID 1 Register	No
0xFFF8	GITS_CIDR2	RO	0x05	32	Component ID 2 Register	No
0xFFFC	GITS_CIDR3	RO	0xB1	32	Component ID 3 Register	No

5.6.1 GITS_IIDR, ITS Implementer Identification Register

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

Configurations

This register is available in all configurations that have one or more ITS blocks.

Attributes

Width 32-bit
Functional group See 5.6 ITS control register summary on page 136 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-29: GITS_IIDR bit assignments

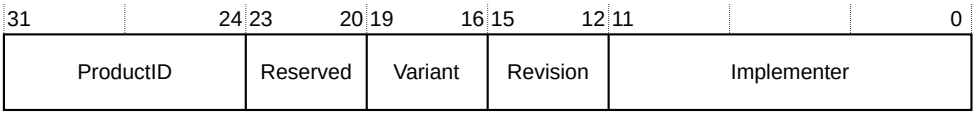


Table 5-37: GITS_IIDR bit descriptions

Bits	Name	Function
[31:24]	ProductID	Indicates the product ID: **0x03** GIC-600AE
[23:20]	-	Reserved, RAZ

Bits	Name	Function
[19:16]	Variant	Indicates the major revision, or variant, of the product `rmpn` identifier: **0x0** r0
[15:12]	Revision	Indicates the minor revision of the product `rmpn` identifier: **0x1** p0 **0x3** p1 **0x4** p2 **0x5** p3
[11:0]	Implementer	Identifies the implementer: **0x43B** Arm

5.6.2 GITS_TYPER, ITS Type Register

This register returns information about the features that this ITS supports.

Configurations

This register is available in all configurations that have one or more ITS blocks.

Attributes

Width 64-bit
Functional group See 5.6 ITS control register summary on page 136 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-30: GITS_TYPER bit assignments

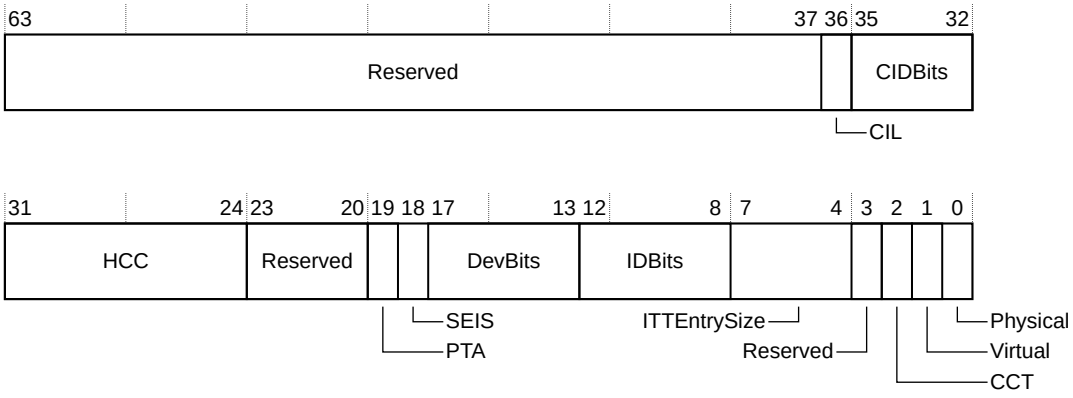


Table 5-38: GITS_TYPER bit descriptions

Bits	Name	Description
[36]	CIL	Collection ID limit: **1** The size of the Collection ID is set by the CIDBits field
[35:32]	CIDBits	The number of Collection ID bits, minus one. Set by the `col_width` configuration parameter.
[31:24]	HCC	Hardware collection count: **0** Interrupt collections are held in external memory only
[23:20]	-	Reserved, returns 0
[19]	PTA	Physical target addresses: **0** The GIC-600AE does not support physical target addresses
[18]	SEIS	System error interrupts: **0** The GIC-600AE does not support locally generated System Error interrupts
[17:13]	DevBits	The number of device identifier bits implemented, minus one. Set by the `did_width` configuration parameter.
[12:8]	IDBits	The number of interrupt identifier bits implemented, minus one. Set by the `vid_width` configuration parameter.
[7:4]	ITTEntrySize	The number of bytes per entry, minus one: **0x3** The GIC-600AE supports a 4-byte ITT entry size
[3]	-	Reserved
[2]	CCT	Cumulative Collection tables: **0** Total number of supported collections is determined by the number of collections that are held in memory only
[1]	Virtual	Indicates whether the ITS supports virtual LPIs and direct injection of virtual LPIs: **0** The ITS does not support virtual LPIs or direct injection of virtual LPIs.
[0]	Physical	Physical LPIs: **1** The GIC-600AE supports physical LPIs

5.6.3 GITS_FCTLR, Function Control Register

This register controls many functions in the local GITS such as cache invalidation, clock gating, and the scrubbing of all RAMs. The register is not distributed and only acts on the local chip.

Configurations

This register is available in all configurations that have one or more ITS blocks.

Attributes

Width 32-bit

Functional group See 5.6 ITS control register summary on page 136 for the address offset, type, and reset value of this register.

Usage constraints

If the ITS is not quiescent, then the GIC ignores writes to some fields. The ITS is quiescent when GITS_CTLR.Quiescent == 1.

Bit descriptions

Figure 5-31: GITS_FCTLR bit assignments

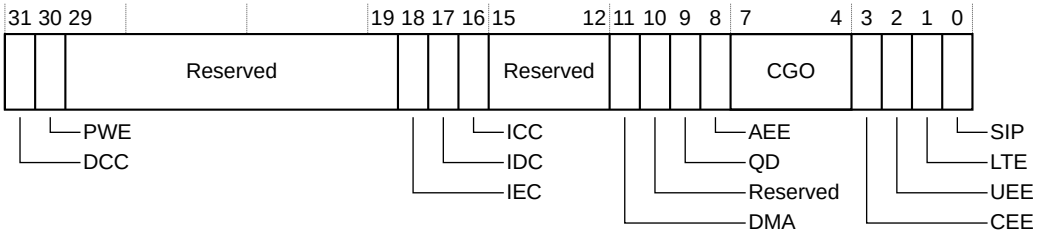


Table 5-39: GITS_FCTLR bit descriptions

Bits	Name	Description	Type
[31]	DCC	Disable cache conversion: **0** Use SMMU attribute for AMBA mapping **1** Use Direct attribute for AMBA mapping Writes ignored if the ITS is not quiescent.	RW
[30]	PWE	Powerdown when enabled: **0** Requests GITS_CTLR.Quiescent to indicate that the ITS is quiescent and can be powered down. **1** Do not request GITS_CTLR.Quiescent to indicate that the ITS is quiescent.	RW
[29:19]	-	Reserved, RAZ/WI	-
[18]	IEC	Invalidate Event cache: **0** No effect **1** Invalidate Event cache	WO
[17]	IDC	Invalidate Device cache: **0** No effect **1** Invalidate Device cache	WO
[16]	ICC	Invalidate Collection cache: **0** No effect **1** Invalidate Collection cache	WO
[15:12]	-	Reserved, RAZ/WI	-
[11]	DMA	Enable translation memory reads through the Distributor to meet PCIe dependency requirements: **0** All memory accesses through ACE-Lite manager interface **1** Enable translation memory reads through Distributor	RW

Bits	Name	Description	Type
[10]	-	Reserved, RAZ/WI	-
[9]	QD	Q-Channel deny: **0** Do not deny Q-Channel requests **1** Always deny Q-Channel requests	RW
[8]	AEE	Access error enable: **0** Do not enable reporting of subordinate access errors **1** Enable reporting of subordinate access errors Writes ignored if the ITS is not quiescent.	RW
[7:4]	CGO	Clock gate override. One bit per clock gate: **0** Use full clock gating **1** Leave clock running. If clock gates are not implemented, then you must use this value. The clock gate bit assignments are: **Bit[7], CGO[3]** Map fetch **Bit[6], CGO[2]** Debug clock **Bit[5], CGO[1]** Command clock **Bit[4], CGO[0]** CCS, translation logic	RW
[3]	CEE	Command error enable: **0** Do not enable reporting of command errors and errors from GITS_OPR operations. **1** Enable reporting of command errors and errors from GITS_OPR operations. See 4.15.6.7 ITS command and translation error records 13 and beyond on page 81. Writes ignored if the ITS is not quiescent.	RW
[2]	UEE	Unmapped error enable: **0** Do not enable reporting of unmapped interrupt errors **1** Enable reporting of unmapped interrupt errors Writes ignored if the ITS is not quiescent.	RW
[1]	LTE	Latency tracking enable: **0** Disable latency tracking of interrupts **1** Enable latency tracking of interrupts Writes ignored if the ITS is not quiescent.	RW

Bits	Name	Description	Type
[0]	SIP	Scrub in progress. When read: **0** No scrub in progress **1** Scrub in progress When written: **0** Abort the scrub **1** Start a scrub When a scrub is complete, the GIC clears the bit to 0.	RW

5.6.4 GITS_OPR, Operations Register

This register controls cache lock.

Configurations

This register is available in all configurations that have one or more ITS blocks.

Attributes

Width 64-bit
Functional group See 5.6 ITS control register summary on page 136 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-32: GITS_OPR bit assignments

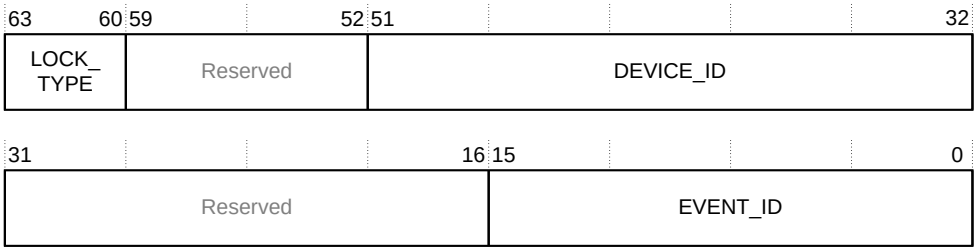


Table 5-40: GITS_OPR bit descriptions

Bits	Name	Description
[63:60]	LOCK_TYPE	Lock type supported: **0** Track **1** Trial **2** ITS lock **3** ITS unlock **4** Track abort **8** ITS unlock all **5-7, 9-15** Reserved **Note:** • If GITS_OPSR.REQUEST_IN_PROGRESS == 1, when attempting a new access (other than Track abort (4) during a Track) the behavior is unpredictable. • Invalidating the VCACHE by using GITS_FCTLR.XXX unlocks all the locked entries. However, if a GITS_OPR lock request occurs while an invalidation is in progress (GITS_FCTLR.XXX == 1), then it is unpredictable whether the entries remain locked when the invalidation completes. This unpredictable behavior might cause GITS_OPSR to return an incorrect status.
[59:52]	-	Reserved, RES0
[51:32]	DEVICE_ID	Sets the DeviceID. The number of bits that are implemented in this field is configuration dependent. To determine the width of this field, software can read GITS_TYPER.DevBits.
[31:16]	-	Reserved, RES0
[15:0]	EVENT_ID	Sets the EventID. The number of bits that are implemented in this field is configuration dependent. To determine the width of this field, software can read GITS_TYPER.IDBits.

5.6.5 GITS_OPSR, Operation Status Register

This register indicates cache lock status.

Configurations

This register is available in all configurations that have one or more ITS blocks.

Attributes

Width 64-bit
Functional group See 5.6 ITS control register summary on page 136 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-33: GITS_OPSR bit assignments

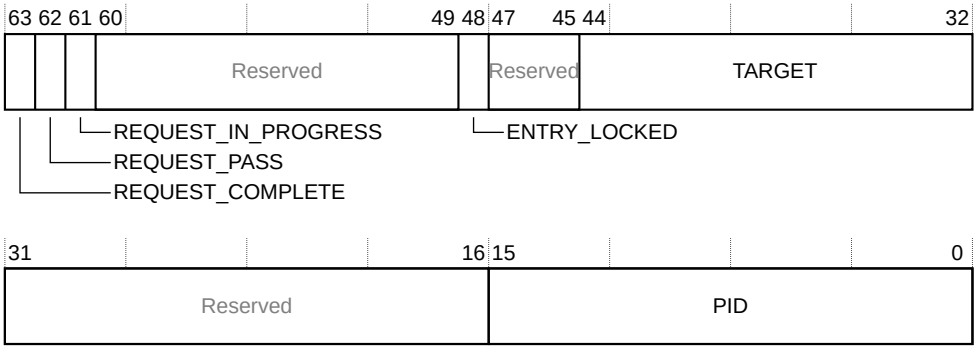


Table 5-41: GITS_OPSR bit descriptions

Bits	Name	Description
[63]	REQUEST_COMPLETE	Request to GITS_OPR completed
[62]	REQUEST_PASS	Request to GITS_OPR completed without error
[61]	REQUEST_IN_PROGRESS	Request to GITS_OPR in progress
[60:49]	-	Reserved, **RES0**
[48]	ENTRY_LOCKED	Locked entry in cache corresponds to request (valid for trial and lock operations)
[47:45]	-	Reserved, **RES0**
[44:32]	TARGET	Target of interrupt requested. Valid for trial and lock operations.
[31:16]	-	Reserved, **RES0**
[15:0]	PID	Physical ID of interrupt requested (valid for trial and lock operations)

5.6.6 GITS_CFGID, Configuration ID Register

This register returns information about the configuration of the ITS block such as its ID number.

Configurations

This register is available in all configurations that have one or more ITS blocks.

Attributes

Width 32-bit
Functional group See 5.6 ITS control register summary on page 136 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-34: GITS_CFGID bit assignments

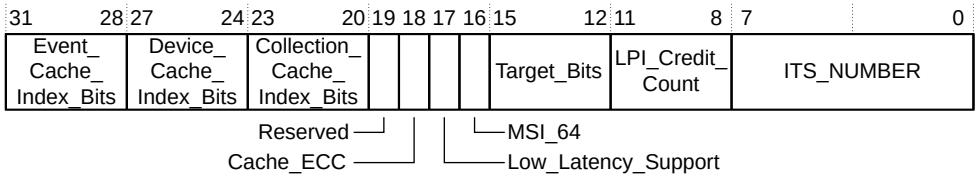


Table 5-42: GITS_CFGID bit descriptions

Bits	Name	Description
[31:28]	Event_Cache_Index_Bits	Number of bits that are used to index the Event cache
[27:24]	Device_Cache_Index_Bits	Number of bits that are used to index the Device cache
[23:20]	Collection_Cache_Index_Bits	Number of bits that are used to index the 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. The `msi_64` configuration parameter sets the value of this bit.
[15:12]	Target_Bits	Number of bits supported for targets
[11:8]	LPI_Credit_Count	Number of LPI credits – 1. The `number_int_credit` configuration parameter minus 1, sets the value of this field.
[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 information

Miscellaneous signals on page 258

5.6.7 GITS_PIDR2, Peripheral ID2 Register

This register returns byte[2] of the peripheral ID. The GITS_PIDR2 register is part of the set of ITS peripheral identification registers.

Configurations

This register is available in all configurations that have one or more ITS blocks.

Attributes

Width 32-bit
Functional group See 5.6 ITS control register summary on page 136 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-35: GITS_PIDR2 bit assignments

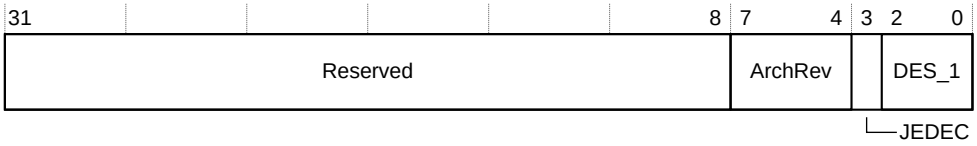


Table 5-43: GITS_PIDR2 bit descriptions

Bits	Name	Description
[31:8]	-	Reserved, RAZ
[7:4]	ArchRev	Identifies the version of the GIC architecture with which the ITS complies: **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[7:4].

5.7 ITS translation register summary

Interrupts to be translated by the GIC-600AE *Interrupt Translation Service* (ITS) are identified by EventIDs that are written to GITS_TRANSLATER, the ITS Translation Register.

This page does not exist in GIC-600AE configurations that do not support LPIs or that do not have an ITS.

Table 5-44: ITS translation register summary

Offset	Name	Type	Reset	Width	Description
0x0000-0x003C	-	-	-	-	Reserved
0x0040	GITS_TRANSLATER	WO	-	32	ITS Translation Register. See the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.
0x0044-0xFFFC	-	-	-	-	Reserved

5.8 GICT register summary

The GIC-600AE trace and debug functions are controlled through registers that are identified with the prefix GICT.

All registers comply with the Arm® Architecture Reference Manual Supplement Reliability, Availability, and Serviceability (RAS), for Armv8-A, except for the GICT_PIDR* and GICT_CIDR* registers.


Note

The GICD_SAC.GICTNS bit controls whether Non-secure software can access the GICT registers.

Table 5-45: GICT register summary

Offset	Name	Type	Reset	Width	Description
0x0000 + (n × 64)	GICT_ERR<n>FR	RO	Record dependent	64	Error Record Feature Register
0x0008 + (n × 64)	GICT_ERR<n>CTLR	RW	0x0	64	Error Record Control Register
0x0010 + (n × 64)	GICT_ERR<n>STATUS	RW	Record dependent	64	Error Record Primary Status register
0x0018 + (n × 64)	GICT_ERR<n>ADDR	RW	Unknown	64	Error Record Address Register
0x0020 + (n × 64)	GICT_ERR<n>MISC0	RW	Unknown	64	Error Record Miscellaneous Register 0
0x0028 + (n × 64)	GICT_ERR<n>MISC1	RW	Unknown	64	Error Record Miscellaneous Register 1
0xE000	GICT_ERRGSR	RO	0x0	64	Error Group Status Register
0xE008- 0xE7FC	-	-	-	-	Reserved, RAZ/WI
0xE800- 0xE808	GICT_ERRIRQCR<n>	RW	0x0	64	Error Interrupt Configuration Registers
0xE810- 0xFFB8	-	-	-	-	Reserved, RAZ/WI
0xFFBC	GICT_DEVARCH	RO	0x47700A00	32	Device Architecture register
0xFFC0- 0xFFC4	-	-	-	-	Reserved, RAZ/WI
0xFFC8	GICT_DEVID	RO	Configuration dependent	32	Device Configuration register
0xFFCC	-	-	-	-	Reserved, RAZ/WI
0xFFD0	GICT_PIDR4	RO	0x44	32	Peripheral ID 4 register
0xFFD4	GICT_PIDR5	RO	0x00	32	Peripheral ID 5 register
0xFFD8	GICT_PIDR6	RO	0x00	32	Peripheral ID 6 register
0xFFDC	GICT_PIDR7	RO	0x00	32	Peripheral ID 7 register
0xFFE0	GICT_PIDR0	RO	0x95	32	Peripheral ID 0 register
0xFFE4	GICT_PIDR1	RO	0xB4	32	Peripheral ID 1 register
0xFFE8	GICT_PIDR2	RO	0x3B	32	Peripheral ID 2 register
0xFFEC	GICT_PIDR3	RO	0x00	32	Peripheral ID 3 register
0xFFF0	GICT_CIDR0	RO	0x0D	32	Component ID 0 register
0xFFF4	GICT_CIDR1	RO	0xF0	32	Component ID 1 register
0xFFF8	GICT_CIDR2	RO	0x05	32	Component ID 2 register
0xFFFC	GICT_CIDR3	RO	0xB1	32	Component ID 3 register

5.8.1 GICT_ERR<n>FR, Error Record Feature Register

This register returns information about the Armv8.2 RAS features that the GIC-600AE implements.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.8 GICT register summary on page 147 for the address offset, type, and reset value of this register.

Usage constraints

If GICD_SAC.GICTNS == 0, then only Secure software can access the contents of this register.

Bit descriptions

Figure 5-36: GICT_ERR<n>FR bit assignments

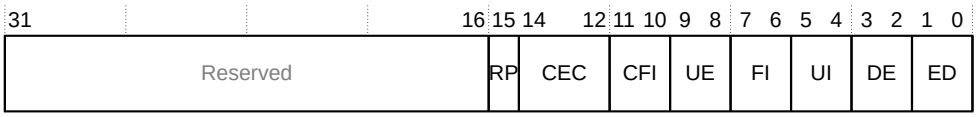


Table 5-46: GICT_ERR<n>FR bit descriptions

Bits	Name	Description
[31:16]	-	Reserved, RAZ
[15]	RP	Repeat corrected error count: **0** The GIC-600AE does not implement a repeat corrected error counter
[14:12]	CEC	Corrected error count: **0b000** The GIC-600AE 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: **0b00** The GIC-600AE does not provide a fault handling interrupt for corrected errors **0b10** The GIC-600AE provides a controllable fault handling interrupt for corrected errors
[9:8]	UE	Uncorrected error. Depending on the configuration, returns either: **0b00** The GIC-600AE does not provide an in-band uncorrected error reporting **0b10** The GIC-600AE provides a controllable in-band uncorrected error reporting
[7:6]	FI	Fault handling interrupt for uncorrected errors. Depending on the configuration, returns either: **0b00** The GIC-600AE does not provide a fault handling interrupt **0b10** The GIC-600AE provides a controllable fault handling interrupt

Bits	Name	Description
[5:4]	UI	Error recovery interrupt for uncorrected errors. Depending on the configuration, returns either: **0b00** The GIC-600AE does not provide an error recovery interrupt for uncorrected errors **0b10** The GIC-600AE provides a controllable error recovery interrupt for uncorrected errors
[3:2]	DE	Deferring of errors support: **0b00** The GIC-600AE does not support the deferring of errors
[1:0]	ED	Uncorrected error reporting: **0b01** Uncorrected error reporting is always enabled

5.8.2 GICT_ERR<n>CTLR, Error Record Control Register

This register controls how interrupts are handled.

Configurations

This register is available in all configurations.

Attributes

Width 64-bit
Functional group See 5.8 GICT register summary on page 147 for the address offset, type, and reset value of this register.

Usage constraints

If GICD_SAC.GICTNS == 0, then only Secure software can access the functions of this register.

Bit descriptions

Figure 5-37: GICT_ERR<n>CTLR bit assignments

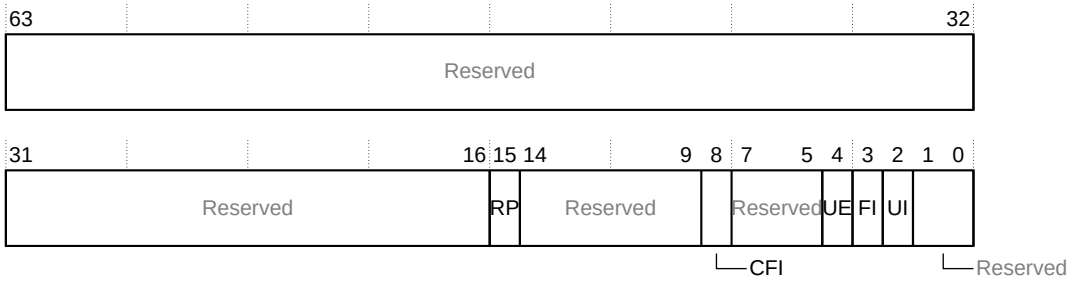


Table 5-47: GICT_ERR<n>CTLR bit descriptions

Bits	Name	Description
[63:16]	-	Reserved, RAZ
[15]	RP	0 = An error response to a transaction is reported
[14:9]	-	Reserved, RAZ

Bits	Name	Description
[8]	CFI	Controls whether a corrected error generates a fault handling interrupt. SBZ on non-correctable errors else: **0** The GIC-600AE does not assert a fault handling interrupt for corrected errors **1** The GIC-600AE asserts a fault handling interrupt, the fault_int signal, when a corrected error occurs
[7:5]	-	Reserved, RAZ
[4]	UE	Uncorrected error. RAZ/WI for all records except GICT error record (0) else: **0** Do not send External abort with transaction **1** Send External abort with transaction. See 4.15.7 Bus errors on page 87.
[3]	FI	Fault handling interrupt. SBZ on *Correctable Error* (CE) records else: **0** Fault handling interrupt is not generated on any error **1** Fault handling interrupt, fault_int signal, is generated on all uncorrectable errors
[2]	UI	Error recovery interrupt for uncorrected error. SBZ on CE records else: **0** Error recovery interrupt is not generated on any error **1** Error recovery interrupt, err_int signal, is generated on all uncorrectable errors
[1:0]	-	Reserved, RAZ

5.8.3 GICT_ERR<n>STATUS, Error Record Primary Status Register

This register indicates information relating to the recorded errors.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.8 GICT register summary on page 147 for the address offset, type, and reset value of this register.

Usage constraints

If GICD_SAC.GICTNS == 0, then only Secure software can access the functions of this register.

Bit descriptions

Figure 5-38: GICT_ERR<n>STATUS bit assignments

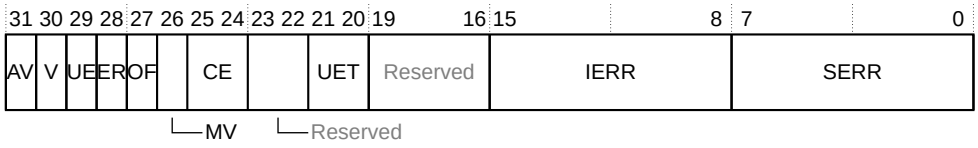


Table 5-48: GICT_ERR<n>STATUS bit descriptions

Bits	Name	Description
[31]	AV	Indicates if the address is valid: **0** GICT_ERR<n>ADDR is not valid **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: **0** GICT_ERR<n>STATUS is not valid **1** GICT_ERR<n>STATUS is valid. One or more errors are recorded.
[29]	UE	Uncorrectable error bit. SBZ in *Correctable Error* (CE) records.
[28]	ER	Indicates that at least one error has been reported over ACE-Lite. 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: **0** GICT_ERR<n>MISC0 and GICT_ERR<n>MISC1 are not valid **1** GICT_ERR<n>MISC0 and GICT_ERR<n>MISC1 are valid
[25:24]	CE	Correctable error. Indicates errors that are correctable as shown in Table 4-7: Error handling records on page 71: **0b00** No CE recorded **0b10** At least one CE recorded
[23:22]	-	Reserved, RAZ/WI
[21:20]	UET	Uncorrectable error type. RES0 unless UE == 1, in which case: **0b10** UEO, uncorrectable error and restartable **0b11** UER, uncorrectable error and recoverable
[19:16]	-	Reserved, RAZ/WI
[15:8]	IERR	Implementation-defined error code. Returns information that Table 5-51: GICT_ERR<n>MISC0.Data field encoding on page 155 shows. This field is RO apart from record 0 and record 13 (and above).
[7:0]	SERR	Architecturally defined primary error code. Returns information that Table 5-51: GICT_ERR<n>MISC0.Data field encoding on page 155 shows. See the Arm® Architecture Reference Manual Supplement Reliability, Availability, and Serviceability (RAS), for Armv8-A for more information about this field. This field is RO apart from record 0.

5.8.4 GICT_ERR<n>ADDR, Error Record Address Register

This register contains the address and security status of the write. This register is only present for GICT software record 0.

Configurations

This register is available in all configurations.

Attributes

Width 64-bit
Functional group See 5.8 GICT register summary on page 147 for the address offset, type, and reset value of this register.

Usage constraints

If GICD_SAC.GICTNS == 0, then only Secure software can access the functions of this register.

Ignores writes if GICT_ERR<n>STATUS.AV == 1.

All bits are RAZ/WI when GICT_ERR<n>STATUS.IERR = 0, 12, or 13.

Bit descriptions

Figure 5-39: GICT_ERR<n>ADDR bit assignments

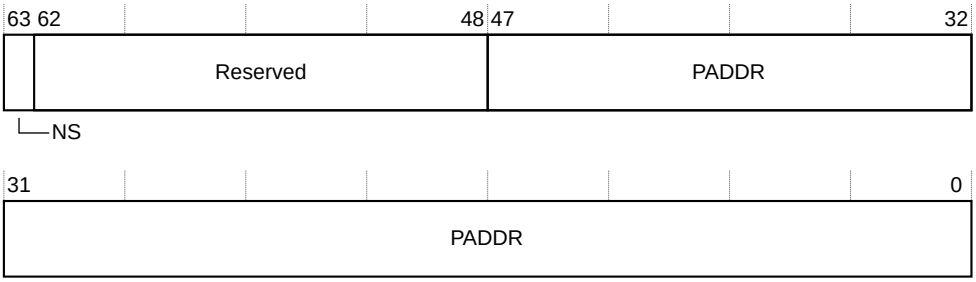


Table 5-49: GICT_ERR<n>ADDR bit descriptions

Bits	Name	Description
[63]	NS	Non-secure attribute: **0** The address is Secure **1** The address is Non-secure
[62:48]	-	Reserved, RAZ/WI
[47:0]	PADDR	The error address. The `axis_addr_width` configuration parameter controls how many bits in this field are implemented, that is, from bit[0]-bit[`axis_addr_width`−1].

5.8.5 GICT_ERR<n>MISC0, Error Record Miscellaneous Register 0

This register contains the corrected error counter and information that assists with identifying the RAM in which the error was detected.

Configurations

This register is available in all configurations.

Attributes

Width 64-bit

Functional group See 5.8 GICT register summary on page 147 for the address offset, type, and reset value of this register.

Usage constraints

None

Bit descriptions

Figure 5-40: GICT_ERR<n>MISC0 bit assignments

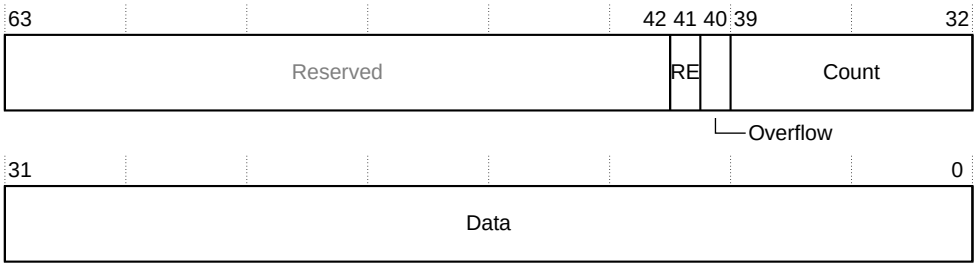


Table 5-50: GICT_ERR<n>MISC0 bit descriptions

Bits	Name	Description
[63:42]	-	Reserved, RAZ
[41]	RE	Rounding error. The rounding error counter is under-reporting.
[40]	Overflow	Sticky overflow bit: **0** counter has not overflowed **1** counter has overflowed If the corrected fault handling interrupt is enabled, then the GIC-600AE generates a fault handling interrupt.
[39:32]	Count	Error count. Error counter is not 0 or is more than 13+. Incremented for each corrected error or uncorrectable error that does not match the recorded syndrome.
[31:0]	Data	Information that is associated with the error. A description of each error code is given in one of the following tables: • Table 4-8: Software errors, record 0 on page 72 • Table 4-9: SPI RAM errors, records 1-2 on page 76 • Table 4-10: SGI RAM errors, records 3-4 on page 78 • Table 4-11: PPI RAM errors, records 7-8 on page 79 • Table 4-12: LPI RAM errors, records 9-10 on page 80 • Table 4-13: ITS RAM errors, records 11-12 on page 80 • 4.15.6.7 ITS command and translation error records 13 and beyond on page 81

The following table shows the Data field encoding for each error record and syndrome.

Table 5-51: GICT_ERR<n>MISC0.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) Always packed from 0 (lowest = 0)
Software error (0)	0x0, SYN_ACE_BAD Illegal ACE-Lite subordinate access.	0xE	AccessRnW, bit[12] AccessSparse, bit[11] AccessSize, bits[10:8] AccessLength, bits[7:0]
Software error (0)	0x1, SYN_PPI_PWRDWN Attempt to access a powered down Redistributor.	0xF	Redistributor, bits[24:16] Core, bits[8:0]
Software error (0)	0x2, SYN_PPI_PWRCHANGE Attempt to power down Redistributor rejected.	0xF	Redistributor, bits[24:16] Core, bits[8:0]
Software error (0)	0x3, SYN_GICR_ARE Attempt to access GICR or GICD registers in mode that cannot work.	0xF	Core, bits[8:0]
Software error (0)	0x4, SYN_PROPBASE_ACC 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 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 Attempt to reprogram ENABLE_LPI when not enabled and not asleep.	0xF	Core, bits[8:0]
Software error (0)	0x7, SYN_WAKER_CHANGE Attempt to change GICR_WAKER abandoned due to handshake rules.	0xF	Core, bits[8:0]
Software error (0)	0x8, SYN_SLEEP_FAIL 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 Core put to sleep before its Group enables were cleared.	0xF	Core, bits[8:0]
Software error (0)	0xA, SYN_GICD_CTLR 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 SGI sent with no valid destinations.	0xE	Core, bits[8:0]
Software error (0)	0x11, SYN_SGI_CORRUPTED SGI corrupted without effect.	0x6	Core, bits[8:0]
Software error (0)	0x12, SYN_GICR_CORRUPTED Data was read from GICR register space that encountered an uncorrectable error.	0x6	GICT_ERR0ADDR is populated
Software error (0)	0x13, SYN_GICD_CORRUPTED Data was read from GICD register space that encountered an uncorrectable error.	0x6	GICT_ERR0ADDR is populated

Record	GICT_ERR<n>STATUS.IERR (syndrome)	GICT_ERR<n>STATUS .SERR	Value and description of GICT_ERR<n>MISC0.Data (other bits RES0) Always packed from 0 (lowest = 0)
Software error (0)	0x14, SYN_ITS_OFF Data was read from an ITS that is powered down.	0xF	GICT_ERR0ADDR is populated
Software error (0)	0x18, SYN_SPI_BLOCK. Attempt to access an SPI block that is not implemented.	0xE	Block, bits[4:0]
Software error (0)	0x19, SYN_SPI_OOR 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 An SPI has no legal target destinations.	0xF	ID, bits[9:0]
Software error (0)	0x1B, SYN_SPI_NO_DEST_1OFN A 1 of N SPI cannot be delivered due to bad GICR_CTLR.DPG<0|1NS|1S> or GICR_CLASSR programming.	0xF	ID, bits[9:0]
Software error (0)	0x1C, SYN_COL_OOR 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 A `Deactivate` command to a nonexistent SPI, or with incorrect groups set. `Deactivate` commands to LPI and nonexistent PPI are not reported.	0xE	None
Software error (0)	0x1E, SYN_SPI_CHIP_OFFLINE An attempt was made to send an SPI to an offline chip.	0xF	ID, bits[9:0]
Software error (0)	0x28, SYN_ITS_REG_SET_OOR 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] Data, bits[15:0]
Software error (0)	0x29, SYN_ITS_REG_CLR _OOR An attempt was made to clear an OOR interrupt. Only valid when GICR LPI injection registers are supported.	0xE	Core, bits[24:16] Data, bits[15:0]
Software error (0)	0x2A, SYN_ITS_REG_INV_OOR An attempt was made to invalidate an OOR interrupt. Only valid when GICR LPI injection registers are supported.	0xE	Core, bits[24:16] Data, bits[15:0]
Software error (0)	0x2B, SYN_ITS_REG_SET_ENB 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] Data, bits[15:0]
Software error (0)	0x2C, SYN_ITS_REG_CLR _ENB 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] Data, bits[15:0]
Software error (0)	0x2D, SYN_ITS_REG_INV_ENB 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] Data, bits[15:0]

Record	GICT_ERR<n>STATUS.IERR (syndrome)	GICT_ERR<n>STATUS.SERR	Value and description of GICT_ERR<n>MISC0.Data (other bits RES0) Always packed from 0 (lowest = 0)
Software error (0)	0x40, SYN_LPI_PROP_READ_FAIL An attempt was made to read properties for a single interrupt where an error response was received with the data.	0x12	Target, bits[29:16] ID, bits[15:0]
Software error (0)	0x41, SYN_PT_PROP_READ_FAIL An attempt was made to read properties for a block of interrupts where an error response was received with the data.	0x12	Target, bits[29:16] ID, bits[15:0]
Software error (0)	0x42, SYN_PT_COARSE_MAP_READ_FAIL An attempt was made to read the coarse map for a target where an error response was received with the data.	0x12	Target, bits[29:16]
Software error (0)	0x43, SYN_PT_COARSE_MAP_WRITE_FAIL An attempt was made to write the coarse map for a target with an error received with the write response.	0x12	Target, bits[29:16]
Software error (0)	0x44, SYN_PT_TABLE_READ_FAIL 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[29:16] ID, bits[15:0]
Software error (0)	0x45, SYN_PT_TABLE_WRITE_FAIL 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[29:16] ID, bits[15:0]
Software error (0)	0x46, SYN_PT_SUB_TABLE_READ_FAIL An attempt was made to read a subblock of interrupts from a Pending table, where an error response was received with the data.	0x12	Target, bits[29:16] ID, bits[15:0]
Software error (0)	0x47, SYN_PT_TABLE_WRITE_FAIL_BYTE 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[29:16] ID, bits[15:0]
Correctable SPI RAM errors (1)	0x00	0x7	See Table 4-9: SPI RAM errors, records 1-2 on page 76
Uncorrectable SPI RAM errors (2)	0x00	0x7	
Correctable SGI RAM errors (3)	0x00	0x7	See Table 4-10: SGI RAM errors, records 3-4 on page 78
Uncorrectable SGI RAM errors (4)	0x00	0x7	
Reserved (5)	-	-	-
Reserved (6)	-	-	-
Correctable PPI RAM errors (7)	0x00	0x7	See Table 4-11: PPI RAM errors, records 7-8 on page 79

Record	GICT_ERR<n>STATUS.IERR (syndrome)	GICT_ERR<n>STATUS .SERR	Value and description of GICT_ERR<n>MISC0.Data (other bits RES0) Always packed from 0 (lowest = 0)
Uncorrectable PPI RAM errors (8)	0x00	0x7	
Correctable LPI RAM errors (9)	0x00	0x7	See Table 4-12: LPI RAM errors, records 9-10 on page 80
Uncorrectable LPI RAM errors (10)	0x00	0x7	
Correctable error from ITS RAM (11)	0x00	0x6	See Table 4-13: ITS RAM errors, records 11-12 on page 80
Uncorrectable error from ITS RAM (12)	0x00	0x6	
Command or translation error in ITS (13+)	**0x00** architectural **0x01** non-architectural	0x1	ITS 24-bit syndrome. See 4.15.6.7 ITS command and translation error records 13 and beyond on page 81.

5.8.6 GICT_ERR<n>MISC1, Error Record Miscellaneous Register 1

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-600AE only supports a single MISC1 register, so n = 10, and therefore this register is identified as GICT_ERR10MISC1.

Configurations

This register is available in all configurations.

Attributes

Width 64-bit
Functional group See 5.8 GICT register summary on page 147 for the address offset, type, and reset value of this register.

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.

Bit descriptions

Figure 5-41: GICT_ERR10MISC1 bit assignments

Table 5-52: GICT_ERR10MISC1 bit descriptions

Bits	Name	Description
[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 the configuration of GIC-600AE.

5.8.7 GICT_ERRGSR, Error Group Status Register

This register shows the status of the GIC-600AE Armv8.2 RAS architecture-compliant error records for correctable and uncorrectable RAM ECC errors, ITS command and translation errors, and uncorrectable software errors.

Configurations

This register is available in all configurations.

Attributes

Width 64-bit
Functional group See 5.8 GICT register summary on page 147 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-42: GICT_ERRGSR bit assignments

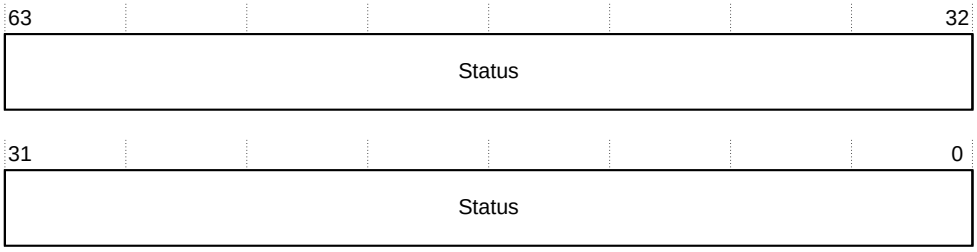


Table 5-53: GICT_ERRGSR bit descriptions

Bits	Name	Description
[n]	Status	Indicates the status of error record n, where n is 0-13+ depending on the configuration: **0** The error record is not reporting any errors **1** The error record is reporting one or more errors

5.8.8 GICT_ERRIRQCR<n>, Error Interrupt Configuration Registers

GICT_ERRIRQCR0 controls which SPI is generated when a fault handling interrupt occurs.
GICT_ERRIRQCR1 controls which SPI is generated when an error recovery interrupt occurs.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.8 GICT register summary on page 147 for the address offset, type, and reset value of this register.

Usage constraints

If GICD_SAC.GICTNS == 0, then only Secure software can access the functions of this register.

Bit descriptions

Figure 5-43: GICT_ERRIRQCR<n> bit assignments

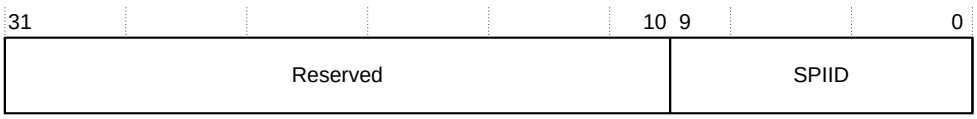


Table 5-54: GICT_ERRIRQCR<n> bit descriptions

Bits	Name	Description
[31:10]	-	Reserved, RAZ

Bits	Name	Description
[9:0]	SPIID	Sets the SPI ID that the GIC generates when a fault handling interrupt occurs (<n>==0) or when an error recovery interrupt occurs(<n>==1). If the value is less than 32, out of range, or not owned on chip for multichip configurations, the register updates to 0 and no internal delivery occurs. Set to 0, when the interrupt routes externally to a core that does not receive interrupts directly from the GIC such as a central system control processor. 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. We recommend 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.

5.8.9 GICT_DEVID, Device Configuration register

This register returns information about the configuration of the GIC-600AE GICT such as whether an LPI or ITS is available.

Note

GICT_DEVID was previously known as GICT_ERRIDR.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.8 GICT register summary on page 147 for the address offset, type, and reset value of this register.

Usage constraints

If GICD_SAC.GICTNS == 0, then only Secure software can read this register.

Bit descriptions

Figure 5-44: GICT_DEVID bit assignments

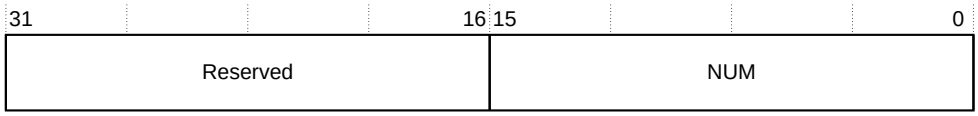


Table 5-55: GICT_DEVID bit descriptions

Bits	Name	Description
[31:16]	-	Reserved, RAZ
[15:0]	NUM	Identifies the device configuration: **9** No LPI available **11** LPI available but no ITS **14** LPI available and 1 × ITS **15** LPI available and 2 × ITS **16** LPI available and 3 × ITS

5.8.10 GICT_PIDR2, Peripheral ID2 Register

This register returns byte[2] of the peripheral ID. The GICT_PIDR2 register is part of the set of trace and debug peripheral identification registers.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.8 GICT register summary on page 147 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-45: GICT_PIDR2 bit assignments

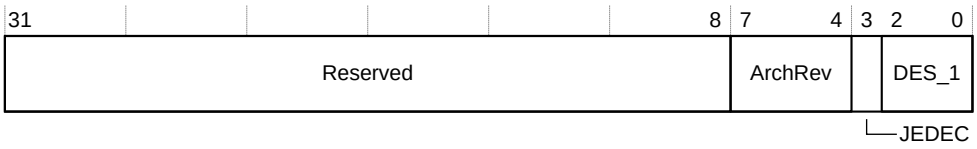


Table 5-56: GICT_PIDR2 bit descriptions

Bits	Name	Description
[31:8]	-	Reserved, RAZ
[7:4]	ArchRev	Identifies the version of the GIC architecture with which the trace and debug block complies: **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[7:4].

5.9 GICP register summary

The GIC-600AE Performance Monitoring Unit functions are controlled through registers that are identified with the prefix GICP.

The GICD_SAC.GICPNS bit controls whether Non-secure software can access the GICP registers.

Table 5-57: GICP register summary

Offset	Name	Type	Reset	Width	Description	Architecture defined?
0x000 + (n × 4)	GICP_EVCNTRn	RW	Unknown	32	Event Counter Registers, n = 0-4	No
0x400 + (n × 4)	GICP_EVTYPERn	RW	Unknown	32	Event Type Configuration Registers, n = 0-4	No
0x600 + (n × 4)	GICP_SVRn	RO	Unknown	32	Shadow Value Registers, n = 0-4	No
0xA00 + (n × 4)	GICP_FRn	RW	Unknown	32	Filter Registers, n = 0-4	No
0xC00	GICP_CNTENSET0	RW	0x0	64	Counter Enable Set Register	No
0xC20	GICP_CNTENCLR0	RW	0x0	64	Counter Enable Clear Register	No
0xC40	GICP_INTENSET0	RW	0x0	64	Interrupt Contribution Enable Set Register 0	No
0xC60	GICP_INTENCLR0	RW	0x0	64	Interrupt Contribution Enable Clear Register 0	No
0xC80	GICP_OVSCLR0	RW	0x0	64	Overflow Status Clear Register 0	No
0xCC0	GICP_OVSSET0	RW	0x0	64	Overflow Status Set Register 0	No
0xD88	GICP_CAPR	WO	-	32	Counter Shadow Value Capture Register	No
0xE00	GICP_CFGR	RO	0x401F04	32	Configuration Information Register	No
0xE04	GICP_CR	RW	0x0	32	Control Register	No
0xE50	GICP_IRQCR	RW	0x0	32	Interrupt Configuration Register	No
0xFB8	GICP_PMAUTHSTATUS	RO	0x088	32	Authentication Status register	No
0xFBC	GICP_PMDEVARCH	RO	0x47702A56	32	Device Architecture register	No
0xFCC	GICP_PMDEVTYPE	RO	0x56	32	Device Type register	No
0xFD0	GICP_PIDR4	RO	0x44	32	Peripheral ID 4 Register	No
0xFD4	GICP_PIDR5	RO	0x00	32	Peripheral ID 5 Register	No
0xFD8	GICP_PIDR6	RO	0x00	32	Peripheral ID 6 Register	No
0xFDC	GICP_PIDR7	RO	0x00	32	Peripheral ID 7 Register	No
0xFE0	GICP_PIDR0	RO	0x96	32	Peripheral ID 0 Register	No
0xFE4	GICP_PIDR1	RO	0xB4	32	Peripheral ID 1 Register	No
0xFE8	GICP_PIDR2	RO	0x3B	32	Peripheral ID 2 Register	No
0xFEC	GICP_PIDR3	RO	0x00	32	Peripheral ID 3 Register	No
0xFF0	GICP_CIDR0	RO	0x0D	32	Component ID 0 Register	No
0xFF4	GICP_CIDR1	RO	0xF0	32	Component ID 1 Register	No

Offset	Name	Type	Reset	Width	Description	Architecture defined?
0xFF8	GICP_CIDR2	RO	0x05	32	Component ID 2 Register	No
0xFFC	GICP_CIDR3	RO	0xB1	32	Component ID 3 Register	No

5.9.1 GICP_EVCNTRn, Event Counter Registers

These registers contain the values of event counter n. The GIC-600AE supports five counters, n = 0-4.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.9 GICP register summary on page 163 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-46: GICP_EVCNTRn bit assignments

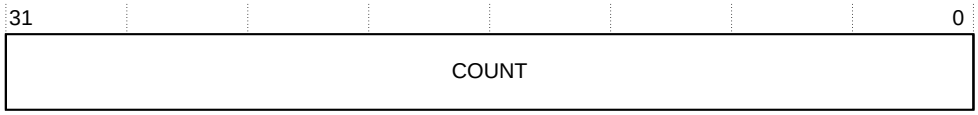


Table 5-58: GICP_EVCNTRn bit descriptions

Bits	Name	Description
[31:0]	COUNT	Counter value. If the counter is enabled, the counter value increments when an event matching GICP_EVTYPERn.EVENT occurs.

5.9.2 GICP_EVTYPERn, Event Type Configuration Registers

These registers configure which events that event counter n counts. The GIC-600AE supports five counters, n = 0-4.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.9 GICP register summary on page 163 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-47: GICP_EVTYPERn bit assignments

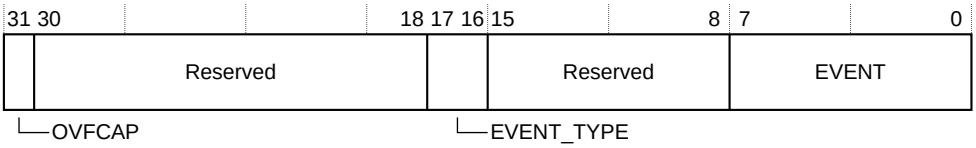


Table 5-59: GICP_EVTYPERn bit descriptions

Bits	Name	Description
[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: **0b00** Count events **0b10** MaximumEvent **0b01, 0b11** Reserved
[15:8]	-	Reserved
[7:0]	EVENT	Event identifier. See Table 5-60: GICP_EVTYPERn.EVENT field encoding on page 165. All events reset to an unknown value. Registers corresponding to unimplemented counters are RES0.

The following table shows the events that the GIC can count.

Table 5-60: GICP_EVTYPERn.EVENT field encoding

EventID	Event	Description	Filter
0x0	CLK	Clock cycle	None
0x1	CLK_NG	Clock cycle that prevents Q-Channel clock gating	None
0x2-0x3	-	Reserved	-
0x4	DN_MSG	Downstream message to core excluding PPIs	Target
0x5	DN_SET	Set to core SPIs and LPIs	Target/ID range
0x6	DN_SET1OFN	Set to core, which is a 1 of N interrupt	Target/ID range
0x7	-	Reserved	-
0x8	UP_MSG	Upstream message from core	Target

EventID	Event	Description	Filter
0x9	UP_ACT	Upstream activate	Target/ID range
0xA	UP_REL	Upstream release	Target/ID range
0xB	UP_ACTREL	Upstream activate or release.	Target/ID range
0xC	UP_SET_COMP	A set followed by an activate. This event counts the set and then decrements on release.	Target/ID range
0xD	UP_DEACT	Upstream deactivate. SPIs only.	Target/ID range
0x10	SGI_BRD	Broadcast SGI messages. Target = source.	Target/ID range
0x11	SGI_TAR	Targeted SGI messages. Target = source.	Target/ID range
0x12	SGI_ALL	All SGI messages. Target = source.	Target/ID range
0x13	SGI_ACC	Accepted SGI. Target = source.	Target/ID range
0x14	SGI_BRD_CC_IN	Broadcast SGI message from cross-chip	ID range/Chip
0x15	SGI_TAR_CC_IN	Targeted SGI message from cross-chip	ID range/Chip
0x16	SGI_TAR_CC_OUT	Targeted SGI sent cross-chip	Chip/ID range
0x20	ITS_NLL_LPI	Incoming LPI	Target/ID range/ITS
0x21	ITS_LL_LPI	Incoming low latency LPI	Target/ID range/ITS
0x22	ITS_LPI	Incoming LPI (or low latency)	Target/ID range/ITS
0x23	ITS_LPI_CMD	Incoming LPI command	Target/ID range/ITS
0x24	ITS_DID_MISS	Number of DeviceID cache misses	Target/ID range/ITS
0x25	ITS_VID_MISS	Number of EventID cache misses	Target/ID range/ITS
0x26	ITS_COL_MISS	Number of Collection cache misses	Target/ID range/ITS
0x27	ITS_LAT	Latency of the ITS transaction	Target/ID range/ITS
0x28	ITS_MPFA	Number of free slots during translation	Target/ID range/ITS
0x29	LPI_CC_OUT	LPI sent cross-chip	ID range/Chip
0x2A	LPI_CMD_CC_OUT	LPI command sent cross-chip	ID range/Chip
0x2B	LPI_CC_IN	LPI coming in from cross-chip	Target/ID range/Chip
0x2C	LPI_CMD_CC_IN	LPI command coming in from cross-chip	Target/ID range/Chip
0x30	LPI_OWN_STORED	LPI stored in own location. Prevents clock gating and Q-Channel clock gating.	-

EventID	Event	Description	Filter
0x31	LPI_OOL_STORED	LPI stored out of location. Prevents clock gating and Q-Channel clock gating.	-
0x32	LPI_HIT_EN	LPI property read cache hit enabled. Uses the filter from counter 0 only.	Target/ID range
0x33	LPI_HIT_DIS	LPI property read cache hit disabled. Uses the filter from counter 0 only.	Target/ID range
0x34	LPI_HIT	LPI property read cache hit. Uses the filter from counter 0 only.	Target/ID range
0x35	LPI_MATCH	LPI coalesced. Uses the filter from counter 0 only.	Target/ID range
0x36	LPI_FAS	Number of slots free on new LPI	None
0x37	LPI_PROP_EN	Enabled LPI property fetch. Uses the filter from counter 0.	Target/ID range
0x38	LPI_PROP_DIS	Disabled LPI property fetch. Uses the filter from counter 0.	Target/ID range
0x39	LPI_PROP	LPI property fetch. Uses the filter from counter 0.	Target/ID range
0x3A	LPI_COMP_INC_MERGE	Indicates that an LPI has completed. Uses the filter from counter 0.	Target/ID range
0x50	SPI_COL_MSG	New message from SPI Collator	ID range
0x51	SPI_ENABLED	SPI enabled (new SPI or register access if pending)	ID range
0x52	SPI_DISABLED	SPI disabled (new SPI that is disabled or register access if pending)	ID range
0x53	SPI_PENDING_SET	New SPI pending valid	ID range
0x54	SPI_PENDING_CLR	SPI pending bit cleared	ID range
0x55	SPI_MATCH	Collated edge-based SPI. Excludes collation in the SPI Collator.	ID range
0x57	SPI_CC_IN	SPI from remote chip	ID range/Chip
0x58	SPI_CC_OUT	SPI sent to remote chip	ID range/Chip
0x5A	SPI_CC_DEACT	SPI deactivate message sent	ID range/Chip
0x60	PT_IN_EN	Enabled interrupt written to Pending table	Target/ID range
0x61	PT_IN_DIS	Disabled interrupt written to Pending table	Target/ID range
0x62	PT_PRI	Priority of interrupt written to Pending table	Target/ID range
0x63	PT_IN	Interrupt written to Pending table	Target/ID range
0x64	PT_MATCH	Interrupt already set in Pending table	Target/ID range
0x65	PT_OUT_EN	Enabled interrupt taken out of Pending table (also covered PT_MATCH when enabled)	Target/ID range
0x66	PT_OUT_DIS	Disabled interrupt taken out of Pending table (also covered PT_MATCH when disabled)	Target/ID range
0x67	PT_OUT	Interrupt taken out of Pending table (also covered PT_MATCH)	Target/ID range
0x68	PT_BLOCK_SENT_CC	Pending table block that is sent as part of MOVALL	None
0x70	SPI_CC_LATENCY	SPIs outstanding	Chip

EventID	Event	Description	Filter
0x71	SPI_CC_LAT_WAIT	SPIs waiting to be sent	Chip
0x72	LPI_CC_LATENCY	LPIs outstanding	Chip
0x73	LPI_CC_LAT_WAIT	LPI waiting to be sent	Chip
0x74	SGI_CC_LATENCY	SGIs outstanding	Chip
0x75	SGI_LAT_WAIT	SGIs waiting to be sent	Chip
0x80	ACC	Counter(n – 1) – counter(n – 2) every cycle. Prevents clock gating and Q-Channel clock gating.	None
0x81	OFLOW	Overflow of counter n – 1. Overflow counters cannot count overflows of the counters that are using the OFLOW event.	None

5.9.3 GICP_SVRn, Shadow Value Registers

These registers contain the shadow value of event counter n. The GIC-600AE supports five counters, n = 0-4.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.9 GICP register summary on page 163 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-48: GICP_SVRn bit assignments

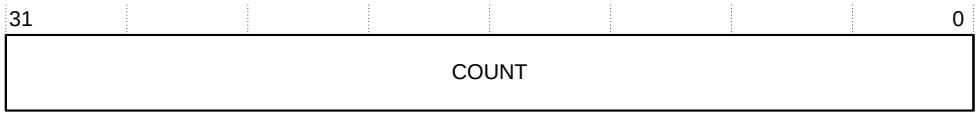


Table 5-61: GICP_SVRn bit descriptions

Bits	Name	Description
[31:0]	COUNT	Captured counter value. This field holds the captured counter values of the corresponding entry in GICP_EVCNTRn.

5.9.4 GICP_FRn, Filter Registers

These registers configure the filtering of event counter `n`. The GIC-600AE supports five counters, `n` = 0-4.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.9 GICP register summary on page 163 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-49: GICP_FRn bit assignments

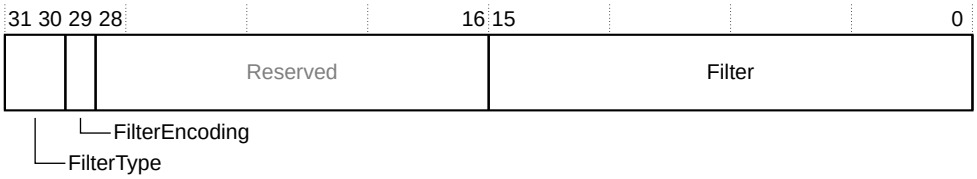


Table 5-62: GICP_FRn bit descriptions

Bits	Name	Description
[31:30]	FilterType	Filter type: **0b00** Filter on core **0b01** Filter on INTID **0b10** Filter on chip or ITS **0b11** Reserved, no effect
[29]	FilterEncoding	**0** Filter on range **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. When FilterEncoding == 1, counter `n` counts events that are only associated with an exact match of the FilterType. When FilterEncoding == 0, this field is encoded so that the first LSB that is zero, indicates the uppermost of a contiguous span of least significant FilterType content bits, that the GIC ignores for the purposes of matching. For example, setting Filter to: • `0b11110111_11110111` matches with values of `0b11110111_1111xxxx` for FilterType content • `0b11110111_11110110` matches with values of `0b11110111_1111011x` for FilterType content • `0b11110101_11111111` matches with values of `0b111101xx_xxxxxxxx` for FilterType content

5.9.5 GICP_CNTENSET0, Counter Enable Set Register 0

These registers contain the counter enables for each event counter. The GIC-600AE supports five event counters.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.9 GICP register summary on page 163 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-50: GICP_CNTENSET0 bit assignments

Table 5-63: GICP_CNTENSET0 bit descriptions

Bits	Name	Description
[31:5]	-	Reserved, RAZ
[4:0]	CNTEN	Counter enable. The CNTEN[n] bit is the enable for counter n. This field resets to an unknown value. Reads return the state of the counter enables. Writing: **Bit[n] = 1** Sets the enable for counter n. **Bit[n] = 0** No effect. To disable a counter, use GICP_CNTENCLR0. Counter n is enabled when CNTEN[n] == 1 and GICP_CR.E == 1.

5.9.6 GICP_CNTENCLR0, Counter Enable Clear Register 0

This register contains the counter disables for each event counter. The GIC-600AE supports five event counters.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.9 GICP register summary on page 163 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-51: GICP_CNTENCLR0 bit assignments

Table 5-64: GICP_CNTENCLR0 bit descriptions

Bits	Name	Description
[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. Reads return the state of the counter enables. Writing: **Bit[n] = 1** Disables counter n **Bit[n] = 0** No effect. To enable a counter, use GICP_CNTENSET0. Counter n is disabled when CNTEN[n] == 0 or GICP_CR.E == 0.

5.9.7 GICP_INTENSET0, Interrupt Contribution Enable Set Register 0

This register contains the set mechanism for the counter interrupt contribution enables. The GIC-600AE supports five counters, n = 0-4.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.9 GICP register summary on page 163 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-52: GICP_INTENSET0 bit assignments

Table 5-65: GICP_INTENSET0 bit descriptions

Bits	Name	Description
[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. Reads return the state of the interrupt enables. Writing: **Bit[n] = 1** Sets the interrupt enable for counter n **Bit[n] = 0** No effect. To disable a counter interrupt enable, use GICP_INTENCLR0. The interrupt enable for counter n is enabled when INTEN[n] == 1 and GICP_CR.E == 1. Overflow of counter n sets GICP_OVSSET0.OVS[n] to 1 and that triggers the PMU interrupt if INTEN[n] == 1.

5.9.8 GICP_INTENCLR0, Interrupt Contribution Enable Clear Register 0

This register contains the clear mechanism for the counter interrupt contribution enables. The GIC-600AE supports five counters, n = 0-4.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.9 GICP register summary on page 163 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-53: GICP_INTENCLR0 bit assignments

Table 5-66: GICP_INTENCLR0 bit descriptions

Bits	Name	Description
[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. Reads return the state of the interrupt enables. Writing: **Bit[n] = 1** Clears the interrupt enable for counter n **Bit[n] = 0** No effect. To set a counter interrupt enable, use GICP_INTENSET0.

5.9.9 GICP_OVSCLR0, Overflow Status Clear Register 0

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-600AE supports five counters, n = 0-4.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.9 GICP register summary on page 163 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-54: GICP_OVSCLR0 bit assignments

Table 5-67: GICP_OVSCLR0 bit descriptions

Bits	Name	Description
[31:5]	-	Reserved, RAZ

Bits	Name	Description
[4:0]	OVS	Overflow status. The OVS[n] bit is the overflow clear for counter n. This field resets to zero. Reads return the state of the overflow status bits. Writing: **Bit[n] = 1** Clears the overflow status for counter n **Bit[n] = 0** No effect. To set a counter overflow status, use GICP_OVSSET0. 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, sets the corresponding OVS bit. In addition, this event can trigger the PMU interrupt and cause a capture of the PMU counter values, see 5.9.2 GICP_EVTYPERn, Event Type Configuration Registers on page 164.

5.9.10 GICP_OVSSET0, Overflow Status Set Register 0

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-600AE supports five counters, n = 0-4.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.9 GICP register summary on page 163 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-55: GICP_OVSSET0 bit assignments

Table 5-68: GICP_OVSSET0 bit descriptions

Bits	Name	Description
[31:5]	-	Reserved, RAZ

Bits	Name	Description
[4:0]	OVS	Overflow status. The OVS[n] bit is the overflow set for counter n. This field resets to zero. Reads return the state of the overflow status bits. Writing: **Bit[n] = 1** Sets the overflow status for counter n **Bit[n] = 0** No effect. To clear a counter overflow status, use GICP_OVSCLR0. When the agent controlling the GIC-600AE sets an OVS bit, it is similar to an OVS bit being set because of a counter overflow. Setting the OVS bit triggers the overflow interrupt if it is enabled.

5.9.11 GICP_CAPR, Counter Shadow Value Capture Register

This register controls the counter shadow value capture mechanism.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.9 GICP register summary on page 163 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-56: GICP_CAPR bit assignments

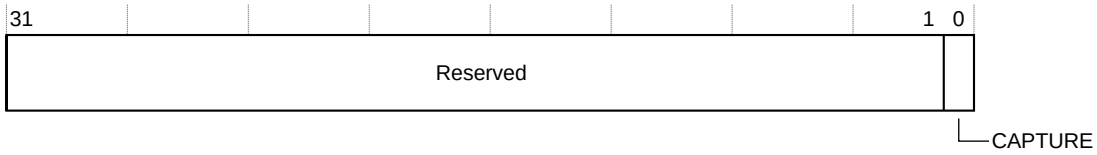


Table 5-69: GICP_CAPR bit descriptions

Bits	Name	Description	Type
[31:1]	-	Reserved	-
[0]	CAPTURE	A write of 1 triggers a capture of all values within the PMU into their respective shadow registers. A write of 0 has no effect. See Snapshot on page 68 for information about other snapshot event triggers.	WO

5.9.12 GICP_CFGR, Configuration Information Register

This register returns information about the PMU implementation.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.9 GICP register summary on page 163 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-57: GICP_CFGR bit assignments

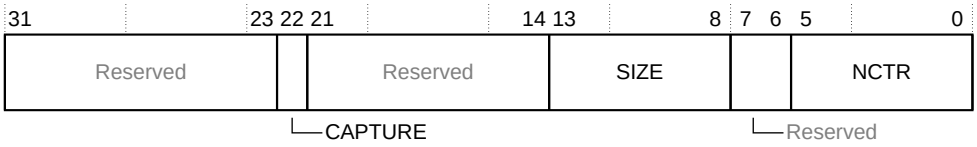


Table 5-70: GICP_CFGR bit descriptions

Bits	Name	Description
[31:23]	-	Reserved, RAZ
[22]	CAPTURE	Returns 1, to indicate that the GIC supports capture
[21:14]	-	Reserved, RAZ
[13:8]	SIZE	Returns 31, to indicate that the GIC supports 32-bit counters
[7:6]	-	Reserved, RAZ
[5:0]	NCTR	Returns 4, to indicate that the GIC provides five counters

5.9.13 GICP_CR, Control Register

This register controls whether all counters are enabled or disabled.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit

Functional group See 5.9 GICP register summary on page 163 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-58: GICP_CR bit assignments

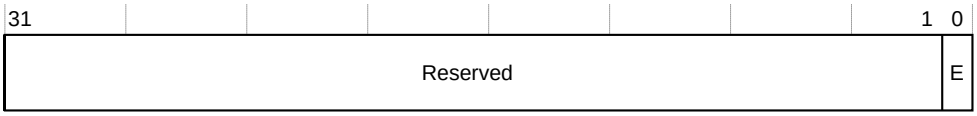


Table 5-71: GICP_CR bit descriptions

Bits	Name	Description
[31:1]	-	Reserved
[0]	E	Global counter enable: **0** No events are counted and the values in GICP_EVCNTRn do not change **1** The counters are enabled Resets to 0. This bit takes precedence over the GICP_CNTENSET0.CNTEN bits.

5.9.14 GICP_IRQCR, Interrupt Configuration Register

This register controls which SPI is generated when a PMU overflow interrupt occurs.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.9 GICP register summary on page 163 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-59: GICP_IRQCR bit assignments

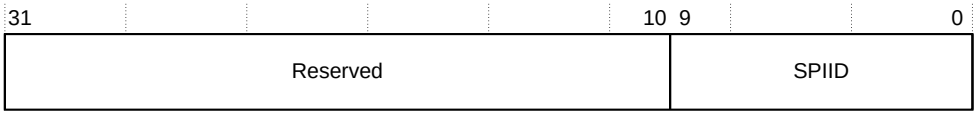


Table 5-72: GICP_IRQCR bit descriptions

Bits	Name	Description
[31:10]	-	Reserved, RAZ
[9:0]	SPIID	Sets the SPI ID that the GIC generates when a PMU overflow interrupt occurs. If the value is less than 32, out of range, or not owned on chip for multichip configurations, the register updates to 0 and no internal delivery occurs. Set to 0, when the interrupt routes externally to a core that does not receive interrupts directly from the GIC such as a central system control processor. Creates a level-triggered interrupt if it is owned on chip. Otherwise it behaves as a normal message-based SPI. 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. We recommend 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.

5.9.15 GICP_PIDR2, Peripheral ID2 Register

This register returns byte[2] of the peripheral ID. The GICP_PIDR2 register is part of the set of performance monitoring peripheral identification registers.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.9 GICP register summary on page 163 for the address offset, type, and reset value of this register.

Usage constraints

There are no usage constraints.

Bit descriptions

Figure 5-60: GICP_PIDR2 bit assignments

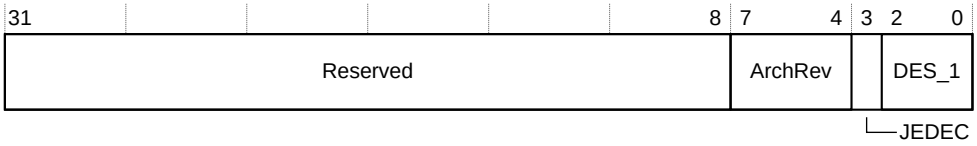


Table 5-73: GICP_PIDR2 bit descriptions

Bits	Name	Description
[31:8]	-	Reserved, RAZ
[7:4]	ArchRev	Identifies the version of the GIC architecture with which the PMU complies: **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[7:4].

5.10 FMU register summary

The GIC-600AE *Fault Management Unit* (FMU) functions are controlled through registers that are identified with the prefix FMU.

Unless otherwise stated in the accompanying FMU register descriptions:

- do not modify Reserved register bits
- ignore Reserved register bits on reads

Table 5-74: FMU register summary

Offset	Name	Type	Reset	Width	Description
0x000 + (n × 64)	FMU_ERR<n>FR	RO	0xA2	64	Error Record Feature Register
0x008 + (n × 64)	FMU_ERR<n>CTLR	RW	0x1	64	Error Record Control Register
0x010 + (n × 64)	FMU_ERR<n>STATUS	RW	0x30_0000	64	Error Record Primary Status register
0xE00	FMU_ERRGSR	RO	0x0	64	Error Group Status Register
0xEA0	FMU_KEY	RW	0x0	32	FMU Key register
0xEA4	FMU_PINGCTLR	RW	0x0	32	Ping Control Register
0xEA8	FMU_PINGNOW	RW	0x0	32	Ping Now register
0xEB0	FMU_SMEN	WO	0x0	32	Safety Mechanism Enable register
0xEB4	FMU_SMINJERR	WO	0x0	32	Safety Mechanism Inject Error register
0xEC0	FMU_PINGMASK	RW	0x0	64	Ping Mask register
0xF00	FMU_STATUS	RO	0x1	32	FMU Status register
0xFC8	FMU_ERRIDR	RO	0x2C	32	Error Record ID Register

5.10.1 FMU_ERR<n>FR, Error Record Feature Register

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

Configurations

This register is available in all configurations.

Attributes

Width 64-bit
Functional group See 5.10 FMU register summary on page 179 for the address offset, type, and reset value of this register.

Usage constraints

Only accessible by Secure accesses.

Bit descriptions

Figure 5-61: FMU_ERR<n>FR bit descriptions

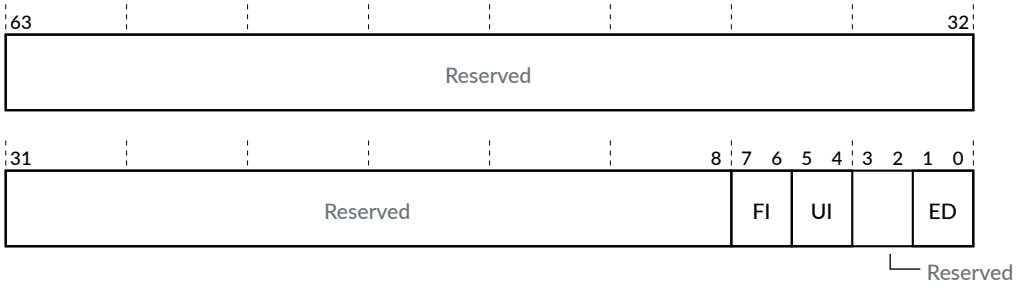


Table 5-75: FMU_ERR<n>FR bit assignments

Bits	Name	Description
[63:8]	-	Reserved, RAZ
[7:6]	FI	Fault handling interrupt. Returns: **0b10** Fault handling interrupt is supported and controllable using FMU_ERR<n>CTLR.FI
[5:4]	UI	Error recovery interrupt for uncorrected errors. Returns: **0b10** Error recovery interrupt is supported and controllable using FMU_ERR<n>CTLR.UI
[3:2]	-	Reserved, RAZ
[1:0]	ED	Error reporting and logging. Returns: **0b10** Error reporting and logging is controllable using FMU_ERR<n>CTLR.ED

5.10.2 FMU_ERR<n>CTLR, Error Record Control Register

This register controls which interrupt types are handled.

Configurations

This register is available in all configurations.

Attributes

Width 64-bit
Functional group See 5.10 FMU register summary on page 179 for the address offset, type, and reset value of this register.

Usage constraints

Only accessible by Secure accesses.

Bit descriptions

Figure 5-62: FMU_ERR<n>CTLR bit assignments

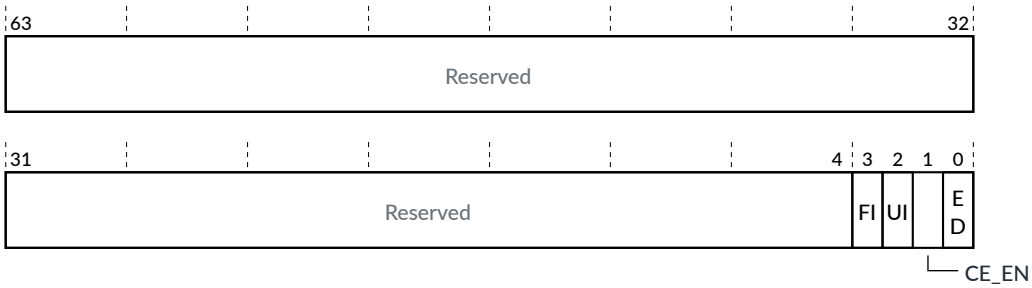


Table 5-76: FMU_ERR<n>CTLR bit descriptions

Bits	Name	Description
[63:4]	-	Reserved, RAZ
[3]	FI	Fault Handling Interrupt (FHI) enables. When set to 1, it enables the fault handling interrupt for all Corrected error events, and Uncorrected errors.
[2]	UI	Error Recovery Interrupt (ERI) enable. This bit controls whether an ERI is generated for all detected, logged (FMU_ERR<n>CTLR.ED == 1) errors that are reported through this error record as UEs. That is: • Correctable errors that are reported as uncorrectable (FMU_ERR<n>CTLR.CE_EN == 0) • Uncorrectable errors An error that is reported as a UE might generate both an ERI and an FHI.
[1]	CE_EN	Correctable error enable: **0** Treats correctable errors as uncorrectable errors (default) **1** Treats correctable errors and uncorrectable errors differently, and reports them separately
[0]	ED	Error reporting and logging enable: **0** Error reporting and logging is disabled for this record **1** Error reporting and logging is enabled for this record. This setting occurs at reset.

5.10.3 FMU_ERR<n>STATUS, Error Record Primary Status register

This register indicates information relating to the recorded errors. Software can write to this register to clear the error records that FMU_ERRGSR reports.

Configurations

This register is available in all configurations.

Attributes

Width 64-bit
Functional group See 5.10 FMU register summary on page 179 for the address offset, type, and reset value of this register. This register is only reset by the dbg_[<domain>]reset_n signal.

Usage constraints

- Only accessible by Secure accesses.
- After a write to this register, poll the FMU_STATUS register to ensure that the effect of the write is complete. Until the write takes effect, that is, FMU_STATUS.idle == 1 then:
  - The corresponding bit of FMU_ERRGSR might still report as 1.
  - Any interrupts caused by this record might still be asserted.
  - Any new error that occurs, is treated as a second error recording on top of this error, and causes an overflow to be set.
  - Any read of this register might return the old value, or if a new error has been recorded, then the newly recorded value.
- Do not write to an FMU_ERR<n>STATUS that corresponds to a powered-off block. See Power management on page 208.

Bit descriptions

Figure 5-63: FMU_ERR<n>STATUS bit assignments

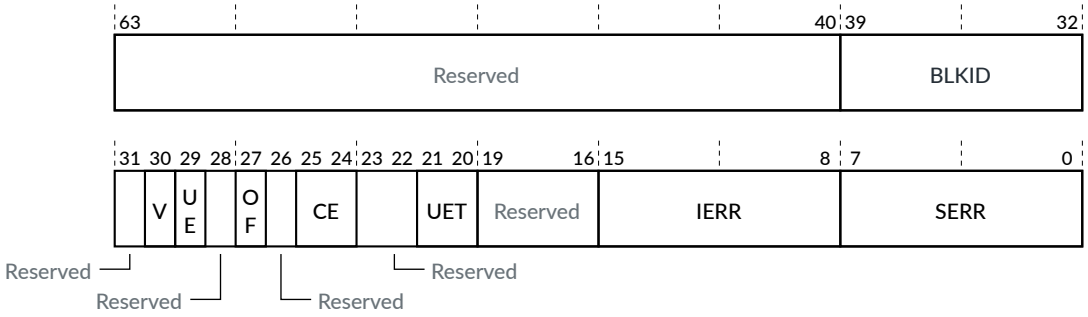


Table 5-77: FMU_ERR<n>STATUS bit descriptions

Bits	Name	Description	Type
[63:40]	-	Reserved, RAZ	-
[39:32]	BLKID	This field is only valid for error record 0 (GICD). Valid only when FMU_ERR<n>STATUS.V==1 and FMU_ERR<n>STATUS.IERR==19 (FMU ping ACK error). When there is a PING_ACK timeout error, this field indicates the block ID of the remote GIC block that caused the error. If software uses FMU_SMINJERR for error injection, this field is not updated when a PING_ACK timeout error is reported.	RO
[31]	-	Reserved, RAZ	-
[30]	V	Indicates if this register is valid: **0** FMU_ERR<n>STATUS is not valid **1** FMU_ERR<n>STATUS is valid. One or more errors are recorded. Write 1 to clear. When clearing this bit, FMU_ERR<n>STATUS.UE and FMU_ERR<n>STATUS.CE must also be cleared.	RW
[29]	UE	Uncorrected error bit. Write 1 to clear. When clearing this bit, FMU_ERR<n>STATUS.V must also be cleared. If FMU_ERR<n>STATUS.V is set to 0, this bit is not valid and reads unknown.	RW
[28]	-	Reserved, RAZ	-
[27]	OF	Record has overflowed. Write 1 to clear.	RW
[26]	-	Reserved, RAZ	-
[25:24]	CE	Corrected error bit: **0b00** No errors were corrected **0b10** One or more errors were corrected Write `0b10` or `0b11` to clear. If FMU_ERR<n>STATUS.V is set to 0, this field is not valid and reads unknown.	RW
[23:22]	-	Reserved, RAZ	-
[21:20]	UET	Uncorrected error type. **0b11** Uncorrected error records This field is not valid and reads unknown if either of the following conditions are true: • FMU_ERR<n>STATUS.V is set to 0 • FMU_ERR<n>STATUS.UE is set to 0	RO
[19:16]	-	Reserved, RAZ	-
[15:8]	IERR	Implementation-defined error code. See Table 6-2: Safety Mechanism IDs on page 199 for Safety Mechanism ID encodings. If FMU_ERR<n>STATUS.V is set to 0, this field is not valid and reads unknown.	RW in error record 0 and 13+, otherwise RES0
[7:0]	SERR	Architecturally defined primary error code. Returns information shown in Table 5-51: GICT_ERR<n>MISC0.Data field encoding on page 155.	RO in error record 0, otherwise RW

5.10.4 FMU_ERRGSR, Error Group Status Register

This register shows the status of the FMU error records.

Configurations

This register is available in all configurations.

Attributes

Width 64-bit
Functional group See 5.10 FMU register summary on page 179 for the address offset, type, and reset value of this register.

Usage constraints

Only accessible by Secure accesses.

Bit descriptions

Figure 5-64: FMU_ERRGSR bit assignments

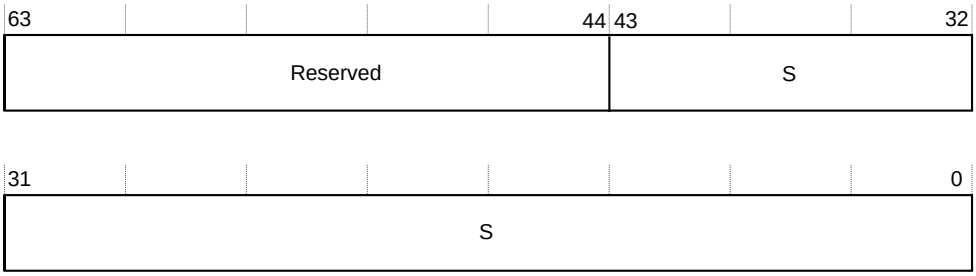


Table 5-78: FMU_ERRGSR bit descriptions

Bits	Name	Description
[63:44]	-	Reserved, RAZ
[43:0]	S	Indicates the status of error record n, where n is 0-13+ depending on the configuration: **0** The error record is not reporting any errors **1** The error record is reporting one or more errors

5.10.5 FMU_KEY, FMU Key register

This register receives the unlock key that is required for writes to FMU registers to be successful. This register reads as 0 if the FMU register file is locked.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.10 FMU register summary on page 179 for the address offset, type, and reset value of this register.

Usage constraints

Only accessible by Secure accesses.

Bit descriptions

Figure 5-65: FMU_KEY bit assignments

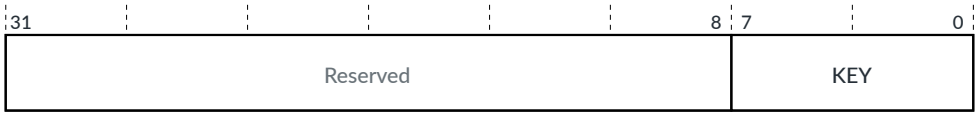


Table 5-79: FMU_KEY bit descriptions

Bits	Name	Description
[31:8]	-	Reserved, RAZ
[7:0]	KEY	Writing the correct key to this field enables the next write to any other writable FMU register to succeed. See 6.2.7 Lock and key mechanism on page 205.

5.10.6 FMU_PINGCTLR, Ping Control Register

This register configures the error ping timing interval.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.10 FMU register summary on page 179 for the address offset, type, and reset value of this register.

Usage constraints

Only accessible by Secure accesses.

Bit descriptions

Figure 5-66: FMU_PINGCTLR bit assignments

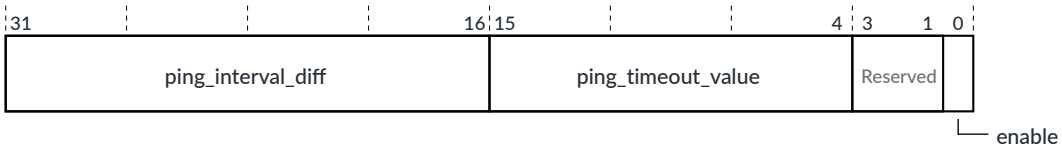


Table 5-80: FMU_PINGCTLR bit descriptions

Bits	Name	Description
[31:16]	ping_interval_diff	Equal to (ping_interval − ping_timeout_value) in GIC clock cycles. See Figure 6-3: Ping mechanism parameters on page 203 for more information. The minimum value supported is 4.
[15:4]	ping_timeout_value	Timeout threshold value for ping timeouts in GIC clock cycles. The minimum value supported is 20. The clock frequency difference and average network congestion must be considered when programming this field.
[3:1]	-	Reserved, RAZ
[0]	enabled	When set to 1, it enables the GICD background ping engine. The GICD sends ping messages to each remote GIC block, and expects a PING_ACK back within the specified timeout. If the PING_ACK is not received within the specified timeout, then the GICD records this situation as an error. The GICD sequentially moves to the next block and sends another ping message after ping_interval. If pings are enabled, then FMU_PINGMASK.ping_mask field must unmask at least one remote GIC block. See 6.2.6 Ping mechanisms on page 202 for more information.

5.10.7 FMU_PINGNOW, Ping Now register

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

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.10 FMU register summary on page 179 for the address offset, type, and reset value of this register.

Usage constraints

- Only accessible by Secure accesses.
- After a write to this register, poll FMU_STATUS.idle to ensure that the effect of the write is complete.
- Do not write to FMU_PINGNOW that corresponds to a powered-off block. See Power management on page 208.

Bit descriptions

Figure 5-67: FMU_PINGNOW bit assignments

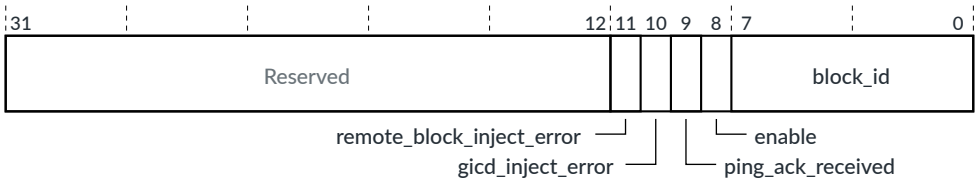


Table 5-81: FMU_PINGNOW bit descriptions

Bits	Name	Description
[31:12]	-	Reserved, RAZ
[11]	remote_block_inject_error	Set to 1 to inject an error on the PING_ACK response packet that is sent from the remote GIC block to the FMU. This action causes errors along the route of the PING_ACK through the interconnect. The presence of errors helps to confirm that the interconnect path from the specified remote GIC block to the FMU has been properly connected.
[10]	gicd_inject_error	Set to 1 to inject an error on the PING data packet that is sent from the FMU to the remote GIC block. This action causes errors along the route of the PING route through the interconnect. The presence of errors helps to confirm that the interconnect path from the FMU to the specified remote GIC block has been properly connected.
[9]	ping_ack_received	Indicates if a PING_ACK has been received: **0** PING_ACK has not been received **1** PING_ACK has been received from the GIC block that was pinged
[8]	enable	Ping enable: **0** Does not initiate a ping. Allows software to clear the status of this register without initiating another ping. **1** Initiates a ping to the GIC block that FMU_PINGNOW.block_id specifies
[7:0]	block_id	Block identifier. Sends a ping request to the specified GIC block. See Table 6-1: Error record block IDs on page 198 for block ID encodings.

5.10.8 FMU_SMEN, Safety Mechanism Enable register

This register enables or disables particular Safety Mechanisms inside a specified GIC block.

This feature cannot be used for the following FMU_SMEN field values:

- BLK = GICD, SMID = 0
- BLK = 3
- BLK = PPI, SMID = 0
- BLK = ITS, SMID = 0
- BLK = SPI Collator, SMID = 0
- BLK = Wake Request, SMID = 0

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.10 FMU register summary on page 179 for the address offset, type, and reset value of this register.

Usage constraints

- Only accessible by Secure accesses.
- After a write to this register, poll FMU_STATUS.idle to ensure that the effect of the write is complete.
- Do not write to FMU_SMEN and enable or disable a Safety Mechanism that corresponds to a powered-off block. See Power management on page 208.

Note

If a block is powered-off and then powered-on again, the enabled state of the Safety Mechanism returns to the default reset state.

Bit descriptions

Figure 5-68: FMU_SMEN bit assignments

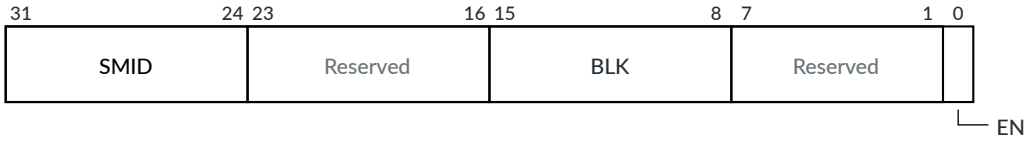


Table 5-82: FMU_SMEN bit descriptions

Bits	Name	Description
[31:24]	SMID	Safety Mechanism identifier. See Table 6-2: Safety Mechanism IDs on page 199 for Safety Mechanism ID encodings.
[23:16]	-	Reserved, RAZ
[15:8]	BLK	Block identifier. See Table 6-1: Error record block IDs on page 198 for block ID encodings.
[7:1]	-	Reserved, RAZ
[0]	EN	Safety Mechanism enable

5.10.9 FMU_SMINJERR, Safety Mechanism Inject Error register

This register injects one error into the specified Safety Mechanism inside a GIC block.

This feature cannot be used for the following FMU_SMINJERR field values:

- BLK = GICD, SMID = 0
- BLK = 3
- BLK = PPI, SMID = 0
- BLK = ITS, SMID = 0
- BLK = SPI Collator, SMID = 0
- BLK = Wake Request, SMID = 0

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.10 FMU register summary on page 179 for the address offset, type, and reset value of this register.

Usage constraints

- Only accessible by Secure accesses.
- After a write to this register, poll FMU_STATUS.idle to ensure that the effect of the write is complete.
- Do not write to FMU_SMINJERR and inject an error that corresponds to a powered-off block. See Power management on page 208.

Bit descriptions

Figure 5-69: FMU_SMINJERR bit assignments

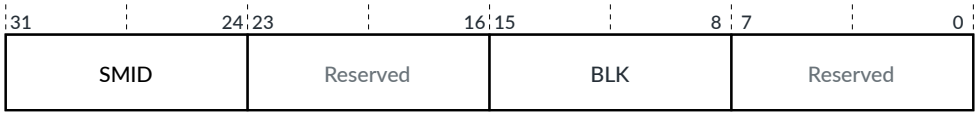


Table 5-83: FMU_SMINJERR bit descriptions

Bits	Name	Description
[31:24]	SMID	Safety Mechanism identifier. See Table 6-2: Safety Mechanism IDs on page 199 for Safety Mechanism ID encodings.
[23:16]	-	Reserved, RAZ
[15:8]	BLK	Block identifier. See Table 6-1: Error record block IDs on page 198 for block ID encodings.
[7:0]	-	Reserved, RAZ

5.10.10 FMU_PINGMASK, Ping Mask register

This register configures the ping mask.

Configurations

This register is available in all configurations.

Attributes

Width 64-bit
Functional group See 5.10 FMU register summary on page 179 for the address offset, type, and reset value of this register.

Usage constraints

- Only accessible by Secure accesses.
- Do not change FMU_PINGMASK when background ping is enabled, that is, when FMU_PINGCTLR.enable == 1.

Bit descriptions

Figure 5-70: FMU_PINGMASK bit assignments

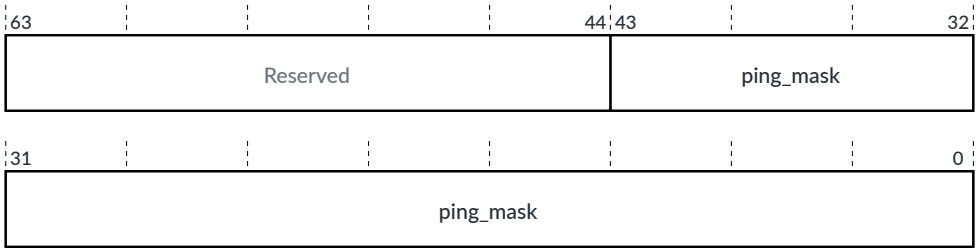


Table 5-84: FMU_PINGMASK bit descriptions

Bits	Name	Description
[63:44]	-	Reserved, RAZ
[43:0]	ping_mask	Ping mask. Bit position corresponds to the GIC block ID. See Table 6-1: Error record block IDs on page 198 for the block ID designations. To make the FMU skip a specific block while generating background ping messages, write a one to the corresponding bit. For unpopulated GIC blocks, corresponding bits have no effect. The same applies to bit[0], because the FMU does not ping GICD.

5.10.11 FMU_STATUS, FMU Status register

This register indicates whether the FMU is idle.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.10 FMU register summary on page 179 for the address offset, type, and reset value of this register.

Usage constraints

Only accessible by Secure accesses.

Bit descriptions

Figure 5-71: FMU_STATUS bit assignments

Table 5-85: FMU_STATUS bit descriptions

Bits	Name	Description
[31:1]	-	Reserved, RAZ
[0]	idle	Indicates if the FMU is idle: **0** FMU is busy processing the previous command **1** FMU is idle

5.10.12 FMU_ERRIDR, Error Record ID Register

This register defines the highest numbered index of the error records in this group.

Configurations

This register is available in all configurations.

Attributes

Width 32-bit
Functional group See 5.10 FMU register summary on page 179 for the address offset, type, and reset value of this register.

Usage constraints

Only accessible by Secure accesses.

Bit descriptions

Figure 5-72: FMU_ERRIDR bit assignments

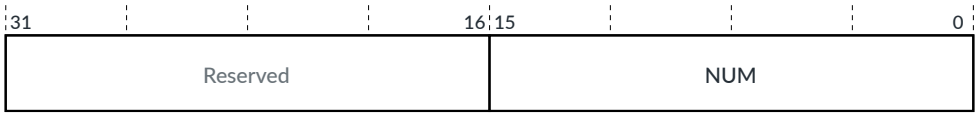


Table 5-86: FMU_ERRIDR bit descriptions

Bits	Name	Description
[31:16]	-	Reserved, RAZ
[15:0]	NUM	Highest numbered index of the error records in this group + 1

6. Functional Safety

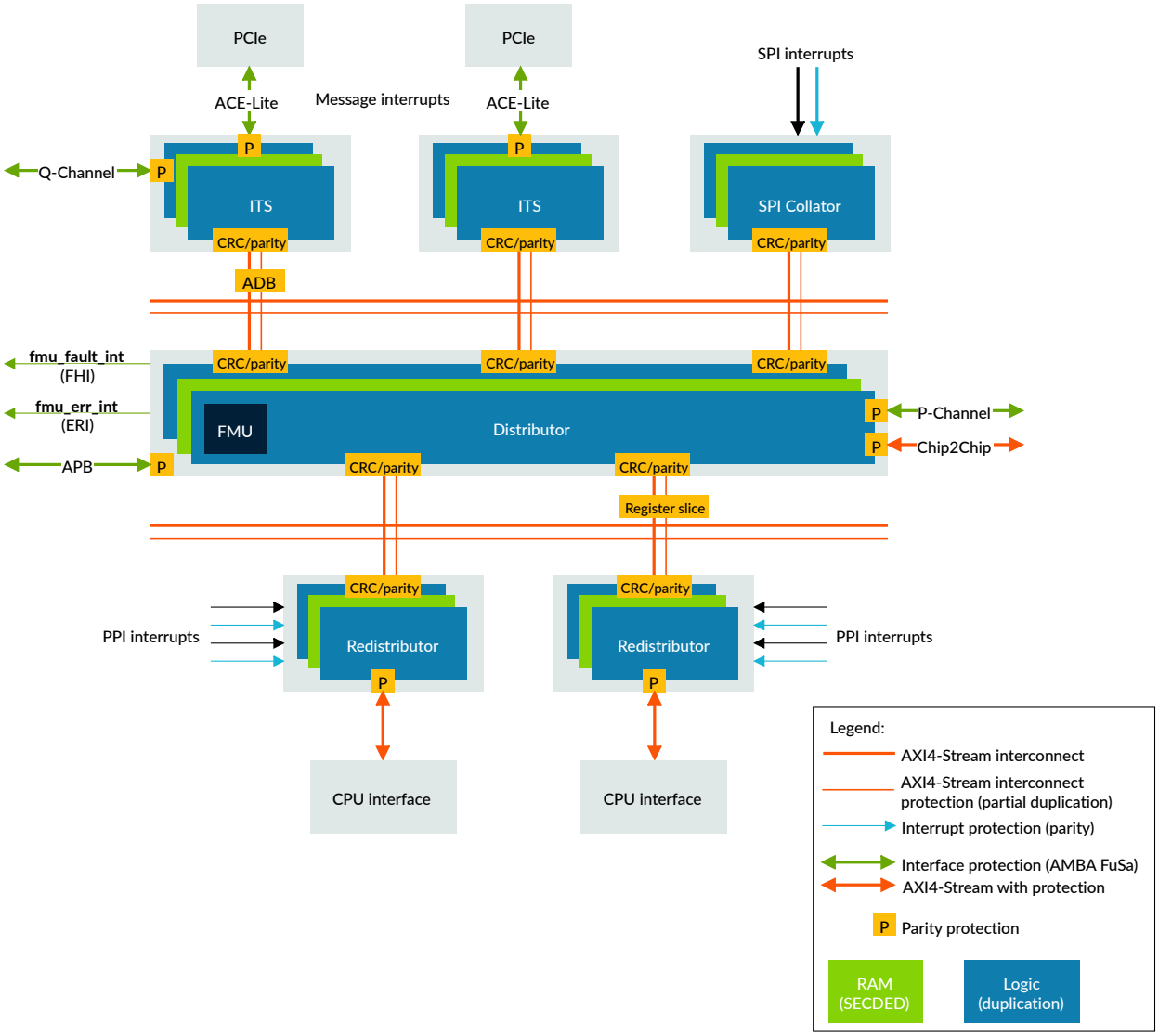
This chapter describes the *Functional Safety* (FuSa) detection features that are unique to GIC-600AE.

6.1 Safety Mechanism overview

GIC-600AE is a version of GIC-600 with FuSa detection features added. *Logic Equivalence Checking* (LEC) is used to ensure that the original GIC-600 logic is unchanged. All FuSa features are “bolted on” to the periphery of GIC-600 and do not alter the original GIC-600 functionality.

The following figure shows where the main Safety Mechanisms of GIC-600AE reside.

Figure 6-1: Safety Mechanism distribution



GIC-600AE contains the following FuSa Safety Mechanisms.

Lockstep logic protection

The logic is protected with duplicated logic running in lockstep.

RAM protection

The RAMs are shared between the duplicated blocks and are protected with SECDED ECC. The address is further protected with parity.

AXI4-Stream interconnect protection

The AXI4-Stream interconnect that connects the GIC blocks, is protected by end-to-end partial duplication. Partial duplication means that the primary interconnect is duplicated with

a compressed CRC representation of the payload data. Therefore, a wide primary payload is represented by a redundant payload of only 8 bits.

The following components are protected with partial duplication:

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

AMBA® external interface protection

All external interfaces are protected with AMBA® Parity Extension. AMBA® Parity Extension protects point-to-point connections consisting of wires and buffers only, and no gates. This protection includes the ACE-Lite, GIC Stream, *Cross-Chip* (CC), and APB external ports.

PPI and SPI source interrupt parity protection

The PPI and SPI interrupt input sources are protected with optional parity protection. There is one parity bit for each PPI and SPI input pin.

P-Channel and Q-Channel protection

The P-Channel and Q-Channel are protected by parity.


Note

- The P-Channel protection is for cross-chip functions, so it must protect the Distributor.
- Figure 6-1: Safety Mechanism distribution on page 194 shows Q-Channel protection that is enabled on only one ITS block. However, the Q-Channel protection can support any block that has a different CDC domain from the others.

Systematic fault watchdog

GIC-600AE contains a watchdog-based PING/ACK mechanism. This mechanism protects against systematic errors on the interconnect that connects the various GIC blocks. It works by engaging a hardware mechanism in the Distributor that pings each GIC block in a round-robin fashion and waits for a response. If the mechanism does not receive a response within the programmable timeout window, it reports a fault.

Clocks and resets

The clocks and resets are duplicated. The clocks operate with a temporal delay of two. That is, the primary logic operates two cycles ahead of the redundant logic.

Fault Management Unit

The *Fault Management Unit* (FMU) resides in the Distributor. It processes faults that are detected by the Safety Mechanisms from all blocks. The FMU records the fault syndrome in the Error Records and reports the fault using *Error Recovery Interrupt* (ERI) and *Fault Handling Interrupt* (FHI). It also provides fault injection and clearing for each Safety Mechanism. The FMU talks to an external Safety Island through the APB port. The APB port is added for FuSa purposes and does not exist on the GIC-600, the non-FuSa version.

Safety Mechanisms

For a detailed list of the Safety Mechanisms available in GIC-600AE, see the *Safety Mechanism descriptions* appendix in the *Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Safety Manual*.

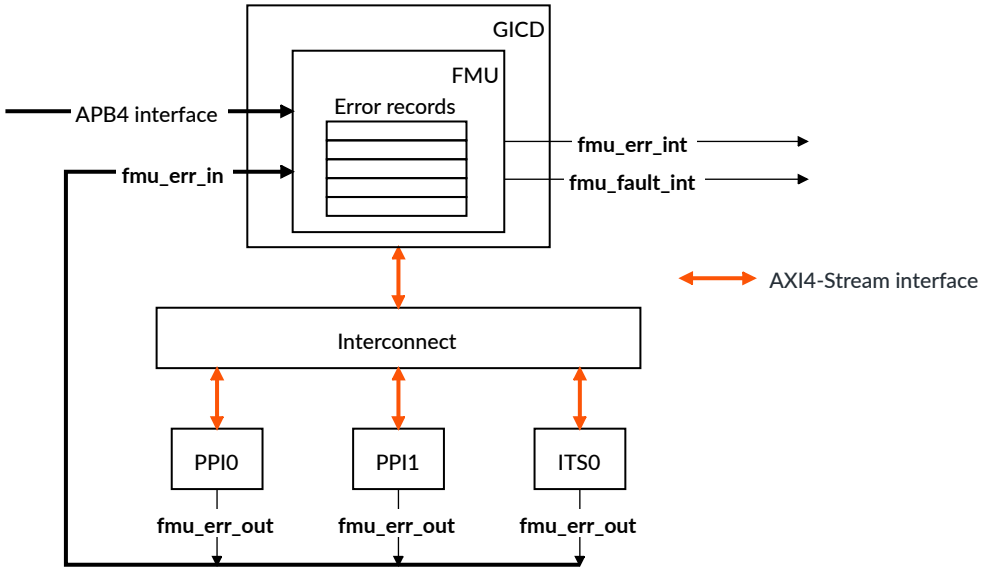
6.2 Fault Management Unit

The FMU is part of the GIC Distributor (GICD) component. It implements the following functionality in GIC-600AE:

- Uses a dedicated APB4 interface to access error records and other registers.
- Routes all errors to the Safety Island, if enabled.
- Provides software the means to enable or disable a Safety Mechanism within a GIC block.
- Receives error signaling from all Safety Mechanisms within other GIC blocks.
- Maintains error records for each GIC block, for software inspection and provides information on the source of the error.
- Retains error records across functional reset.
- Enables software error recovery testing by providing error injection capabilities in a Safety Mechanism.

The following figure shows the FMU and its interconnections.

Figure 6-2: FMU interconnections



6.2.1 FMU APB4 interface

The programmer view registers inside the FMU are accessible through an APB4 interface that is protected with AMBA parity extensions.

The APB interface width is 32 bits. Some of the FMU registers are 64 bits wide, so two APB accesses are necessary for reads or writes of those registers.

The APB4 port allows only Secure access to the FMU. To implement this access restriction, the pprot[1] signal is checked during an access. If the access fails the security check, the GIC does not use the pslverr signal to indicate this error condition, so the pslverr signal remains LOW.

6.2.2 Error signaling

This section describes how GIC blocks signal errors, and how the FMU reports these errors.

Error signaling from a GIC block to the FMU

GIC-600AE implements several *Safety Mechanisms* (SMs) in each GIC block to protect against random transient or permanent errors. Each Safety Mechanism sends an error signal to its GIC block. The GIC block then forwards the error signal to the central GIC Distributor using the existing AXI4-Stream interface.

In addition to reporting errors through the AXI4-Stream interconnect, each remote GIC block has an fmu_err_out output signal that indicates an actual uncorrected error within its block. Corrected errors never raise the fmu_err_out signal, even if configured to report as uncorrected. See 6.2.8 Correctable Error enable on page 206. The fmu_err_out signal must connect to the fmu_err_in input signal of the GICD to provide a redundant path for error signaling from the remote GIC block to the FMU residing in the GICD. The remote GIC block keeps the fmu_err_in signal asserted until the error recovery software clears the error.

Error signaling by the FMU

When a Safety Mechanism detects an error, it forwards the error to the FMU. If the FMU is enabled, it signals the error to the entire system using the error interrupt signals. These signals are:

- Error recovery interrupt, fmu_err_int (ERI) signal
- Fault handling interrupt, fmu_fault_int (FHI) signal

Error reporting through ERI or FHI is enabled by the FMU_ERR<n>CTLR register. The ERI and FHI interrupts are disabled on reset, so they must be enabled in the boot-up routine.

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

6.2.3 Error record format

The FMU contains one error record for each GIC block.

GIC-600AE faults are recorded in error records.

The error record registers are accessible through a separate APB interface on the GICD. We expect that there is a separate reset (Cold reset), so that the error record retains its state even when the GIC block is being reset.

The following table lists the block IDs for each GIC block.

Table 6-1: Error record block IDs

Block ID	GIC block
0	GICD
1	SPI Collator
2	Wake Request
3	Reserved
4	ITS0
5	ITS1
6	ITS2
7	ITS3
8	ITS4
9	ITS5
10	ITS6
11	ITS7
12	PPI0
13	PPI1
14	PPI2
15	PPI3
16	PPI4
17	PPI5
18	PPI6
19	PPI7
20	PPI8
21	PPI9
22	PPI10
23	PPI11
24	PPI12
25	PPI13
26	PPI14
27	PPI15
28	PPI16

Block ID	GIC block
29	PPI17
30	PPI18
31	PPI19
32	PPI20
33	PPI21
34	PPI22
35	PPI23
36	PPI24
37	PPI25
38	PPI26
39	PPI27
40	PPI28
41	PPI29
42	PPI30
43	PPI31

GIC-600AE supports a maximum of 32 PPI blocks and 8 ITS blocks.


Note

For unsupported ITS or PPI blocks, the error record registers become RAZ.

6.2.4 FMU reset

When the FMU reports multiple uncorrectable errors, the error recovery procedure might require the GIC to be reset. To facilitate this situation, the FMU operates on a dbg_reset_n reset signal.

This reset differs from the GIC functional reset, reset_n signal. It allows the FMU to retain error records across GIC functional reset.

6.2.5 Safety Mechanism IDs

The following table lists the IDs for each Safety Mechanism inside each GIC block.

Table 6-2: Safety Mechanism IDs

GIC block	Safety Mechanism ID	Description
GICD	0	Reserved
	1	GICD dual lock-step error
	2	GICD ACE-Lite subordinate interface error
	3	GICD→PPI AXI4-Stream interface error

GIC block	Safety Mechanism ID	Description
	4	GICD→ITS AXI4-Stream interface error
	5	GICD→SPI Collator AXI4-Stream interface error
	6	GICD ACE-Lite manager interface error
	7	SPI RAM DED error
	8	SGI RAM DED error
	9	Reserved
GICD	10	LPI RAM DED error
	11	GICD→remote GICD AXI4-Stream interface error
	12	GICD Q-Channel interface error
	13	GICD P-Channel interface error
	14	SPI RAM address decode error
	15	SGI RAM address decode error
	16	Reserved
	17	LPI RAM address decode error
	18	FMU dual lock-step error
	19	FMU ping ACK error
GICD	20	FMU APB parity error
	21	GICD→Wake Request AXI4-Stream interface error
	22	GICD PageOffset or chip ID error
	23	MBIST REQ error. This Safety Mechanism is disabled by default.
	24	SPI RAM SEC error
	25	SGI RAM SEC error
	26	Reserved
	27	LPI RAM SEC error
	28	User custom SM0 error
	29	User custom SM1 error
GICD	30	GICD→ITS monolithic switch error
	31	GICD→ITS Q-Channel interface error
	32	GICD→ITS monolithic interface error
	33	GICD FMU ClkGate override. This Safety Mechanism is disabled by default.
PPI	0	Reserved
	1	PPI dual lock-step error
	2	PPI→GICD AXI4-Stream interface error
	3	PPI→CPU interface AXI4-Stream interface error
	4	PPI Q-Channel interface error
	5	PPI RAM DED error
	6	PPI RAM address decode error
	7	PPI RAM SEC error
	8	PPI User0 SM

GIC block	Safety Mechanism ID	Description
	9	PPI User1 SM
	10	MBIST REQ error. This Safety Mechanism is disabled by default.
	11	PPI interrupt parity protection error
	12	PPI FMU ClkGate override. This Safety Mechanism is disabled by default.
ITS	0	Reserved
	1	ITS dual lock-step error
	2	ITS→GICD AXI4-Stream interface error
	3	ITS ACE-Lite subordinate interface error
	4	ITS ACE-Lite manager interface error
	5	ITS Q-Channel interface error
	6	ITS RAM DED error
	7	ITS RAM address decode error
	8	Bypass ACE switch error
	9	ITS RAM SEC error
	10	ITS User0 SM
	11	ITS User1 SM
	12	ITS→GICD monolithic interface error
	13	MBIST REQ error. This Safety Mechanism is disabled by default.
	14	ITS FMU ClkGate override. This Safety Mechanism is disabled by default.
SPI Collator	0	Reserved
	1	SPI Collator dual lock-step error
	2	SPI Collator→GICD AXI4-Stream interface error
	3	SPI Collator Q-Channel interface error
	4	SPI Collator Q-Channel clock error
	5	SPI interrupt parity error
Wake Request	0	Reserved
	1	Wake Request dual lock-step error
	2	Wake Request→GICD AXI4-Stream interface error

The SMID value 0 for error record[N] indicates that the FMU has detected an uncorrected error in the corresponding remote GIC block (PPI, ITS, SPI Collator, or Wake Request) as indicated by the fmu_err_out/fmu_err_in signals. The Safety Mechanism that reports this error has still not been determined. The Safety Mechanism that reports the error is updated after the Safety Mechanism in the remote GIC block sends this information over the DTI interface to the FMU in the GICD, and then this information is updated in the FMU_ERR<n>STATUS.IERR field.

If a software read of FMU_ERR<n>STATUS.IERR returns SMID:0, then we expect the software to read this register again. If repeat reads of IERR always return SMID:0, then it might indicate that the AXI4-Stream interconnect is broken, possibly due to a permanent fault, and is unable to receive

messages. The error recovery software does not have the SM information from the remote GIC block that had this fault, so it must perform error recovery by resetting that remote GIC block and the AXI4-Stream interconnect components.

6.2.5.1 Enabling or disabling a Safety Mechanism

All Safety Mechanisms are enabled on reset, except for the MBIST REQ Safety Mechanism.

To enable or disable a Safety Mechanism, write to the FMU_SMEN register. FMU_SMEN.BLK selects the GIC block, and FMU_SMEN.SMID selects the specific Safety Mechanism in the GIC block to be enabled or disabled.

The following P-Channel and Q-Channel SMs cannot be disabled through the FMU_SMEN register:

- GICD SMs 12, 13, and 31
- PPI SM 4
- ITS SM 5
- SPI SM 3

These SMs must be disabled using design time parameters or tie-offs. For more information, see 6.10 P-Channel and Q-Channel protection on page 232.

MBIST REQ SMs are not enabled on reset, and must be enabled after reset. For more information, see 6.13 DFT protection on page 247.

6.2.5.2 Injecting an error in a Safety Mechanism

To inject an error into a Safety Mechanism, write to the FMU_SMINJERR register.

The FMU_SMINJERR.BLK field specifies the GIC block, and the FMU_SMINJERR.SMID field specifies the SM into which to inject the error.

FMU_STATUS.idle protects the FMU_SMINJERR register. See FMU idle on page 208.

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

By introducing error through the software, the error injection feature can be used to test the software error recovery handler.


Note

The ClkGate override Safety Mechanisms do not support error injection.

6.2.6 Ping mechanisms

The FMU provides background ping and directed ping mechanisms.

Background ping

The background ping mechanism can help identify the following issues:

- Connectivity issue between remote GIC blocks and the GICD
- Systematic issue in the network that is causing misrouting of messages
- Congestion in the network that exceeds the programmed ping_timeout_value
- Permanent deadlock caused by valid and ready signals that are stuck LOW

The GICD sends a ping message over the AXI4-Stream network to a remote GIC block, one at a time. It starts a timer and waits for the PING_ACK message from the GIC block. If the PING_ACK message is not received within the expected interval, the FMU indicates a PING_ACK timeout error. The FMU repeats this process for each GIC block.

The FMU sends ping messages in the following sequence, which repeats until background pings are disabled:

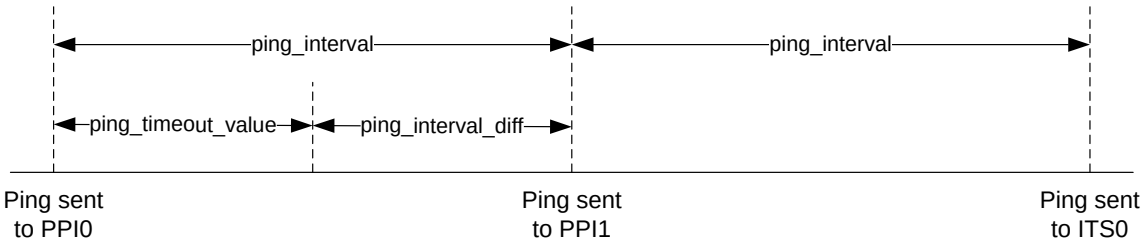
1. PPI0 through PPI<ppi_count−1>
2. ITS0 through ITS<its_count−1>
3. SPI Collator
4. Wake Request

Tip

To skip a particular GIC block in the sequence, write to the FMU_PINGMASK register.

The following figure shows the relationship between the ping mechanism parameters.

Figure 6-3: Ping mechanism parameters



FMU_PINGCTLR controls the value of ping_timeout_value, which defines the ping timeout in FMU clock cycles.

ping_interval defines the interval at which the FMU pings the next remote block. As Figure 6-3: Ping mechanism parameters on page 203 shows, ping_interval == ping_timeout_value + ping_interval_diff.

To enable or disable the background ping mechanism, write to FMU_PINGCTLR.enable. When background ping is enabled, the FMU_PINGCTLR[31:1] bits must not change. Software must disable background ping, before it changes the values in FMU_PINGCTLR. It is acceptable to program FMU_PINGCTLR and enable background ping at the same time.

When programming the ping_timeout_value in the FMU_PINGCTLR register, you must account for the following:

- Round-trip ping latency
- Concurrent GICD request traffic. Any concurrent GICD requests can delay transmission of the ping.
- Clock domain ratios. For example, if the FMU/GICD domain clock is running faster than a remote domain clock, you must increase ping_timeout_value.

If the FMU indicates a PING_ACK timeout error, it is helpful to know which remote GIC block caused the error. To determine its block ID, read the FMU_ERR<n>STATUS register.

Arm expects that background ping using FMU_PINGCTLR and directed ping using FMU_PINGNOW are used mutually exclusively. Therefore:

- When background pings are enabled, do not set FMU_PINGNOW.enable = 1. When initiating background ping, you must ensure that a previously initiated software ping (PINGNOW) is complete, by polling for FMU_PINGNOW.ping_ack_received==1.
- Before generating directed pings using the FMU_PINGNOW register, turn off background ping by setting FMU_PINGCTLR.enable = 0 and wait for the last PING_ACK to return.

When the FMU indicates a PING_TIMEOUT error, you can obtain the remote GIC block ID by the reading the FMU_ERR<n>STATUS register.

To conserve operational power of the GICD, the GICD accepts the Q-Channel handshake to enter low powerdown state, if requested by the clock controller. When the GICD is in the low-power clock gated state, it does not send background ping messages to the remote GIC block and does not report PING_ACK violations. When the GICD exits the low-power clock gated state, the FMU resumes background pings.

Directed ping

The software can also send a directed ping message to a specific block using the FMU_PINGNOW register. Using this method can be helpful to debug PING_ACK violations that background pings cause.

The procedure to initiate a directed software ping is as follows:

1. If background ping is enabled, disable background pings by writing FMU_PINGCTLR.enable=0.
2. Clear all flags by writing all zeros to FMU_PINGNOW.
3. Initiate a directed ping by writing:
   a. FMU_PINGNOW.enable=1
   b. FMU_PINGNOW.ping_ack_received=0
   c. The appropriate block ID to FMU_PINGNOW.block_id
4. Poll FMU_PINGNOW.ping_ack_received==1
5. Optionally, set the *_inject_error bits to test remote GIC block or GICD integration, software, or both.

The PINGNOW feature can be used to send an erroneous packet from the GICD to a targeted remote GIC block or from a targeted remote GIC block to the GICD. Using this feature enables the integrator to verify the AXI4-Stream connections between the remote GIC block and the GICD.

Injecting an error on a GICD ping message and on the subsequent remote GIC block PING_ACK message causes mismatches along the PING/PING_ACK route through the interconnect.

After injecting a PINGNOW error, you can read the GICD error records and verify that the expected SM errors are reported along the PING or PING_ACK route, for example by the receiving block and by any ADB components along the path.

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

- A single ping is sent for each write to a present block.
- If another ping is sent before a previous PING_ACK has been received, then:
  - If sent to the same destination, then the first ping back sets FMU_PINGNOW.ping_ack_received.
  - If sent to a different destination, then the first PING_ACK is silently discarded if or when received because it does not match the programmed FMU_PINGNOW.block_id.
- An attempt to send a ping to a non-present block does not launch a ping and FMU_PINGNOW.ping_ack_received is not set to 1.

If FMU_PINGNOW.gicd_inject_error == 1, an error is injected on the outgoing PING packet on the GICD to the remote GIC block interface. The receiving remote GIC block and the ADB, if present, detect the erroneous payload and report it as a fault.

If FMU_PINGNOW.remote_block_inject_error == 1, an error is injected on the outgoing PING_ACK packet by the remote GIC block on the remote GIC block to the GICD interface. The receiving GICD block and the ADB, if present, detect the erroneous payload and report it as a fault.

6.2.7 Lock and key mechanism

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

The FMU registers are in a locked state after reset. If the register file is locked, then any write access to any register other than the FMU_KEY register is ignored.

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

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

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

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

A write to FMU_KEY, when unlocked, leaves the register file unlocked only if the write satisfies the criteria for unlocking the register file. Otherwise, it locks the register file.

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

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

Accessing 64-bit FMU registers

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

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

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

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

6.2.8 Correctable Error enable

By default, the FMU considers all errors to be *Uncorrectable Errors* (UEs). To allow the FMU to treat RAM *Single Error Correct* (SEC) error indications as *Correctable Errors* (CEs), set the FMU_ERR<n>CTLR.CE_EN bit.

When FMU_ERR<n>CTLR.CE_EN is set to 1, the RAM SEC errors set the FMU_ERR<n>STATUS.CE bit.

When a CE is followed by an UE, FMU_ERR<n>STATUS.IERR is updated to reflect the UE Safety Mechanism ID. See Prioritized FMU_ERR<n>STATUS registers on page 208 for more information.

6.2.9 Software interaction

This section describes how software interacts with the FMU.

Initialization

The initialization routine can determine that 44 implemented error records exist, by reading the FMU_ERRIDR register. It can iterate over the FMU_ERR<n>FR registers to understand the capabilities of each error record.


Note

All Safety Mechanisms are enabled on reset, which might lead to errors being logged in the error records. If the system does not support or want to check a particular safety feature, then the software can disable that Safety Mechanism.

To disable a Safety Mechanism, write the corresponding block ID and Safety Mechanism ID to the FMU_SMEN register.

To analyze the logged errors, read the FMU_ERR<n>STATUS register.

To clear all logged errors, write all ones to the FMU_ERR<n>STATUS registers.

To enable error reporting through either the ERI or FHI, write to FMU_ERR<n>CTLR.UI or FMU_ERR<n>CTLR.FI, respectively.

Interrupt handler

When an interrupt is received, the interrupt handling software identifies the error record ID by reading the FMU_ERRGSR register. The asserted bit[M] indicates that error record M is in error. For additional information about the error, read the FMU_ERR<M>STATUS register.

FMU_ERR<M>STATUS.IERR indicates which Safety Mechanism reported the error.

If more than one error has been reported by this block to this error record, FMU_ERR<M>STATUS.OF is asserted. In case of overflow, the error record retains the Safety Mechanism ID of the first error.

When the recovery procedure is complete, the error from this error record can be cleared by writing all ones to this register. Then the software should poll for FMU_STATUS.idle==1.

Prioritized FMU_ERR<n>STATUS registers

If a CE is followed by a UE before software has responded to the initial CE, the following sequence occurs:

1. The FMU_ERR<n>STATUS registers are updated to reflect the SM ID of the UE.
2. The UE bit is set along with the CE bit.
3. The OF bit is not set in this case. Overflow is only set when one of the following cases occurs:
    a. Two UEs are received before software has responded, regardless of whether CEs were received.
    b. Two CEs are received back-to-back before software has responded.

To avoid a CE blocking a head-of-line UE, the GIC-600AE has separate UE and CE pipelines.

FMU idle

The APB port to the FMU is designed not to introduce backpressure by deasserting the pready signal. This design feature prevents software lockup and always keeps the error records accessible.

There are several operations which take multiple clock cycles to complete within the FMU. The FMU frees up the APB bus by asserting the pready signal to complete the APB transaction. However, it might still be processing the previous request.

When software writes to one of the following FMU registers, it must poll for FMU_STATUS.idle==1 before it issues another write to these registers:

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

Power management

The software can power down the Redistributor (PPI block) using the procedure that 4.6.1 Redistributor power management on page 56 describes, or the ITS could be powered down by using the GITS_CTLR register. However, the powerdown state of the PPI block and the ITS block affects certain functions of the FMU.

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

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

The software must be aware of the power state of the remote blocks and not initiate writes to these registers that target a powered-down remote GIC block. If software initiates a write to the following registers that target a powered-off remote GIC block, then:

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

6.3 FuSa programmer's view

The FMU contains the functional safety registers.

The GIC-600 memory map that is used to address the legacy GIC functional logic is unchanged on GIC-600AE. See 5. Programmers model on page 95 for the functional GIC-600 memory map.

GIC-600AE uses a separate and independent memory map for the *Fault Detection and Control* (FDC) programmer's view. For a description of the registers that are specific to GIC-600AE, see 5.10 FMU register summary on page 179.

6.4 FuSa I/O

Ports have been added for FuSa fault detection and control.

See 6.8 External interface protection on page 221 for more information about the new interfaces.

6.4.1 Non-architected FuSa ports

The following ports have been added for fault detection and control.

In the following table, *Granularity* refers to the hierarchy or the block in which the ports are relevant.

Table 6-3: Non-architected FuSa ports

Port	Direction	Granularity	Description
clk_fdc	Input	Per domain	Clock for redundant logic and SMs
reset_n_fdc	Input	Per domain	Reset for redundant logic and SMs
nmbistreset_fdc	Input	Per domain	Redundant nmbistreset signal. Both resets must assert together.
dbg_reset_n_fdc	Input	GICD domain	Redundant dgb_reset_n reset signal for PMU and FMU. Both resets must assert together.
dftrstdisable_fdc	Input	All domains	Prevents reset from asserting when reset generation FDC flops are scanned

Port	Direction	Granularity	Description
dftcgen_fdc	Input	All domains	Forces FDC clock gate enable, to ensure scanned flops get a clock
dftramhold_fdc	Input	All domains	Redundant dftramhold signal
dftse_fdc	Input	All domains	Scan enable for FDC clock flops
usr0_err	Input	Per block	External IP user fault input 0
usr1_err	Input	Per block	External IP user fault input 1
fmu_err_in[43:0]	Input	GICD block	Redundant fault indicator inputs connecting outer GIC blocks to Distributor
fmu_err_out	Output	Outer blocks	Redundant fault indicator outputs connecting outer GIC blocks to Distributor
fmu_fault_int	Output	GICD block	*Fault Handling Interrupt* (FHI) from GICD FMU to Safety Island
fmu_fault_int_chk	Output	GICD block	Redundant fmu_fault_int signal
fmu_err_int	Output	GICD block	*Error Recovery Interrupt* (ERI) from GICD FMU to Safety Island
fmu_err_int_chk	Output	GICD block	Redundant fmu_err_int signal
gicd_page_offset_chk	Input	GICD block	Redundant gicd_page_offset tie-off signal. Only present in monolithic configurations. This port is the inverted duplication of the gicd_page_offset signal.
its_transr_page_offset_chk	Input	GICD block	Redundant its_transr_page_offset tie-off signal. Only present in monolithic configurations. This port is the inverted duplication of the its_transr_page_offset signal.
ppi_id_chk	Input	PPI block	Redundant ppi_id tie-off signal
wake_request_chk[cpus −1:0]	Output	Wake Request block	Parity CHK for the wake_request signals. Odd parity.
fault_*	Input/ output	Outer blocks	See Table 6-12: fault_* tie-off signals on page 232 for more information

6.4.2 P-Channel and Q-Channel FuSa ports

The following interfaces add the parity extended chk signal bits.

In the following table, *Granularity* refers to the hierarchy or the block in which the ports are relevant. See the AMBA® Low Power Interface Specification for more information about the parity extended chk signal bits.

Table 6-4: P-Channel and Q-Channel FuSa ports

Port	Direction	Granularity	Description
pwrqreqn_chk	Input	Q-Channel device interface for power control in each domain	Redundant pwrqreqn port
pwrqactive_chk	Output		Redundant pwrqactive port
pwrqacceptn_chk	Output		Redundant pwrqacceptn port
pwrqdeny_chk	Output		Redundant pwrqdeny port
clkqreqn_chk	Input	Q-Channel device interface for clock control in each domain	Redundant clkqreqn port
clkqactive_chk	Output		Redundant clkqactive port
clkqacceptn_chk	Output		Redundant clkqacceptn port
clkqdeny_chk	Output		Redundant clkqdeny port
preq_chk	Input	P-Channel device interface for the GICD block	Redundant preq port

Port	Direction	Granularity	Description
paccept_chk	Output		Redundant paccept port
pdeny_chk	Output		Redundant pdeny port
pstate_chk	Input		Redundant pstate port
pactive_chk	Output		Redundant pactive port

See 6.10 P-Channel and Q-Channel protection on page 232 for more information.

6.4.3 AMBA interface FuSa ports

The following interfaces add the parity extended chk signal bits, as specified in the Arm AMBA® Parity Extensions.

In the following table, *Granularity* refers to the hierarchy or the block in which the ports are relevant.

Table 6-5: AMBA interface FuSa ports

Port	Granularity	Description
APB4 interface	GICD block	APB4 interface added for FMU
AXI4-Stream AMBA® parity	All blocks	AMBA® parity added to all external AXI4-Stream interfaces
ACE-Lite AMBA® parity	GICD/ITS blocks	AMBA® parity added to all external ACE-Lite interfaces
CPU interface (AXI4-Stream)	Per PPI block	irit signal bus (PPI to core) and icct signal bus (core to PPI) parity extensions for AXI4-Stream
Cross-chip (AXI4-Stream)	Per CC chip	Parity extensions for AXI4-Stream

The APB port has been added for fault detection and control between the FMU block and the Safety Island in the SoC.

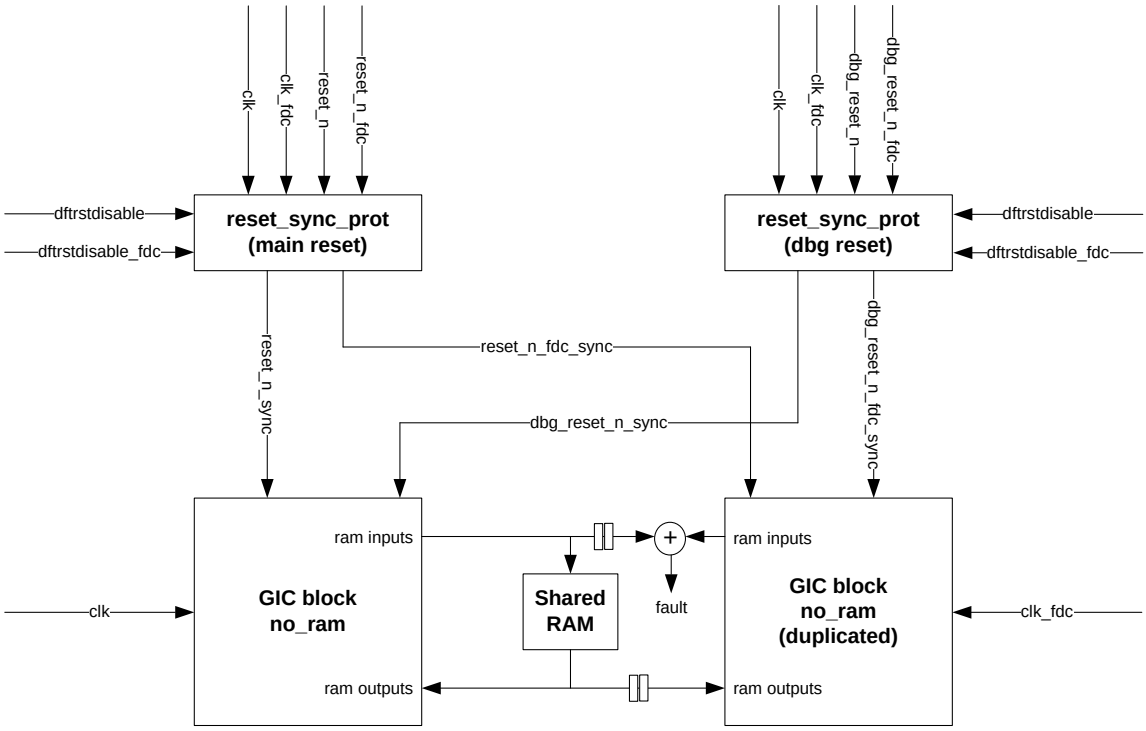
See 6.8 External interface protection on page 221 for more information.

6.5 Clocks and resets

The GIC-600AE clocks and resets are identical to those signals of the GIC-600, except for the added redundant clock and reset signals.

The following figure shows how the redundant clock and reset signals are used by the FDC logic.

Figure 6-4: GIC clocks and resets



Note

- Internal _sync reset signals are asynchronous-assert and synchronous-deassert.
- The reset_n_fdc_sync and dbg_reset_n_fdc_sync signals are deasserted two cycles after the non-FDC signals.
- The reset qualification with the nmbistreset signal is not shown before the `reset_sync_prot` block.

The extra reset_n_fdc and clk_fdc signals provide redundancy in the clock and reset trees to guard against faults on the tree branches. If a fault occurs on a branch in the primary or FDC clock trees, the *Dual Lock-Step* (DLS) comparators detect it.

6.5.1 Clocks

The GIC-600AE has two global clocks for each stitched level.

The clocks that are used for wrap components are:

clk This signal clocks the primary mission-critical logic
clk_fdc This signal clocks the *Fault Detection and Control* (FDC) redundant logic

The clocks that are used for stitched domain modules, and the top level, are:

<domain>clk This signal clocks the primary mission-critical logic
<domain>clk_fdc This signal clocks the redundant logic

The functional requirements for the clk and clk_fdc signals are:

- clk and clk_fdc must be edge-synchronous and run at the same frequency
- clk and clk_fdc must start and stop at the same time

Asynchronous inputs to clk and clk_fdc

Some signals such as qreqn[_*] and interrupt wires have built-in or optional inverters and synchronizers. These inverters and synchronizers are set by the `*_INV` and `*_SYNC` parameters, respectively. All other signals belonging to the same module must be synchronous to the clock.

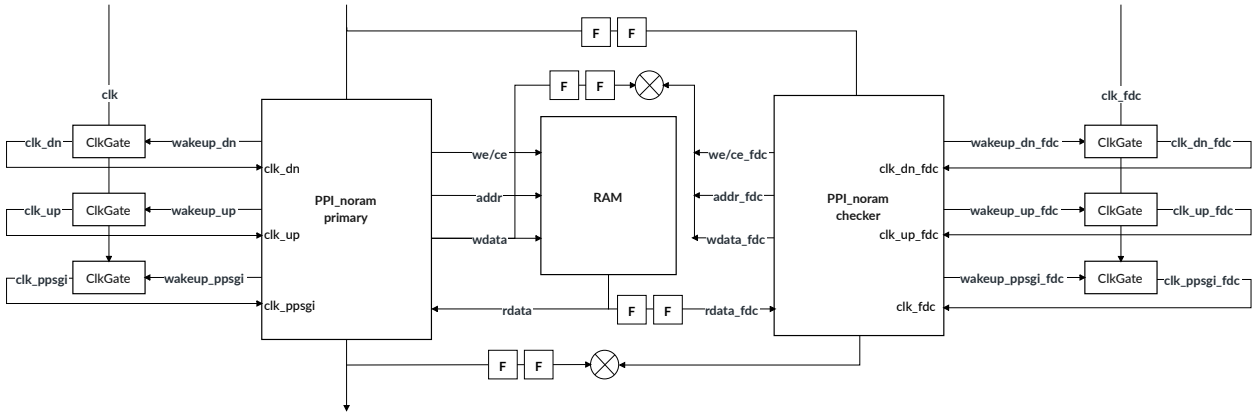
For more information, see 3.2.4 Redistributor PPI signals on page 33 and 3.5.2 SPI Collator wires on page 42.

6.5.1.1 Block-level clocking

The GICD, GICR, and ITS blocks all have a similar clocking structure.

The following figure shows an example clocking structure for the GICR.

Figure 6-5: GICR block-level clocking example



On the primary side, the clk signal is the always-on clock. It generates architecturally clock gated versions of the clocks through the `ClkGate` cells. In Figure 6-5: GICR block-level clocking example on page 213, the architecturally gated clocks are the clk_dn and clk_ppsgi signals.

On the redundant side, the clk_fdc signal works similarly but uses its own redundant `ClkGate` cells.

6.5.2 Resets

Each stitched level has two resets, which are active-LOW.

The resets that are used for wrap components are:

reset_n This signal is the reset for primary mission-critical logic
reset_n_fdc This signal is the reset for redundant logic

The resets that are used for stitched domain modules, or the top level, are:

<domain>reset_n
This signal is the reset for primary mission-critical logic

<domain>reset_n_fdc
This signal is the reset for redundant logic

The GIC-600AE has an internal reset synchronizer, so that on reset the internal reset signal asserts asynchronously and deasserts synchronously.

The functional requirements for the reset_n and reset_n_fdc signals are:

- reset_n and reset_n_fdc must both assert before the reset can propagate to downstream logic. If only one reset asserts, then GIC-600AE does not reset.
- To ensure that reset can properly propagate through the primary and redundant logic pipelines, the reset_n and reset_n_fdc signals must assert simultaneously for at least 16 clock cycles. Otherwise, false fault assertions might occur.

The domain that contains the Distributor has separate dbg_[<domain>]reset_n and dbg_[<domain>]reset_n_fdc signals. The dbg_* reset signals are used to reset debug, trace, and the FMU error records containing fault status information. This functionality allows the GIC to be reset using the reset_n and reset_n_fdc signals, while leaving any trace, debug, and fault error record information available for later interrogation. It must be reset only when the [<domain>]reset_n and [<domain>]reset_n_fdc signals are asserted.

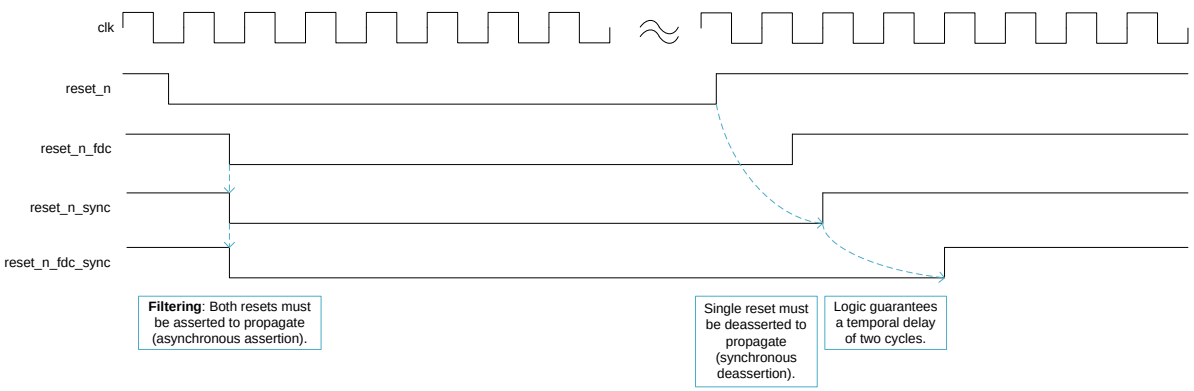
6.5.2.1 DLS resetting

For blocks with *Dual LockStep* (DLS) logic, the redundant block must exit reset two cycles after the primary block, or else false fault assertions occur.

The `reset_sync_prot` block guarantees this behavior for the main resets and the dbg_* reset signals. It also filters out transient reset assertion by preventing reset from propagating unless the reset_n and reset_n_fdc signals are both asserted.

The following figure shows this behavior in a timing diagram.

Figure 6-6: FuSa reset timing diagram



FuSa reset port fault protection

Transient reset port protection

The GIC protects the reset_n_sync, reset_n_fdc_sync, dbg_reset_n_sync, and dbg_reset_n_fdc_sync signals from spurious transient faults. It does this in the `reset_sync_prot` block by requiring both the primary and FDC resets to be asserted before it asserts the synchronized reset to the downstream logic. For example, the reset_n_sync and reset_n_fdc_sync signals are not asserted unless the reset_n and reset_n_fdc signals are asserted at the same time.

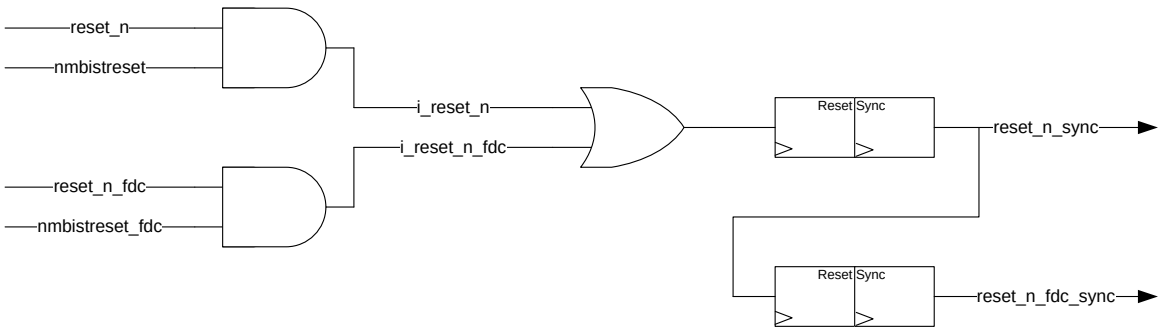
Stuck-at-reset port protection/detection

The GIC protects itself from *stuck-at-zero* (STA0) faults on the reset pin inputs. *Stuck-at-one* (STA1) faults are not detected or reported, as they prevent the GIC from resetting correctly. If an implementation must detect STA1 faults, the SoC integrator can do so through external hardware or self-test means. See Table 6-6: GIC reset failure modes on page 216 for more information.

Internal reset fault protection/detection

The reset trees are duplicated, so faults on reset trees are detected through lockstep protection mechanisms for the affected blocks.

Figure 6-7: GIC reset protection



The following table describes the GIC reset failure modes.

Table 6-6: GIC reset failure modes

Signal	Fault	Detected by GIC?	Failure mode
reset_n	STA0 (asserted)	No	GIC logic behaves normally. GIC prevents the reset_n signal from resetting GIC until the reset_n_fdc signal is asserted.
	STA1	No	GIC logic behaves normally. GIC cannot be reset.
reset_n_fdc	STA0	No	GIC logic behaves normally. GIC prevents the reset_n_fdc signal from resetting GIC until the reset_n signal is asserted.
	STA1	No	GIC logic behaves normally. GIC cannot be reset.
dbg_reset_n	STA0	No	PMU/FMU logic behaves normally. GIC prevents the dbg_reset_n signal from resetting DBG/FMU until the dbg_reset_n_fdc signal is asserted.
	STA1	No	PMU/FMU logic behaves normally. PMU/FMU cannot be reset.
dbg_reset_n_fdc	STA0	No	PMU/FMU logic behaves normally. GIC prevents the dbg_reset_n_fdc signal from resetting DBG/FMU until the dbg_reset_n signal is asserted.
	STA1	No	PMU/FMU logic behaves normally. PMU/FMU cannot be reset.

6.5.2.2 Cold reset sequence

Follow these steps to initiate a Cold reset of the GIC.

Procedure

1. Assert the reset_n_sync, reset_n_fdc_sync, dbg_reset_n_sync, and dbg_reset_n_fdc_sync signals simultaneously.
   - reset_n_sync and reset_n_fdc_sync signals assert asynchronously at the same time. To assert the reset_n_sync and reset_n_fdc_sync signals, both external ports must be asserted.
   - dbg_reset_n_sync and dbg_reset_n_fdc_sync signals assert asynchronously at the same time. To assert the dbg_reset_n_sync and dbg_reset_n_fdc_sync signals, both external ports must be asserted.
2. Keep the resets asserted for 16 clk cycles.

This guarantees a reset flush through the non-resettable flops.

3. Release the resets.
   When either:
   - The reset_n or reset_n_fdc signal deasserts, the reset_n_sync signal deasserts synchronously, followed by the reset_n_fdc_sync signal two clk cycles later.
   - The dbg_reset_n or dbg_reset_n_fdc signal deasserts, the dbg_reset_n_sync signal deasserts synchronously, followed by the dbg_reset_n_fdc_sync signal two clk cycles later.

6.5.2.3 Warm reset sequence

Follow these steps to initiate a Warm reset of the GIC.

About this task

A Warm reset is a reset that occurs after the component has already been operating for some time. The reset preserves the state of the PMU and the error records, in both the functional and FuSa GIC address maps. This state preservation is accomplished by not toggling the dbg_reset_n signals.

Procedure

1. Assert the reset_n_sync and reset_n_fdc_sync signals simultaneously.
   reset_n_sync and reset_n_fdc_sync signals assert asynchronously at the same time. To assert the reset_n_sync and reset_n_fdc_sync signals, both external ports must be asserted.
2. Keep the resets asserted for 16 clk cycles.
   This duration guarantees a reset flush through the non-resettable flops.
3. Release the resets.
   When either the reset_n or reset_n_fdc signal deasserts, the reset_n_sync signal deasserts synchronously, followed by the reset_n_fdc_sync signal two clk cycles later.

6.6 Lock-step protection

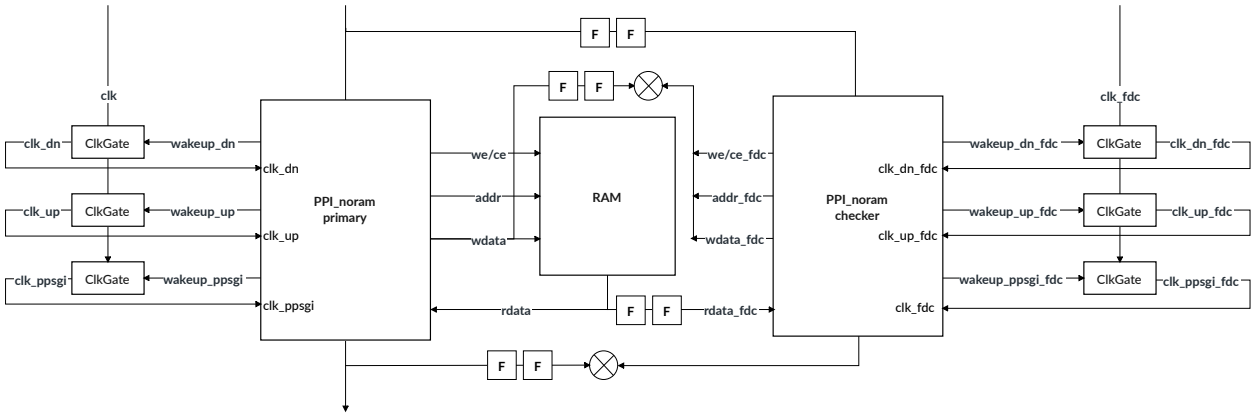
The GIC-600AE logic is protected by redundant lock-step checking.

The exceptions to this are:

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

The following figure shows the lock-step for the PPI (Redistributor).

Figure 6-8: PPI lock-step



The lock-step has a standard temporal delay of two cycles, with RAM sharing and comparators. A circle with an X in the middle indicates a comparator. To save power, CRC is used to compress the outputs.

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

The clocking is also duplicated. To provide redundancy in the reset and clock trees, the primary (main) and checker (shadow) logic are clocked by a separate clock and separate reset. In the clock tree, if a branch of the reset fails in the primary domain, then the checker domain detects the failure. Similarly, if a branch of the reset fails in the checker domain, then the primary domain detects the failure.

6.6.1 Comparators

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

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

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

6.6.1.1 Comparator duplication option

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

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

To duplicate the comparators, set `FUSA_COMP_DUP`=1 before rendering the GIC-600AE.

All comparators in the GIC-600AE can be duplicated, including lock-step and CRC comparators.

6.6.2 Non-resettable flops

All non-resettable flops that could not be proven benign have been changed to resettable versions.

6.6.3 Reset

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

See the following sections for more information on reset assumptions and requirements related to lockstep logic and FuSa.

Related information

Clocks and resets on page 211

6.6.4 Error injection

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

The main purpose is to test connectivity and software. It is not meant to be an exhaustive test of the comparator XOR tree. For this purpose, the comparators can be duplicated as described in 6.6.1.1 Comparator duplication option on page 218.

6.7 RAM protection

The GIC-600AE inherits SECDED ECC protection and patrol scrubbing from GIC-600. The address is not protected on GIC-600, so this protection is added on GIC-600AE.

6.7.1 SECDED ECC data protection

SECDED ECC is a legacy GIC-600 feature.

For information about how to use this feature, see the GIC-600 sections of this document.

6.7.1.1 SECDED ECC fault reporting

SECDED ECC faults are reported by separate registers in both the legacy GIC-600 and the GIC-600AE FuSa programmer's view.

The GIC-600 programmer's view reports all information about the RAM fault, including the fault address. The GIC-600AE programmer's view is limited to reporting whether a *Single Error Corrected* (SEC) or *Double Error Detected* (DED) fault occurred. You cannot retrieve the affected address from the GIC-600AE programmer's view.

6.7.1.2 SBEs treated as fatal errors or corrected errors

FMEDA analysis might show that it is necessary to treat *Single-Bit Errors* (SBEs) as fatal errors.

This might be necessary if *Multiple-Bit Errors* (MBEs) are common, meaning there are more than two errors in read data. If an MBE is encountered, conservative SECDED math tells us there is approximately a 33 percent chance that the SECDED algorithm mistakes an MBE for an SBE and corrects it erroneously.

The RAM scrubbers can be used to mitigate the chance of an MBE error by finding and correcting SBEs before they have the chance to become *Double-Bit Errors* (DBEs) or MBEs. However, if all SBEs are treated as fatal data, the achieved coverage is approximately 99.4 percent (measured). SECs can be flagged as fatal by programming the GIC FMU to output an ERI interrupt instead of an FHI interrupt. In this case, the correction is ignored, and all SEC faults are treated as fatal. Correction cannot be disabled.

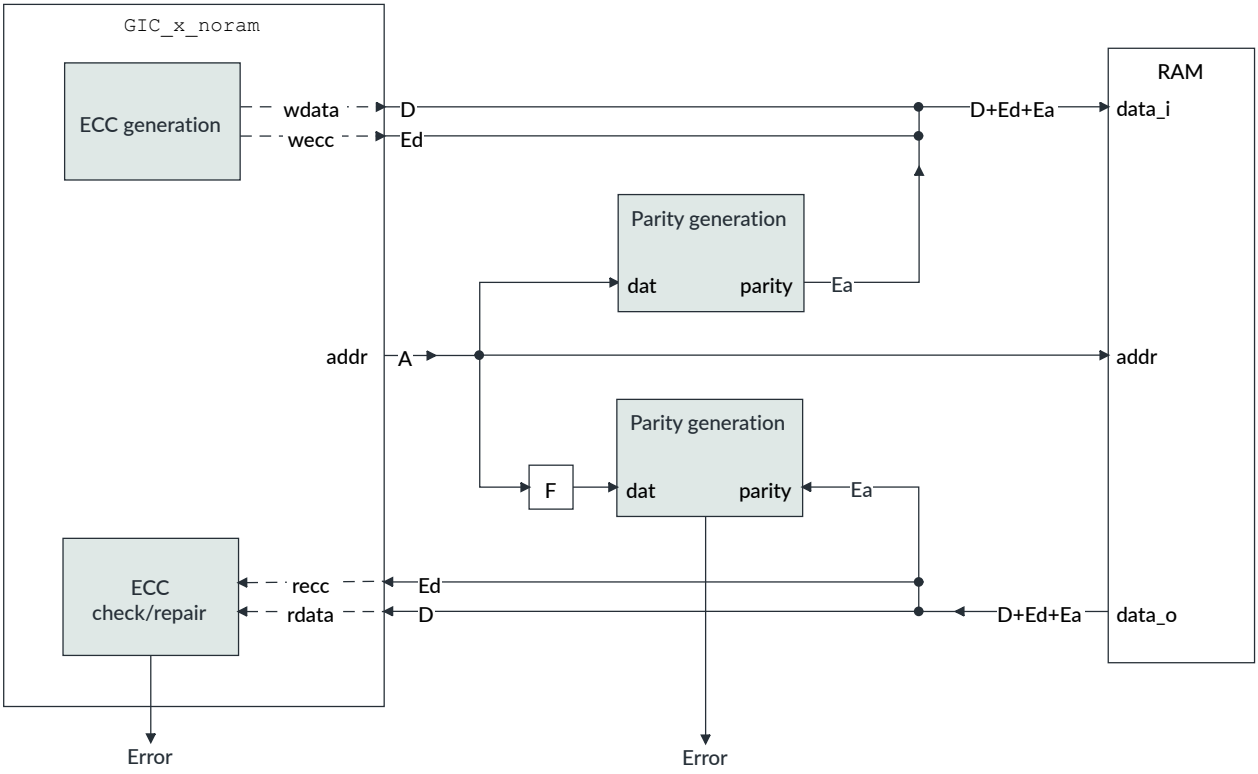
6.7.2 Address protection

Address protection must consider the protection of address decoders within the RAM decoder macro themselves.

This is because the RAM is shared, and otherwise faults within the RAM macro address decoder cause *Common Mode Failure* (CMF). This protection is achieved by calculating parity for the address bits and writing the parity into the RAM along with the data.

The following figure shows how the address protection works.

Figure 6-9: Address protection structure



6.7.3 RAM scrubbing

The GIC-600AE supports software-initiated RAM data scrubbing.

This feature reads an incremental address location and checks it for SBEs using the SECDED ECC algorithm. If an SBE is found, the error is corrected and written back to memory.

Data scrubbing is a legacy GIC-600 feature. For more information on how to use data scrubbing, see the GIC-600 sections of this document.

6.8 External interface protection

All external bus interfaces are protected as defined by the AMBA® parity extensions.

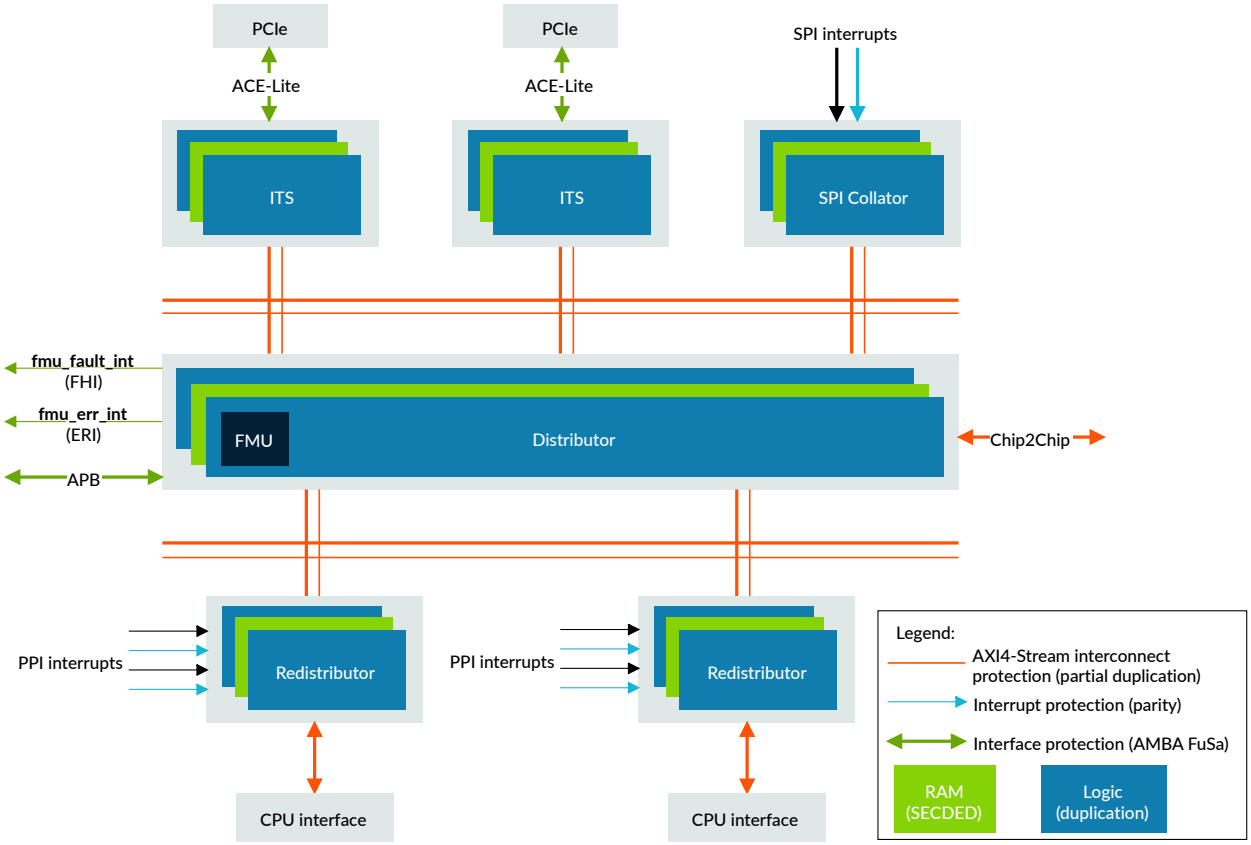
These external interfaces include:

- ACE-Lite
- APB
- AXI4-Stream, and the following interfaces, which use AXI4-Stream as their transport:
  - GIC Stream

- Chip2Chip

The following figure shows the distribution of interface protection within the GIC-600AE.

Figure 6-10: Interface protection distribution



Point-to-point protection

Point-to-point protection is sufficient for wires and buffers that cannot cause multiple-bit faults. An example of an interconnect component that might cause multiple-bit faults is a switch. A single fault on a switch mux input can switch the wrong data, causing multiple bits to fail.

6.8.1 ACE-Lite interface parity protection

The GIC-600AE supports ACE-Lite interface parity protection for point-to-point connections from the GIC-600AE to another functionally safe IP or FuSa interconnect. If a parity fault is detected, the GIC-600AE flags a fault.

If this protection is not needed, software can disable the appropriate ACE-Lite Safety Mechanisms by programming the FMU_SMEN register. Disable this protection when using an interconnect that does not generate AMBA® parity.

Assumptions of Use for FuSa purposes

We expect that:

- The GIC-600AE connects directly to the far-end IP with only wires and repeater buffers
- No complex logic gates such as ADBs or crossbar switches exist in the path, because they could be a source of *Multiple-Bit Errors* (MBEs)
- The far-end IP checks the parity bits that the GIC-600AE generates
- The far-end IP generates the incoming parity bits, as the following table describes

Table 6-7: ACE-Lite interface parity

Check signal	Signals covered	Width	Granularity	Check enable
awvalidchk	awvalid	1	1	-
awreadychk	awready	1	1	-
awidchk	awid	ceil(`IdWidthW`/8)	`IdWidthW`	awvalid==1
awaddrchk	awaddr	ceil(`AddrWidth`/8)	1-8	awvalid==1
awlenchk	awlen	1	8	awvalid==1
awctlchk0	awsize, awburst, awlock, awprot	1	1-9	awvalid==1
awctlchk1	awregion, awcache, awqos	1	4-12	awvalid==1
awctlchk2	awdomain, awsnoop, awunique, awbar	1	4-9	awvalid==1
awuserchk	awuser	ceil(`AWUserWidth`/8)	1-8	awvalid==1
awatopchk[12]	awatop	1	6	awvalid==1
wvalidchk	wvalid	1	1	aresetn==1
wreadychk	wready	1	1	aresetn==1
wdatachk	wdata	`DataWidthW`/8	8	wvalid==1
wstrbchk	wstrb	ceil(`DataWidthW`/64)	1-8	wvalid==1
wlastchk	wlast	1	1	wvalid==1
wuserchk	wuser	ceil(`WUserWidth`/8)	1-8	wvalid==1
bvalidchk	bvalid	1	1	aresetn==1
breadychk	bready	1	1	aresetn==1
bidchk	bid	ceil(`IdWidthW`/8)	`IdWidthW`	bvalid==1
brespchk	bresp	1	2	bvalid==1
buserchk	buser	ceil(`BUserWidth`/8)	1-8	bvalid==1
arvalidchk	arvalid	1	1	aresetn==1
arreadychk	arready	1	1	aresetn==1
aridchk	arid	ceil(`IdWidthR`/8)	`IdWidthR`	arvalid==1
araddrchk	araddr	ceil(`AddrWidth`/8)	8	arvalid==1
arlenchk	arlen	1	8	arvalid==1
arctlchk0	arsize, arburst, arlock, arprot	1	1-9	arvalid==1
arctlchk1	arregion, arcache, arqos	1	4-12	arvalid==1
arctlchk2	ardomain, arsnoop, arbar	1	4-8	arvalid==1

[12] awatop is used for atomics, and is only visible when `atomic_support==1`

Check signal	Signals covered	Width	Granularity	Check enable
aruserchk	aruser	ceil(AWUserWidth/8)	1-8	arvalid==1
rvalidchk	rvalid	1	1	aresetn==1
rreadychk	rready	1	1	aresetn==1
ridchk	rid	ceil(IdWidthR/8)	IdWidthR	rvalid==1
rdatachk	rdata	DataWidthR/8	8	rvalid==1
rrespchk	rresp	1	2-4	rvalid==1
rlastchk	rlast	1	1	rvalid==1
ruserchk	ruser	ceil(RUserWidth/8)	1-8	rvalid==1

6.8.2 AXI4-Stream interface parity protection

The GIC-600AE supports AXI4-Stream interface parity protection on point-to-point connections from the GIC-600AE to another FuSa IP or FuSa interconnect. If a parity fault is detected, the GIC-600AE flags a fault.

If this protection is not needed, software can disable the appropriate AXI4-Stream Safety Mechanisms by programming the FMU_SMEN register. When using an interconnect that does not generate AMBA® parity, set FUSA_AXIS_INT_BUSPROT_TYPE=0 to indicate parity, tie off all parity bits to 1, and disable the AXI4-Stream protection for all blocks in the programmers view.

Assumptions of Use for FuSa purposes

We expect that:

- The GIC-600AE connects directly to the far-end IP with only wires and repeater buffers
- No complex logic gates such as ADBs or crossbar switches exist in the path, because they could be a source of MBEs
- The ADB FuSa parameters FW_CHK_FIFO_DEPTH and RV_CHK_FIFO_DEPTH are set as the *Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Configuration and Integration Manual* describes
- The far-end IP checks the parity bits that the GIC-600AE generates
- The far-end IP generates the incoming parity bits as the following table describes

Table 6-8: AXI4-Stream interface parity

Check signal	Signals covered	Width	Granularity	Check enable
tclkchk	tclk	1	1	-
tresetchk	tresetn	1	1	-
tvalidchk	tvalid	1	1	-
treadychk	tready	1	1	-
tdatachk	tdata	n	8	tvalid==1
tstrbchk	tstrb	ceil(n/8)	1-8	tvalid==1
tkeepchk	tkeep	ceil(n/8)	1-8	tvalid==1
tlastchk	tlast	1	1	tvalid==1

Check signal	Signals covered	Width	Granularity	Check enable
tidchk	tid	1 The recommended maximum width for the tid signal is 8 bits. If the tid signal is wider than 8 bits, then the tidchk signal is wider than 1 bit.	1-8	tvalid==1
tdestchk	tdest	1 The recommended maximum width of the tdest signal is 4 bits. If the tdest signal is wider than 4 bits, then the tdestchk signal is wider than 1 bit.	1-4	tvalid==1

6.8.3 APB interface parity protection

The GIC-600AE supports APB interface parity protection on point-to-point connections from the GIC-600AE to another FuSa IP or FuSa interconnect. If a parity fault is detected, the GIC-600AE flags a fault.

If this protection is not needed, software can disable the FMU APB Safety Mechanism by programming the FMU_SMEN register. Disable this protection when using an interconnect that does not generate AMBA® parity.

Assumptions of Use for FuSa purposes

We expect that:

- The GIC-600AE connects directly to the far-end IP with only wires and repeater buffers.
- No complex logic gates such as ADBs or crossbar switches exist in the path, because they could be a source of MBEs.
- The far-end IP checks the parity bits that the GIC-600AE generates.
- The far-end IP generates the incoming parity bits as the following table describes.

Table 6-9: APB interface parity

Check signal	Signals covered	Width	Granularity	Check enable
paddrchk	paddr	ceil(`AddrWidth`/8)	1-8	psel==1
pctrlchk	pprot, pwrite	1	4	psel==1
pselchk	psel	1	1	-
penablechk	penable	1	1	psel==1
pwdatachk	pwdata	ceil(`DataWidth`/8)	8	psel&&pwrite
preadychk	pready	1	1	penable==1
prdatachk	prdata	ceil(`DataWidth`/8)	8	psel&&pready&&!pwrite
pslverrchk	pslverr	1	1	pready

6.9 AXI4-Stream internal interconnect protection

The GIC-600AE renders a protected AXI4-Stream interconnect for connecting the various GIC blocks. Alternatively, the system integrator can use a non-GIC-600AE AXI4-Stream-interface-compliant IP to connect the GIC blocks.

The GIC-600AE supports the following options for protecting the AXI4-Stream interfaces:

Duplicated AXI4-Stream interfaces

Use the GIC-600AE protected AXI4-Stream interconnect. The GIC-600AE generates an interconnect, in which the interconnect components are partially duplicated. That is, the redundant interconnect payload is represented by an 8-bit CRC code. See 6.9.1 GIC-rendered partially duplicated interconnect on page 226 for more information.

Single AXI4-Stream interface with AMBA® protection

Use non-GIC-600AE interconnect IP with interfaces between GIC blocks and the interconnect. The interface between the GIC blocks and interconnect IP is protected with AMBA® Parity Extensions. In this mode, the GIC-600AE generates the parity for the interface outputs, checks the parity for interface inputs, and flags a fault if there is a mismatch. See 6.8.2 AXI4-Stream interface parity protection on page 224 for more information.

Single AXI4-Stream interface with no protection

Use non-GIC-600AE standard AXI4-Stream interconnect IP without AMBA® Parity Extensions to connect GIC blocks. The GIC-600AE must ignore the input parity signals.

6.9.1 GIC-rendered partially duplicated interconnect

The AXI4-Stream internal interconnect is protected with partial duplication.

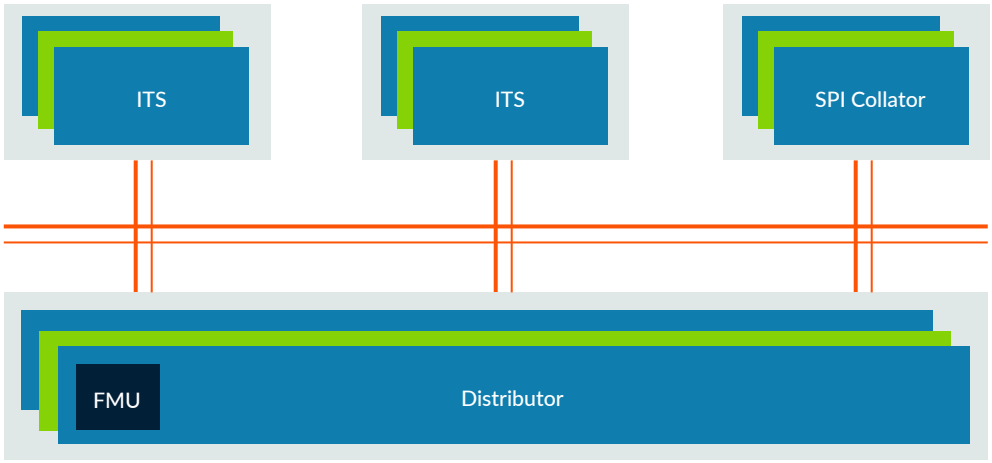
Partial duplication is the same as full duplication, except a CRC code is sent on the redundant leg instead of the fully duplicated payload. The CRC code on the shadow leg is then compared with the data from the primary leg at the destination.

Compared to full duplication, all known random faults are covered at a lower cost. This includes faults appearing on single-shot packets that do not have an associated response packet.

The following figure shows the GIC-600AE partial duplication microarchitecture, where:

- The wide orange line represents the primary interconnect and payload
- The narrow orange line represents the redundant interconnect compressed and represented by CRC

Figure 6-11: Partial duplication microarchitecture



Interfaces that use partial duplication do not invert their *_chk signals.

If a GIC-600AE configuration needs no interconnect components between two GIC blocks, the GIC interconnect render engine automatically uses *point-to-point* (P2P) protection.

Example

Consider an interconnect with the following conditions:

- One ITS block, with no ADB
- No register slices between the ITS and Distributor

The connections are point-to-point, and AMBA parity extensions are used instead of partial duplication with CRC. If any interconnect component lies between the ITS and Distributor, the render engine chooses partial duplication.

6.9.1.1 AMBA Domain Bridge

To maintain lockstep operation between the primary and redundant interconnects, the SoC integrator must use the GIC-600AE *AMBA Domain Bridge* (ADB).

To support partial duplication across asynchronous CDC, the ADB must also be partially duplicated.

The asynchronous nature of the CDC makes the arrival time at the subordinate indeterminate. Assuming a temporal delay of two cycles between the primary and shadow, the nondeterminism means that any of the following scenarios can occur. The primary can arrive:

- One cycle ahead of the shadow, which is a *fast shadow*
- Two cycles ahead of the shadow, which is the normal case
- Three cycles ahead of the shadow, which is a *slow shadow*

Any variation in arrival times between the primary and shadow at the subordinate or in the ready assertion at the manager causes the lockstep blocks to lose sync. As a result, a fault is flagged. To prevent this fault, the ADB must ensure the primary and shadow are always two cycles apart, or a temporal delay of two cycles, when they exit the ADB.

6.9.1.2 BAS switch

The BAS switch is partially duplicated.

There is no detection in the duplicated switch itself. Instead, fault detection occurs at the endpoint blocks or the ADB. Therefore, there is no fault wire exiting the switch.

6.9.1.3 Register slice

The register slice is partially duplicated.

Although the register slice has a Q-Channel interface, the interface must be synchronous to that clock domain by using an LPD-500. Therefore, it does not need to be protected by any special means of asynchronous protection. Faults that appear on the Q-Channel interface or logic are addressed in one of the following ways:

- If the faults are benign, the register slice absorbs them.
- If the faults are not benign, they are passed downstream for detection. The downstream block can be one of the main blocks that contains fault detection mechanisms. It can also be the LPD-500 itself, because it has fault detection mechanisms and a fault wire to report the faults.

6.9.2 Non-GIC interconnect IP

Any interconnect that supports AXI4-Stream-compliant interfaces can be used to connect the GIC blocks.

The GIC-600AE can be configured to have AMBA® parity protection on the AXI4-Stream interface. This interface is adequate for protecting wires and buffers that connect a GIC block to any non-GIC interconnect IP in a point-to-point configuration. This allows the GIC to connect with any unprotected legacy AXI4-Stream interconnect IP.

In this mode, the GIC-600AE generates the parity for the interface outputs, checks the parity for interface inputs, and flags a fault if there is a mismatch.

If the parity protection is not needed, tie off the extra parity inputs and program the GIC-600AE to disable this protection.

6.9.2.1 Configuring and integrating with a non-GIC interconnect

To use AXI4-Stream interconnect IP not rendered by the GIC, configure the GIC as follows:

Procedure

1. Render the configuration that you require, ensuring that it satisfies the following requirements:
   - There are no AXI4-Stream components between the rendered GIC blocks
   - The `fusa_axis_int_busprot_type` parameter is set to 0, which indicates the setting for AXI4-Stream-interface-compliant with AMBA parity protection.

   
   Note

   We expect that the rendered top-level files that contain the GIC-rendered interconnect are not used.
2. Discard the top level and connect the blocks to other AXI4-Stream IP.
3. Tie off the fault_* ports. See Table 6-12: fault_* tie-off signals on page 232.
   These ports are proprietary, and are only used by the GIC-rendered interconnect. We expect that one of the following is true:
   - The non-GIC interconnect has its own method of protecting interconnect components.
   - The non-GIC interconnect does not require this protection.
4. Connect the fmu_err_out ports in Table 6-11: Mandatory fmu_err_out signal connections for a non-GIC-rendered interconnect on page 230.
   These connections are from the outer GIC blocks to the GICD. They are a redundant way to communicate an outer GIC block fault to the FMU. The outer GIC blocks include:
   - PPI
   - ITS
   - SPI Collator
   - Wake Request

   If the system integrator does not make these connections, a deadlock or livelock condition can occur on the AXI4-Stream interconnect. This condition can block the fault from propagating to, and being flagged by, the FMU.

6.9.2.2 Operating an unprotected AXI4-Stream interface

Follow these steps to operate interconnect IP with a legacy AXI4-Stream interface that is not protected with AMBA® parity extensions.

Before you begin

Configure and render your non-GIC interconnect, as 6.9.2.1 Configuring and integrating with a non-GIC interconnect on page 228 describes.

Procedure

1. Tie off the unused parity chk input signal bits to any value, either HIGH or LOW.
2. Disable the following AXI4-Stream interface SMs, using the FMU_SMEN register.

Table 6-10: Safety Mechanisms to disable for unprotected interconnect

Block	Block ID	SM ID	SM description
GICD	0	3	GICD-PPI AXI4-Stream interface error
	0	4	GICD-ITS AXI4-Stream interface error
	0	5	GICD-SPI Collator AXI4-Stream interface error
	0	20	FMU APB parity error
	0	21	GICD-Wake Request AXI4-Stream interface error
SPI Collator	1	2	SPI Collator-GICD AXI4-Stream interface error
Wake Request	2	2	Wake Request-GICD AXI4-Stream interface error
ITS	4-11	2	ITS-GICD AXI4-Stream interface error. Disable this SM for each ITS.
PPI	12-43	2	PPI-GICD AXI4-Stream interface error. Disable this SM for each PPI.

6.9.2.2.1 Mandatory connections for safety

This section lists the connections that are required when using a non-GIC-rendered interconnect.

The remote blocks use the AXI4-Stream interface to report faults that are detected by their own Safety Mechanisms to the central GICD.

Each GIC-600AE remote block has an fmu_err_out output signal, which indicates an error within the block. You must connect this signal to the fmu_err_in input signal of the GICD. This connection provides a redundant path for error signaling from all remote GIC-600AE blocks to the FMU. The remote block keeps the fmu_err_out signal asserted until the error recovery software clears the error.

When using a GIC-rendered interconnect, the fmu_err_* fault signals connect automatically. However, when using a non-GIC-rendered interconnect, you must manually connect these signals.

The following table lists the mandatory fmu_err_out connections to make when using a non-GIC-rendered interconnect for FuSa operation.

Table 6-11: Mandatory fmu_err_out signal connections for a non-GIC-rendered interconnect

fmu_err_in[x] && block ID	Block
0	GICD
1	SPI Collator
2	Wake Request
3	Reserved
4	ITS0
5	ITS1
6	ITS2

fmu_err_in[x] && block ID	Block
7	ITS3
8	ITS4
9	ITS5
10	ITS6
11	ITS7
12	PPI0
13	PPI1
14	PPI2
15	PPI3
16	PPI4
17	PPI5
18	PPI6
19	PPI7
20	PPI8
21	PPI9
22	PPI10
23	PPI11
24	PPI12
25	PPI13
26	PPI14
27	PPI15
28	PPI16
29	PPI17
30	PPI18
31	PPI19
32	PPI20
33	PPI21
34	PPI22
35	PPI23
36	PPI24
37	PPI25
38	PPI26
39	PPI27
40	PPI28
41	PPI29
42	PPI30
43	PPI31

fault_* signal connections for a non-GIC interconnect

There are some fault_* signals that are used only by a GIC rendered interconnect. Therefore, if a non-GIC rendered interconnect is used, those fault_* signals must be tied off.

We assume that if a non-GIC rendered interconnect is used, the non-GIC interconnect either has its own method of protecting the interconnect components or this protection is not required.

The following table lists the fault_* signals and the tie-off values.

Table 6-12: fault_* tie-off signals

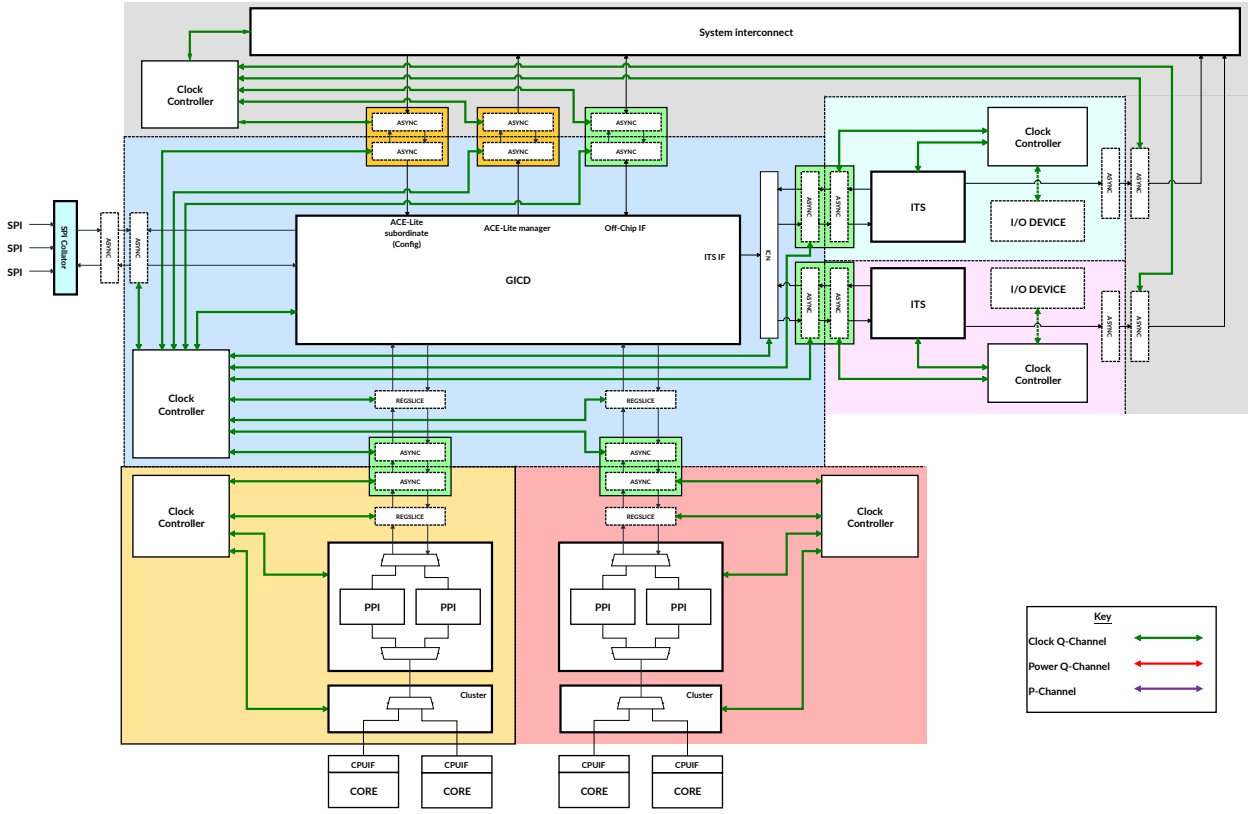
Signal	Block	Tie-off value	Description
fault_icpdp*	PPI	LOW	Carries fault information from ADB to PPI, on the icpd/icdp bus
fault_icidi*	ITS	LOW	Carries fault information from ADB to ITS, on the icid/icdi bus
fault_external*		LOW	Carries fault information from ACE-bypass switch to ITS
fault_qchannel*		LOW	Carries Q-Channel fault information from LPD-CG to ITS
fault_qchannel_pwr*		LOW	Carries Q-Channel fault information from LPD-PWR to ITS
fault_icwdw*	Wake Request	LOW	Carries fault information from ADB to Wake Request, on the icwd/icdw bus
fault_iccdc*	SPI Collator	LOW	Carries fault information from ADB to SPI Collator, on the iccd/icdc bus
fault_icpdp*	GICD	LOW	Carries fault information from ADB to GICD, on the icpd/icdp bus
fault_icidi*		LOW	Carries fault information from ADB to GICD, on the icid/icdi bus
fault_iccdc*		LOW	Carries fault information from ADB to GICD, on the iccd/icdc bus
fault_icwdw*		LOW	Carries fault information from ADB to GICD, on the icwd/icdw bus
fault_ace_switch*		LOW	Carries fault information from monolithic switch to GICD
fault_qchannel*		LOW	Carries Q-Channel fault information from LPD-CG to GICD
fault_qchannel_pwr*		LOW	Carries Q-Channel fault information from LPD-PWR to GICD

6.10 P-Channel and Q-Channel protection

The P-Channel and Q-Channel logic and connections can be complex for topologies with multiple clock or power domains.

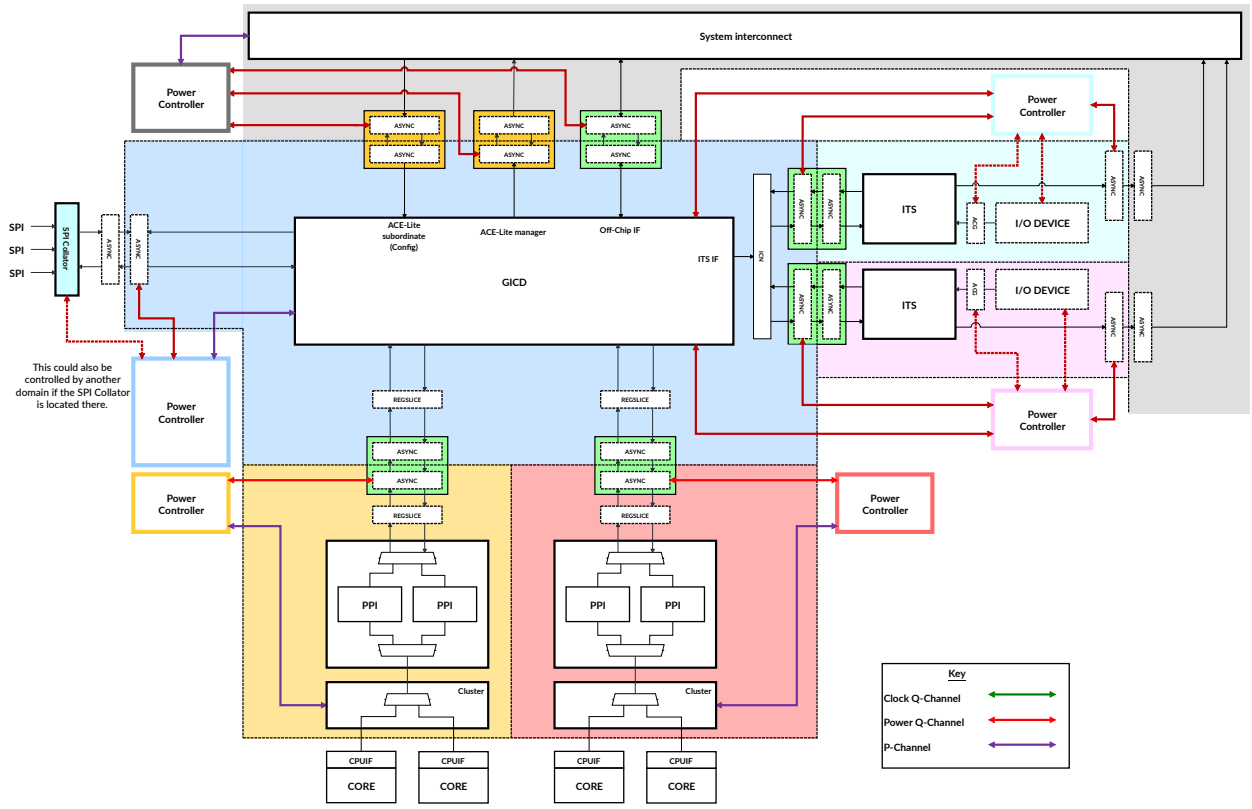
The following figure shows a top-level example of a GIC topology with multiple clock domains.

Figure 6-12: GIC topology with multiple clock domains



The following figure shows a top-level example of a GIC topology with multiple power domains.

Figure 6-13: GIC topology with multiple power domains



This example power domain hook-up has the following power domain relationships:

- Core before cluster
- Cluster before GICD
- ITS before GICD

Note

Possible scenarios also relate to making the ITS quiescent while the I/O domain is on.

- GICD before interconnect

Note

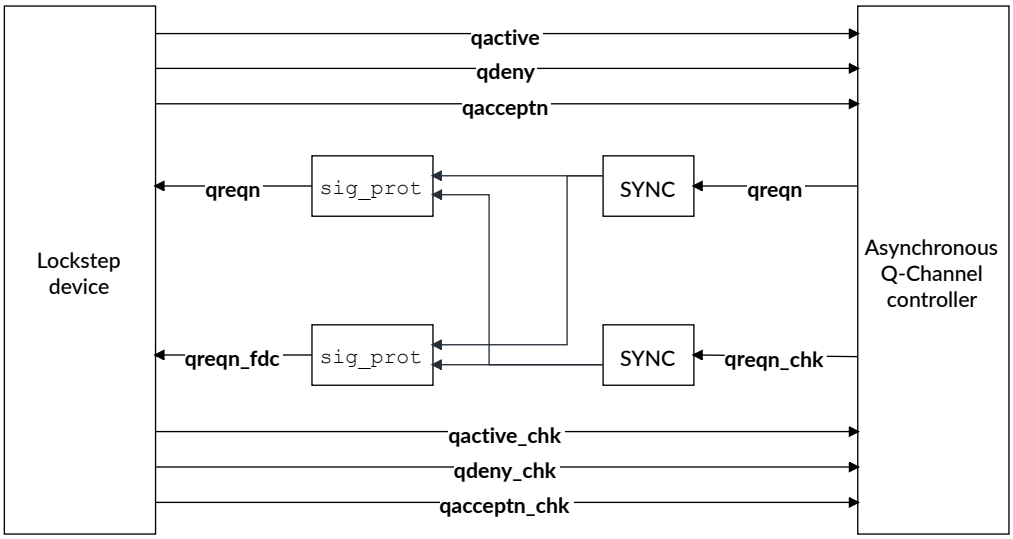
It is also beneficial to control the interconnect before the GICD. This implies different control on the bridges, either from the other side, or independent/ combined if there is no fixed relationship.

In Figure 6-12: GIC topology with multiple clock domains on page 233 and Figure 6-13: GIC topology with multiple power domains on page 234, the Q-Channel connections are made by the GIC rendering engine. The GIC uses a Q-Channel for power control in all cases except for cross/remote chip power control, which uses a P-Channel port on the Distributor.

The P-Channel and Q-Channel are protected with redundant chk signal bits with reverse polarity. Due to the 4-phase asynchronous nature of the P-Channel and Q-Channel, all signals are checked individually, except for the pstate signal. With 4-phase handshaking, all assertions must be held until handshaking feedback is received. Therefore, transient assertions are treated as faults, which are filtered by the protection logic for reliability. The protection logic prevents these faults from reaching mission mode logic and causing errors. Permanent faults, or *Stuck-At Faults* (SAFs), are detected and flagged.

The following figure shows a high-level Q-Channel example that the GIC blocks employ.

Figure 6-14: Q-Channel protection example



The qreqn and qreqn_chk signals are synchronized separately. These signals then pass through redundant `sig_prot` blocks, which contain the transient filtering logic and stuck-at checker counters.

The Q-Channel outputs are passed to the external power controller, or internal GIC LPD, with a temporal delay no greater than two cycles. The temporal delay can vary from 0-2 cycles, due to corner cases regarding clock alignment in the ADB. The P-Channel and Q-Channel AMBA® extensions allow this variation.

6.10.1 CHK bit timing

There is a hard timing requirement that the *Stuck-At Fault* (SAF) detection logic imposes.

The skew of the preqn, preqn_chk, qreqn, and qreqn_chk signals must be less than the maximum skew that the SAF detection logic allows.

***Clock Ratio* (CR)**

Equal to (GIC clock frequency) / (channel controller clock frequency).

Implementation skew

Silicon skew due to asynchronous clock domain crossings or other factors.

Temporal delay skew

Skew between lock-step primary and redundant logic blocks.

Since the GIC-600AE SAF detector counts to 64 before flagging an SAF, the permitted skew is calculated as follows:

- Maximum skew allowed = 64 / CR

Example 6-1: Q-Channel skew calculation

- GIC clock frequency = 1000MHz
- Q-Channel frequency = 125MHz

Based on these frequencies, then:

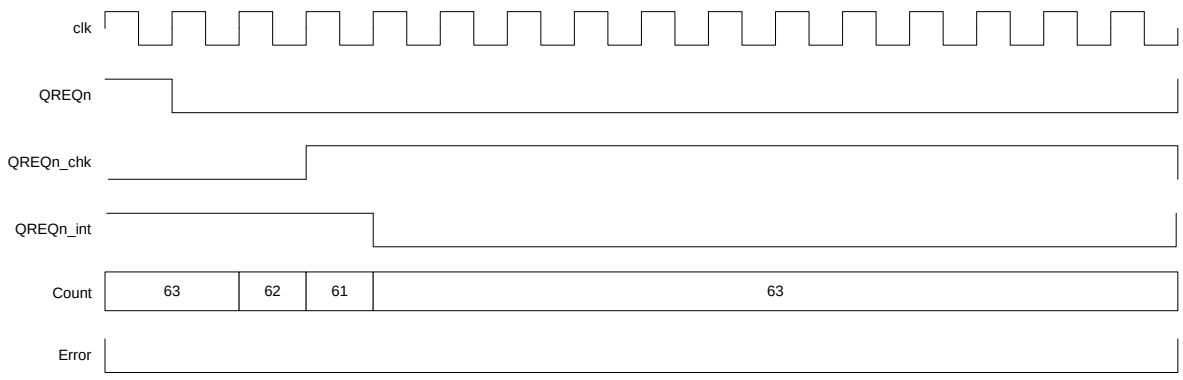
- CR = (GIC clock frequency)/(channel controller clock frequency) = 1000MHz / 125MHz = 8.
- Maximum skew allowed = 64 / CR = 64 / 8 = 8 cycles.

Therefore, the system integrator is allowed eight cycles for implementation skew and temporal delay skew that originate from the SoC Q-Channel controller.

6.10.2 Transient faults

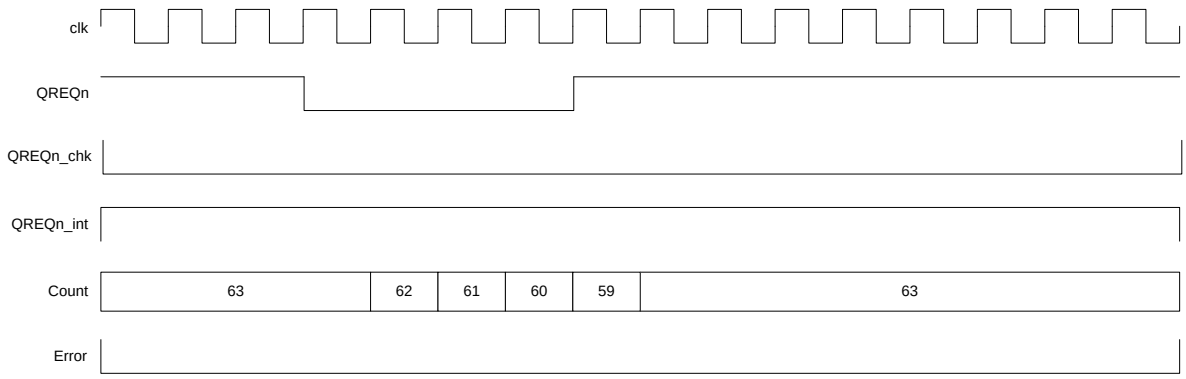
The following figure shows the normal situation with no fault.

Figure 6-15: Normal assertion of qreqn and qreqn_chk signals



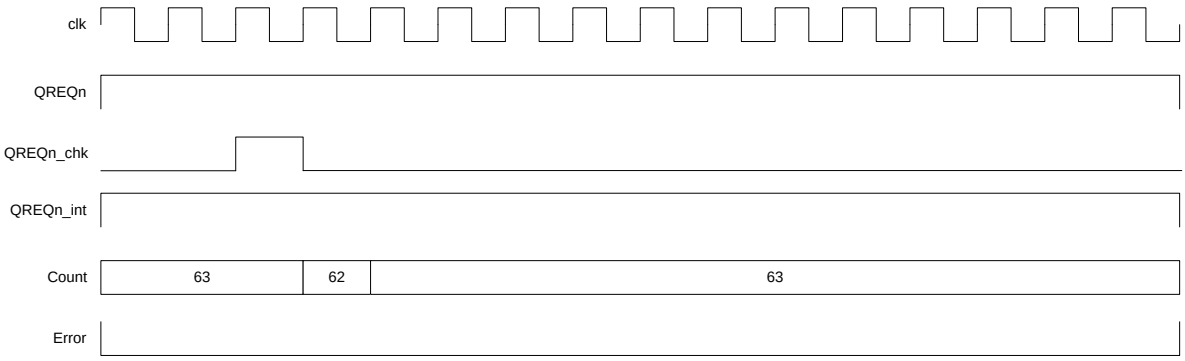
The following figure shows how a transient fault on the qreqn signal is filtered.

Figure 6-16: Transient fault on the qreqn signal



The following figure shows how a transient fault on the qreqn_chk signal is filtered.

Figure 6-17: Transient fault on qreqn_chk signal

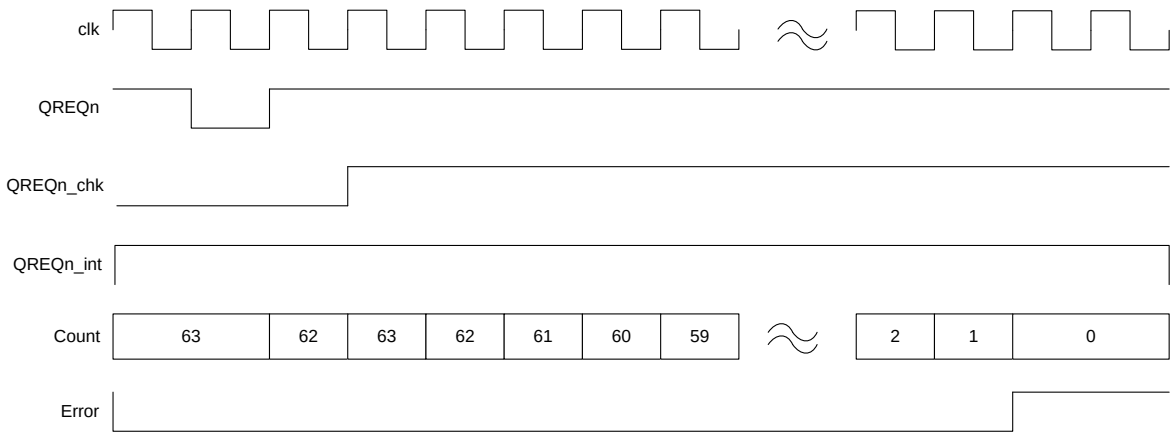


The output of the filtering logic, the qreqn_int signal, does not assert. The figures depict a version of the qreqn and qreqn_chk signals after they pass synchronizer cells. The counter depicts the operation of the SAF detector. In this example, the SAF detector is set to a value of 63 whenever the qreqn and qreqn_chk signals are the same polarity. If it detects a polarity difference between the qreqn and qreqn_chk signals, it starts counting down. If the counter reaches zero, it flags an error.

6.10.3 Stuck-at faults

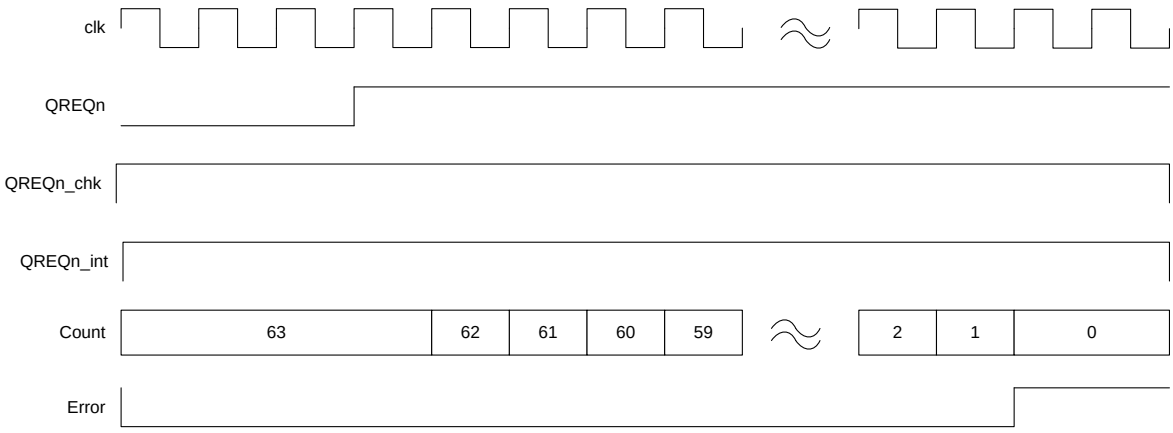
The following figure shows how the SAF detector detects a stuck-at-one error on the qreqn signal.

Figure 6-18: Stuck-at-one error on qreqn signal



The following figure shows how the SAF detector detects a stuck-at-one error on the qreqn_chk signal.

Figure 6-19: Stuck-at-one error on qreqn_chk signal



6.10.4 Disabling P-Channel and Q-Channel Safety Mechanisms

The FMU_SMEN register cannot disable the P-Channel and Q-Channel Safety Mechanisms. However, they can be disabled during design time.

During design time, you can use one of the following methods to disable these SMs:

- To disable specific P-Channel and Q-Channel SMs:
  - On the P-Channel interfaces for which you want to disable protection, tie the preqn_chk signal to the value of the !preqn signal.
  - On the Q-Channel interfaces for which you want to disable protection, tie the qreqn_chk signal to the value of the !qreqn signal.
- To disable all P-Channel and Q-Channel SMs in the GIC-600AE, set `FUSA_DISABLE_PQCHAN_PROT`=1. For a list of the SMs that FMU_SMEN cannot disable, see 6.2.5.1 Enabling or disabling a Safety Mechanism on page 202.

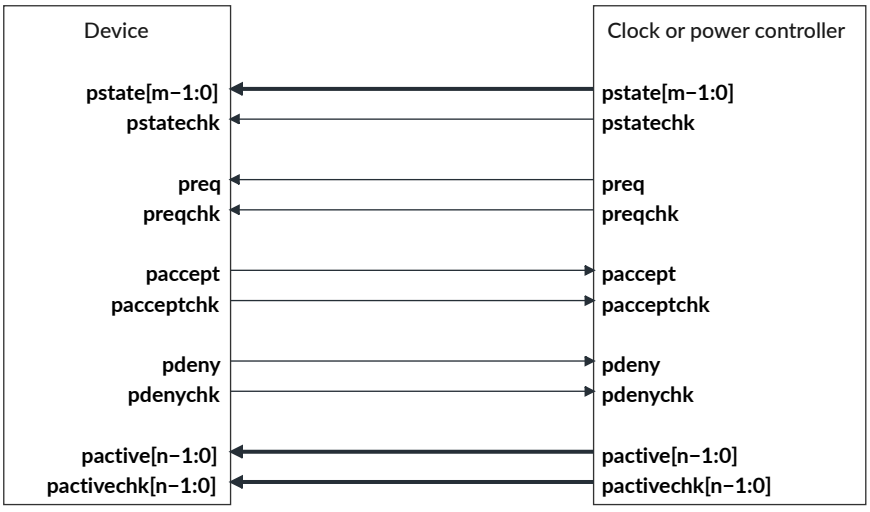
6.10.5 P-Channel

This section contains information for P-Channel protection.

6.10.5.1 P-Channel signaling

The following figure shows the device and controller signal mappings, including the added chk signals.

Figure 6-20: P-Channel device and controller signal mappings



Except for the pstate signal, each P-Channel signal bit has a corresponding chk signal bit with inverted polarity.

6.10.5.1.1 Capturing pstate signal

In a non-FuSa case, the pstate signal is captured with a synchronized preq signal. In the FuSa case, we must wait until both the preq and preqchk signals are at the correct level before sampling the pstate and pstatechk signals. At capture, the pstatechk signal is then tested against the pstate signal.

The implied timing constraint is similar to the non-FuSa constraint:

Non-FuSa P-Channel constraint

pstate maximum delay < preq signal delay + 2 capture cycles

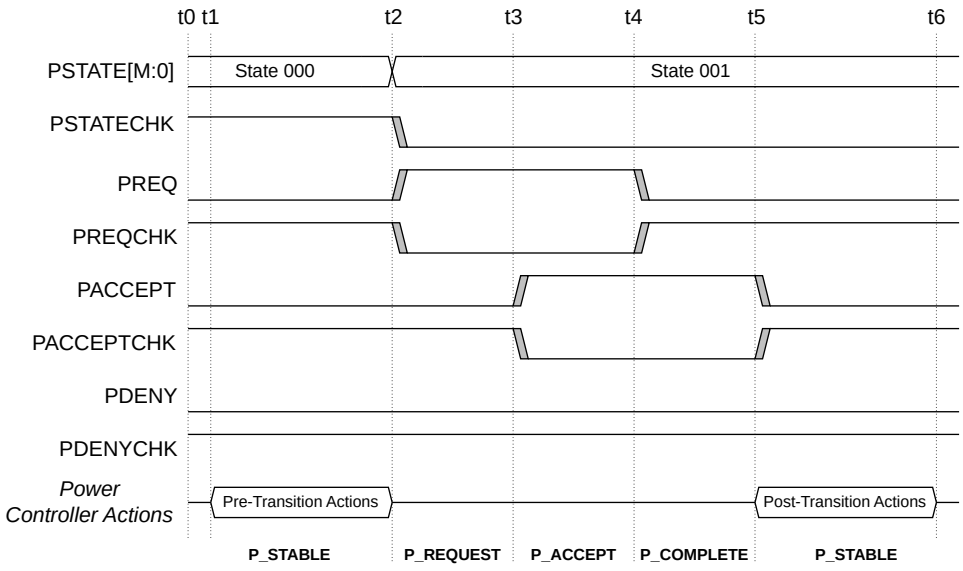
FuSa P-Channel constraint

- pstate and pstatechk maximum delay < preq signal delay + 2 capture cycles
- pstate and pstatechk maximum delay < preqchk signal delay + 2 capture cycles

6.10.5.2 P-Channel acceptance

The following figure shows the opposite polarity of the chk signal bits and the pstatechk signal bit during the P-Channel acceptance and entry sequence.

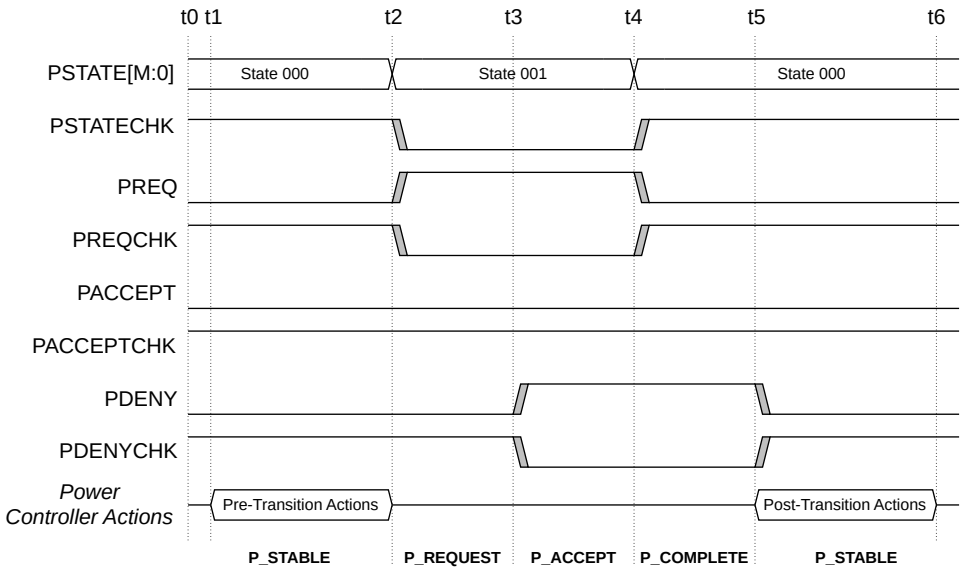
Figure 6-21: P-Channel acceptance



6.10.5.3 P-Channel denial

The following figure shows the opposite polarity of the chk signal bits and the pstatechk signal bit during the P-Channel denial sequence.

Figure 6-22: P-Channel denial



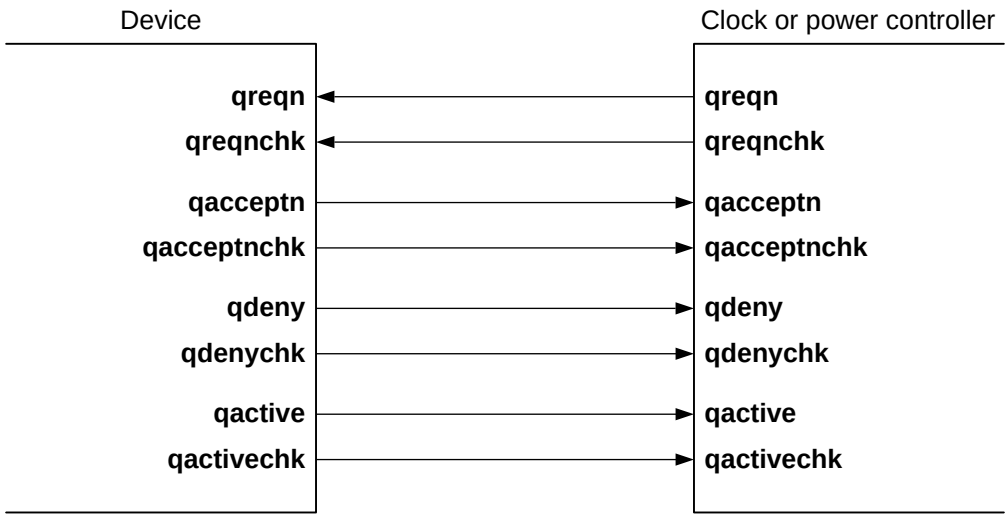
6.10.6 Q-Channel

This section contains information for Q-Channel protection.

6.10.6.1 Q-Channel signaling

The following figure shows the Q-Channel device and controller signal mappings, including the added chk signals.

Figure 6-23: Q-Channel device and controller signal mappings

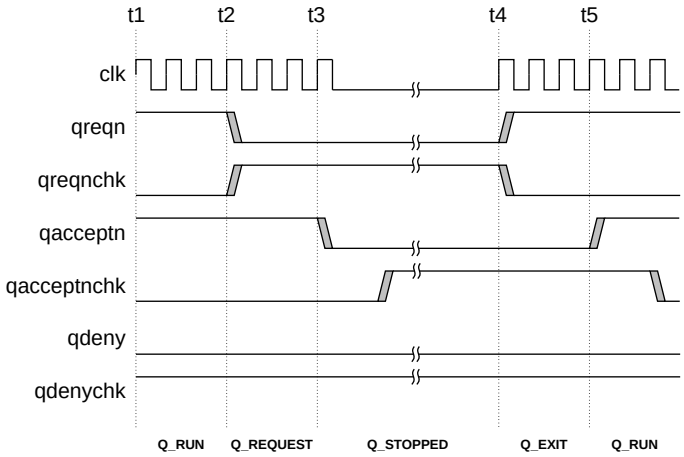


Each Q-Channel signal bit has a corresponding chk signal bit with inverted polarity.

6.10.6.2 Q-Channel acceptance

The following figure shows the opposite polarity of the chk signal bits during the Q-Channel entry, acceptance, and exit sequence.

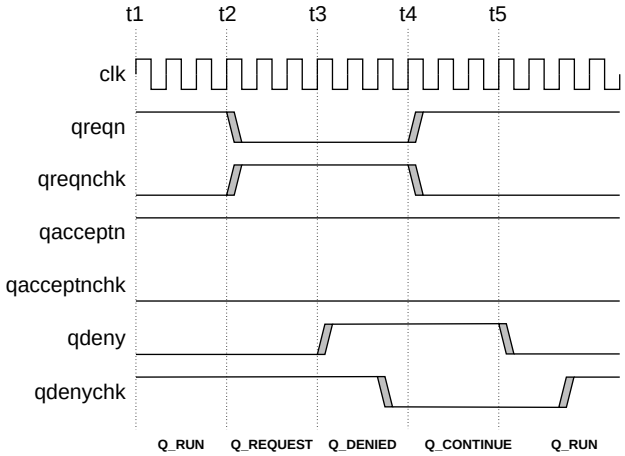
Figure 6-24: Q-Channel acceptance



6.10.6.3 Q-Channel denial

The following figure shows the opposite polarity of the chk signal bits during the Q-Channel denial sequence.

Figure 6-25: Q-Channel denial



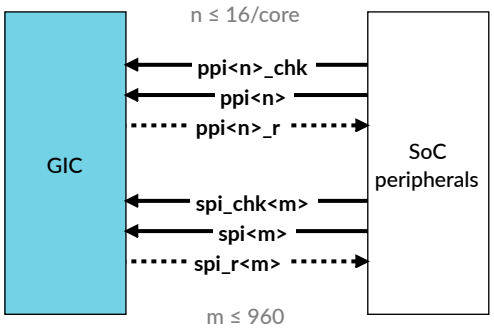
6.11 PPI and SPI interrupt interface protection

PPIs and SPIs are protected by _chk parity signal bits, which can be optionally added. A _chk signal bit is added for each physical SPI and PPI signal rendered when setting the following parameters:

- `spi_wires`
- All PPI parameters that affect the number of PPI ports on the Redistributors

The following figure shows the signals that relate to PPI and SPI interrupt interface protection.

Figure 6-26: PPI and SPI interrupt interface protection



The _chk signal bits have inverse polarity from the ppi and spi signals that they protect. The ppi and spi input signals and their corresponding _chk parity signal bit are considered asynchronous inputs. The GIC-600AE contains specific logic to handle asynchronous uncertainty on the ppi/ppi_chk and spi/spi_chk signal pairs.

6.11.1 PPI and SPI CHK bit timing

It is permissible for the chk signal bit to arrive on a different cycle than the ppi/spi signal bit that it protects.

The SAF detector defines the upper limit of the allowed skew. If the SAF detector detects a difference between the chk signal bit and the ppi/spi signal bit that it protects, it starts counting. If it reaches the skew limit, the SAF detector assumes a SAF, and the GIC FMU flags the fault.

***Clock Ratio* (CR)**
: Equal to (GIC clock frequency)/(channel controller clock frequency)

Implementation skew
: Silicon skew due to asynchronous clock domain crossings or other factors

Temporal delay skew
: Skew between lock-step primary and redundant logic blocks

Since the GIC-600AE SAF detector counts to 16 before flagging an SAF, the permitted skew is calculated as follows:

- Maximum skew allowed = 16/CR.

Example 6-2: Interrupt skew calculation

- GIC clock frequency = 800MHz
- Interrupt source frequency = 200MHz

Based on these frequencies, then:

- CR = (GIC clock frequency)/(channel controller clock frequency) = 800MHz / 200MHz = 4
- Maximum skew allowed = 16 / CR = 16 / 4 = 4 cycles

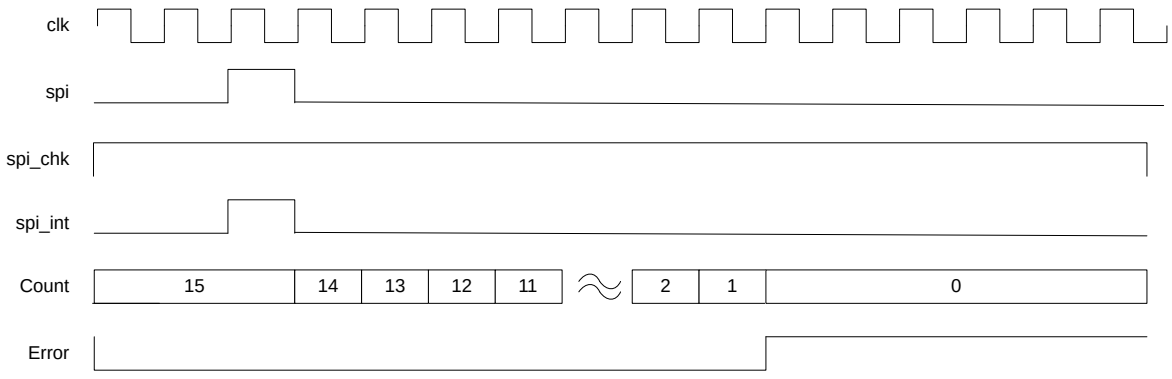
Therefore, the system integrator is allowed four cycles for implementation skew and temporal delay skew that originate from the interrupt source IP.

6.11.2 PPI and SPI transient faults

If a corresponding spi_chk signal edge is not detected within 16 cycles, a transient fault is reported.

The following figure shows an assertion of an spi signal that causes a transient fault.

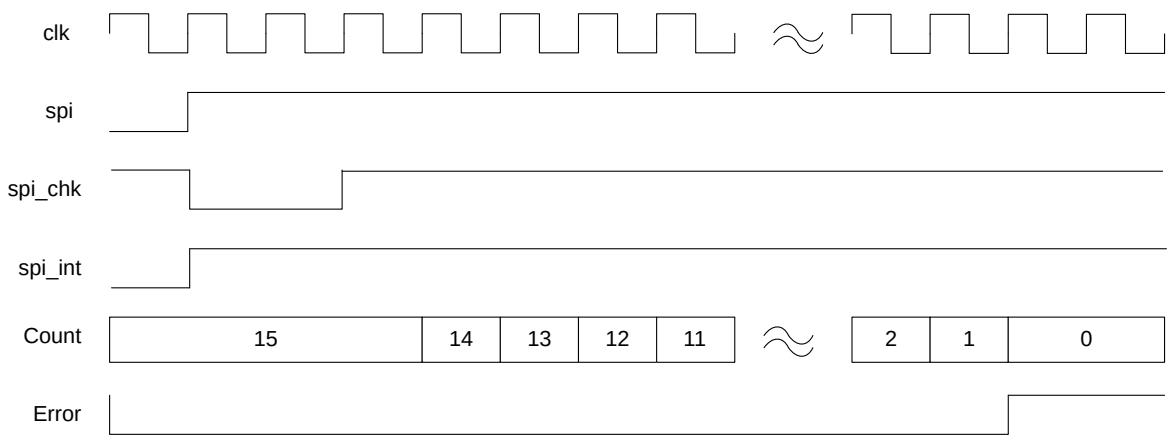
Figure 6-27: Transient fault on spi



6.11.3 PPI and SPI stuck-at faults

The following figure shows the case where an spi signal is stuck for 16 cycles, prompting the FMU to flag a fault.

Figure 6-28: Stuck-at-one fault on spi signal



The spi_int signal has inverse polarity.

6.11.4 PPI and SPI configuration parameters

The following parameters apply to PPI and SPI interrupt interface protection.

- `fusa_spi_prot`
- `fusa_ppi_prot`

For more information about these parameters, see *Configuration options* in the *Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Configuration and Integration Manual*.

6.12 Systematic fault watchdog protection

The GIC-600AE contains a watchdog-based PING/ACK mechanism that guards against systematic errors on the interconnect.

The Distributor pings each GIC block in a round-robin sequence and waits for a response. If a response is not received within a programmable timeout window, a fault is reported. This PING/ACK mechanism can guard against:

- Lockup on the interconnect that connects the GIC blocks.

- Possible lockup on external buses that causes the GIC blocks and internal interconnect to stall.

The source of the lockup might be software issues, DoS issues, or systematic faults in the silicon.

Related information

Ping mechanisms on page 202

6.13 DFT protection

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

6.13.1 MBIST

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

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

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

Table 6-13: Protected MBIST inputs

Signal	Protection	Notes
mbistreq	Assertion detection	If the mbistreq signal is asserted when the GIC block is in functional mode, the GIC detects and reports it. If this assertion happens in MBIST mode, we assume that software ignores and clears the fault, either through a reset or the FMU clearing mechanism. Alternately, to prevent the fault from asserting, software can disable the SM before entering MBIST mode.
nmbistreset	Duplication	The reset is duplicated. The duplicated reset signal is nmbistreset_fdc. For a reset to occur, both the nmbistreset and nmbistreset_fdc signals must assert. See 6.5.2 Resets on page 213 for more information.

The MBIST interface pins themselves are unchanged from GIC-600.

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

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

6.13.2 ATPG/Scan

All DFT/ATPG input ports are duplicated.

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

The following table shows the duplicate ports.

Table 6-14: Duplicate ATPG input ports

clk scan input	clk_fdc scan input	Description
dftrstdisable	dftrstdisable_fdc	Prevents reset from asserting when reset generation flops are scanned
dftcgen	dftcgen_fdc	Ensures that scanned flops receive a clock, by forcing the clock gate enable
dftramhold	dftramhold_fdc	Asserting prevents RAM access during ATPG. This can reduce coverage for logic in the RAM shadow.
dftse	dftse_fdc	Scan enable

6.13.3 LBIST

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

6.14 Generic fault inputs

Each GIC block has generic fault inputs that allow the SoC integrator to connect and flag external faults through the FMU.

For instance, an SoC integrator might have an external Safety Mechanism physically located next to a GIC block. The SoC integrator can connect the fault signal from this external SM to the usr0_err or usr1_err input signals of the GIC block. If a fault occurs, the GIC flags and reports a fault in the same manner it does with internal faults.

The following guidelines are associated with this safety feature:

- The fault signal can be pulsed or level. However, there is no mechanism for the GIC to clear the fault state in the external mechanism. Therefore a pulse is the recommended implementation.
- To be captured correctly, the fault pulse must be held for at least one GIC block clock cycle.
- A captured fault is reported in the GIC block error record with a User Custom SMx error code.
- There are two generic fault inputs on each GIC block, the usr0_err and usr1_err signals.
- Tie unused inputs LOW.

6.15 Configuration and parameters

This section summarizes the differences between GIC-600 and GIC-600AE.

Added configurations to GIC-600

- The `fusa_comp_dup` parameter can add one more gate into the comparator paths. For more information, see 6.6.1 Comparators on page 218.
- The `fusa_axis_int_busprot_type` parameter indicates the bus protection to be deployed on the AXI4-Stream interfaces between the GIC-600AE blocks.
- The `fusa_spi_prot` parameter indicates whether parity _chk signal bit protection is deployed to SPI interrupt inputs.
- The `fusa_ppi_prot` parameter indicates whether parity _chk signal bit protection is deployed to PPI interrupt inputs.
- The `fusa_disable_pqchan_prot` parameter can disable P-Channel and Q-Channel Safety Mechanisms.
  For more information, see 6.10.4 Disabling P-Channel and Q-Channel Safety Mechanisms on page 238.

For more information about these parameters, see the *Configuration options* chapter of the *Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Configuration and Integration Manual*.

Reduced configurations from GIC-600

- The GIC-600AE does not support the option to have RAM macros (compiled memories) without ECC.
- The GIC-600AE does not render logic on external interfaces, including the external GIC-Stream-AXI4-Stream interface.
- The GIC-600AE can render the `msi_64` module, but it is not protected. This block is an optimization and is not required for correct operation of the GIC-600AE.
- On GIC-600AE, the supported range of values for `ppi_count` is 1-32.
- On GIC-600AE, the supported range of values for `its_count` is 1-8.

For more information about these considerations, see the *Configuration options* chapter of the *Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Configuration and Integration Manual*.

Appendix A Signal descriptions

This appendix describes the input and output signals.

A.1 Common control signals

The following table shows the GIC-600AE common control signal set.

Signal definitions

Table A-1: Common control signals

Signal	Direction	Description
[<domain>]clk	Input	Clock input
[<domain>]reset_n	Input	Active-LOW reset
dbg_[<domain>]reset_n	Input	Active-LOW reset for the PMU and error records. This signal is only present for the domain that contains the Distributor.

Test signals		
Signal	**Direction**	**Description**
dftrstdisable	Input	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		
Signal	**Direction**	**Description**
[<domain>_]mbistack	Output	MBIST mode ready. GIC-600AE acknowledges that it is ready for MBIST testing.
[<domain>_]mbistreq	Input	MBIST mode request. Request to GIC-600AE to enable MBIST testing. This signal must be tied LOW during functional operation.
[<domain>_]nmbistreset	Input	Resets MBIST logic. Resets functional logic to enable MBIST operation by an active-LOW signal. This signal must be tied HIGH during functional operation.
[<domain>_]mbistaddr[variable:0][13]	Input	Logical address. The width is based on the RAM with the largest number of words. You must drive the most significant bits to zero when accessing RAMs with fewer address bits.
[<domain>_]mbistindata[variable:0][13]	Input	Data in. Write data. Width that is based on the RAM with the largest number of data bits.
[<domain>_]mbistoutdata[variable:0][13]	Output	Data out. 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	

[13] The variable is configuration-dependent.

MBIST controller signals		
Signal	**Direction**	**Description**
[<domain>_]mbistarray[variable:0][13]	Input	Array selector. This signal controls which RAM array is accessed. For the single RAM configuration, this port is unused. This signal is not present on a block containing only one RAM.
[<domain>_]mbistcfg	Input	MBIST ALLMODE enable. When enabled, allows simultaneous access to all RAM arrays for maximum array power consumption. This signal is not present on a block containing only one RAM.

A.2 Power control signals

The following table shows the GIC-600AE power control signals.

Signal definitions

Table A-2: Power control signals

Signal	Direction	Description
cpu_active[_<ppi_block>][_<bus>][<cpus>−1:0]	Input	Indicates if the core is active and not in a low-power state such as retention. This signal is used for lowering the priority of selection for 1 of N SPIs. There is 1 bit per core on the ICC bus. See 4.6.2 Processor core power management on page 57.
wake_request[<cpus>−1:0]	Output	Wake Request signal to power controller indicating that an interrupt is targeting this core and it must be woken. When asserted, the wake_request signal is sticky unless the Distributor is put into the gated state.
wake_request_chk[<cpus>−1:0]	Output	Odd parity protection bits

SPI Collator Q-Channel device interface for power control		
Signal	**Direction**	**Description**
qreqn_col	Input	Q-Channel device interface to flush out the path between the SPI Collator and the Distributor to aid in power down. When asserted, messages are not sent to the Distributor until low-power state is exited. **Note:** It is only safe to stop the SPI Collator clock if all interrupts are level sensitive, or if edge-triggered interrupts are pulse extended into the SPI Collator.
qacceptn_col	Output	
qdeny_col	Output	
qactive_col	Output	

ITS Q-Channel device interfaces for power control		
Signal	**Direction**	**Description**
qreqn_its[<its>]	Input	Required to flush out the path between the ITS and the Distributor. There is one Q-Channel for each ITS. All Distributor ITS Q-Channels are combined as a single set of vectored signals, qreqn_its[its_count−1:0]. The `its_count` parameter sets the number of ITS blocks on the chip. These signals are not present in monolithic configurations where the Distributor and ITS share an ACE-Lite port.
qacceptn_its[<its>]	Output	
qdeny_its[<its>]	Output	
qactive_its[<its>]	Output	

ITS Q-Channel device interface for clock control		
Signal	**Direction**	**Description**
qreqn	Input	Q-Channel device interface for clock gating of an ITS. The qreqn signal is synchronized into the GIC-600AE. This bus must be treated asynchronously.
qacceptn	Output	
qdeny	Output	
qactive	Output	
qactive_gicd	Output	This signal is HIGH when the ITS requires its AXI4-Stream bus to the Distributor to be active. If the Distributor is powered down, the system can use this signal to wakeup the Distributor.

Distributor Q-Channel device interface for clock control		
Signal	**Direction**	**Description**
qreqn	Input	Q-Channel device interface for clock gating of the Distributor. The qreqn signal is synchronized into the GIC-600AE. This bus must be treated asynchronously.
qacceptn	Output	
qdeny	Output	
qactive	Output	

PPI block Q-Channel device interface for clock control		
Signal	**Direction**	**Description**
qreqn	Input	Q-Channel device interface for clock gating of a Redistributor. The qreqn signal is synchronized into the GIC-600AE. This bus must be treated asynchronously.
qacceptn	Output	
qdeny	Output	
qactive	Output	

Q-Channel device interfaces for clock control		
Signal	**Direction**	**Description**
[<domain_>]clkqreqn	Input	Q-Channel device interface for clock gating of everything in the domain. The [<domain_>]clkqreqn signal is synchronized into the GIC-600AE. This bus must be treated asynchronously.
[<domain_>]clkqacceptn	Output	
[<domain_>]clkqdeny	Output	
[<domain_>]clkqactive	Output	

Q-Channel ADB-400 device interfaces for power control		
Signal	**Direction**	**Description**
[<domain_>]pwrqreqn	Input	Q-Channel device interface for the CoreLink™ ADB-400 AMBA® Domain Bridge power interface within the domain.
[<domain_>]pwrqacceptn	Output	
[<domain_>]pwrqdeny	Output	
[<domain_>]pwrqactive	Output	

P-Channel device interface for chip-level save and restore		
Signal	**Direction**	**Description**
preq	Input	This P-Channel device interface is only present in multichip configurations. See 4.16.4 Power control and P-Channel on page 91. The preq signal is synchronized into the GIC-600AE. The pstate signal must be stable when the preq signal is asserted. This bus must be treated asynchronously.
pstate[4:0]	Input	
paccept	Output	
pdeny	Output	
pactive	Output	

A.3 Interrupt signals

The GIC-600AE has interrupt signals for SPIs and PPIs.

Signal definitions

Table A-3: Interrupt signals

Signal	Direction	Description
ppi<n>[_<ppi_block>][_<bus>][<cpus>-1:0] If there are: • 8 PPIs per core, n is 22-27, 29, or 30. • 12 PPIs per core, n is 20-31. • 16 PPIs per core, n is 16-31.	Input	PPI input wires for interrupt <n>. One bit per core. The PPIs for each core are independent and are typically used for peripherals that are not shared between cores. For example, timers on the core typically use PPIs. By default, PPIs are active-LOW. The GIC provides top-level RTL parameters so that a PPI can be active-HIGH. The GIC also provides top-level RTL parameters so that a PPI can be synchronized to the clk signal. By default, PPIs are level-sensitive interrupts. However, software can change an interrupt to be edge triggered by programming the GICR_ICFGR1 register.
ppi<n>_r_[_<ppi_block>][_<bus>]	Output	PPI output after synchronization and edge detection. You can use these signals to create pulse extenders for edge-triggered interrupts that cross clock domains.
spi[spi_wire-1:0] The `spi_wire` configuration parameter controls the number of SPIs.	Input	This signal is the number of SPI wires that the GIC supports. **Note:** This is not the same as the number of SPIs supported because they could be message-based only or be on another chip. By default, SPIs are active-HIGH. The GIC provides top-level RTL parameters so that an SPI can be active-LOW. The GIC also provides top-level RTL parameters so that an SPI can be synchronized to the clk signal.
spi_r[spi_wire-1:0] The `spi_wire` configuration parameter controls the number of SPIs.	Output	SPI output after synchronization and edge detection. Can be used for cross-domain pulse detection.

A.4 CPU interface signals

The CPU interface signals of a cluster connect to a Redistributor using two GIC Stream interfaces.

In the following tables, <ppi_num>, <bus>, and <cpus> are configuration options that are set using the ppi_ref, bus, and cpus parameters. See the *Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Configuration and Integration Manual* for more information.

Signal definitions

Table A-4: CPU interface signals

GIC Stream-compliant bus for communication from a cluster to a Redistributor		
Signal	**Direction**	**Description**
icctready[_<ppi_num>][_<bus>]	Output	This GIC Stream-compliant bus is fully credited and can be sent over any free-flowing interconnect. For more information, see *Table A-2 CPU interface to upstream Redistributor interface* in the *GIC Stream Protocol interface Appendix* of the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4. If the cluster issues IDs on the ICCTID signal with values other than <cpus−1:0>, then the behavior is unpredictable.
icctvalid[_<ppi_num>][_<bus>]	Input	
icctdata[_<ppi_num>][_<bus>][15:0]	Input	
icctid[_<ppi_num>][_<bus>][<cpus>−1:0]	Input	
icctlast[_<ppi_num>][_<bus>]	Input	
icctwakeup[_<ppi_num>][_<bus>]	Input	Registered wake signal to indicate that a message is arriving or is about to arrive on the icc bus

GIC Stream-compliant bus for communication from a Redistributor to a cluster		
Signal	**Direction**	**Description**
iritready[_<ppi_num>][_<bus>]	Input	This GIC Stream-compliant bus is fully credited and can be sent over any free-flowing interconnect. For more information, see *Table A-1 Redistributor to downstream CPU interface* in the *GIC Stream Protocol interface Appendix* of the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.
iritvalid[_<ppi_num>][_<bus>]	Output	
iritdata[_<ppi_num>][_<bus>][15:0]	Output	
iritdest [_<ppi_num>][_<bus>][<cpus>−1:0]	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 of the cluster

A.5 ACE-Lite interface signals

The following table shows the GIC-600AE ACE-Lite signals.

Table A-5: ACE-Lite subordinate interface signals

Subordinate write address channel signals		
Signal	**Direction**	**Description**
There are multiple versions of this bus. Buses that have _its[_<num>] are dedicated ITS subordinate 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[n:0]	Input	Optional User signal. For GICD interface, n = `axis_awuser_width`−1. For an ITS switch, n is a minimum of `did_width`−1. Indicates the *DeviceID* of writes to GITS_TRANSLATER if MSI_64 is not configured.
awaddr_[its[_<num>]]_s[n:0]	Input	The write address gives the address of the first transfer in a write burst transaction. Where n = `axis_addr_width`−1.
awid_[its[_<num>]]_s[n:0]	Input	This signal is the identification tag for the write address group of signals. Where n = `axis_wid_width`−1.
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 subordinate 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[3:0]	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.

Subordinate write data channel signals		
Signal	**Direction**	**Description**
wstrb_[its[_<num>]]_s[n:0]	Input	This signal indicates which byte lanes hold valid data. There is one write strobe bit for every 8 bits of the write data bus.
wdata_[its[_<num>]]_s[n:0]	Input	Write data, where n = `axis_data_width`−1
wvalid_[its[_<num>]]_s	Input	This signal indicates that valid write data and strobes are available.
wready_[its[_<num>]]_s	Output	This signal indicates that the subordinate can accept the write data.
wlast_[its[_<num>]]_s	Input	This signal indicates the last transfer in a write burst.

Subordinate write response channel signals		
Signal	**Direction**	**Description**
bid_[its[_<num>]]_s[n:0]	Output	This signal is the ID tag of the write response. Where n = `axis_wid_width`−1.
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 manager can accept a write response.

Subordinate write response channel signals		
Signal	**Direction**	**Description**
bresp_[its[_<num>]]_s[1:0]	Output	This signal indicates the status of the write transaction.
buser_[its[_<num>]]_s[n:0]	Output	Write response User signal, where n = axis_buser_width–1.

Subordinate read address channel signals		
Signal	**Direction**	**Description**
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[n:0]	Input	The read address gives the address of the first transfer in a read burst transaction. Where n = axis_addr_width–1.
arid_[its[_<num>]]_s[n:0]	Input	This signal is the identification tag for the read address group of signals. Where n = axis_rid_width–1.
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[n:0]	Input	This signal indicates some user-defined sideband content that transfers with the read address. The GIC-600AE ignores aruser data that it receives on the GICD (Distributor) subordinate port or the ITS page containing the GITS_TRANSLATER register. Where n = 0 on the GICD interface and n = axis_aruser_width–1 on an ITS interface.
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 subordinate is ready to accept an address and associated control signals.

Subordinate read data channel signals		
Signal	**Direction**	**Description**
rid_[its[_<num>]]_s[n:0]	Output	This signal is the identification tag for the read data group of signals that the subordinate generates. Where n = axis_rid_width–1.
rdata_[its[_<num>]]_s[n:0]	Output	Read data, where n = axis_data_width–1
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 manager can accept the read data and response information.
ruser_[its[_<num>]]_s[n:0]	Output	Read response User signal, where n = axis_ruser_width–1

[14] The variable is configuration-dependent.

Table A-6: ACE-Lite manager interface signals

Manager write address channel signals. Only present if LPI support is configured.		
Signal	**Direction**	**Description**
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][14]	Output	The write address gives the address of the first transfer in a write burst transaction.
awid_[its[_<num>]]_m[variable:0][14]	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[3:0]	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_[its[_<num>]]_m[variable:0][14]	Output	Optional User signal. For an ITS switch, variable is a minimum of `did_width`−1.

Manager write data channel signals. Only present if LPI support is configured.		
Signal	**Direction**	**Description**
wstrb_[its[_<num>]]_m[variable:0][14]	Output	This signal indicates which byte lanes hold valid data. There is one write strobe bit for every 8 bits of the write data bus.
wdata_[its[_<num>]]_m[variable:0][14]	Output	Write data
wvalid_[its[_<num>]]_m	Output	This signal indicates that valid write data and strobes are available.
wready_[its[_<num>]]_m	Input	This signal indicates that the subordinate can accept the write data.
wlast_[its[_<num>]]_m	Output	This signal indicates the last transfer in a write burst.

Manager write response channel signals. Only present if LPI support is configured.		
Signal	**Direction**	**Description**
bid_[its[_<num>]]_m[variable:0][14]	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.
buser_[its[_<num>]]_m[n:0]	Input	Write response User signal, where `n` = `axis_buser_width`−1

Manager read address channel signals. Only present if LPI support is configured.		
Signal	**Direction**	**Description**
araddr_[its[_<num>]]_m[variable:0][14]	Output	The read address gives the address of the first transfer in a read burst transaction.
arid_[its[_<num>]]_m[variable:0][14]	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 subordinate 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][14]	Output	Optional User signal. For an ITS switch, variable is a minimum of `did_width`−1.

Manager read data channel signals. Only present if LPI support is configured.		
Signal	**Direction**	**Description**
rid_[its[_<num>]]_m[variable:0][14]	Input	This signal is the identification tag for the read data group of signals generated by the subordinate.
rdata_[its[_<num>]]_m[variable:0][14]	Input	Read data
rresp_[its[_<num>]]_m[1:0]	Input	This signal indicates the status of the read transfer.
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 manager can accept the read data and response information.
ruser_[its[_<num>]]_m[n:0]	Input	Read response User signal, where `n` = `axis_ruser_width`−1

A.6 Miscellaneous signals

The following table shows the GIC-600AE miscellaneous signals.

Signal definitions

Table A-7: Miscellaneous signals

Signal	Direction	Description
chip_id[<CHIP_ID_WIDTH>−1:0]	Input	An ID number that identifies the chip in the system. Only present if there is more than one chip in the system.
ppi_id[15:0]	Input	An ID number that identifies the Redistributor in the system. Software can read the GICR_CFGID0 register to access the value of this signal.

Signal	Direction	Description
its_id[7:0]	Input	An ID number that identifies the ITS block in the system. Software can read the GITS_CFGID register to access the value of this signal. It must be tied to the ic<x>dtdest signal value that is used to read the ITS using the AXI4-Stream interface.
fault_int	Output	Fault handling interrupt. The GIC-600AE can deliver this interrupt internally but the output is provided for any other device such as a system control processor that does not receive normal interrupts from the GIC. See 4.15.5 Error recovery and fault handling interrupts on page 70.
err_int	Output	Error handling interrupt. The GIC-600AE can deliver this interrupt internally but the output is provided for any other device such as a system control processor that does not receive normal interrupts from the GIC. See 4.15.5 Error recovery and fault handling interrupts on page 70.
pmu_int	Output	PMU counter overflow interrupt. This signal is a level-sensitive interrupt. The GIC-600AE can deliver this interrupt internally but the output 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. See Overflow interrupt on page 68.
sample_req	Input	Request from a *Cross Trigger Interface* (CTI) to sample the PMU counters. Equivalent to writing to the GICP_CAPR register. See Snapshot on page 68 for more information.
sample_ack	Output	This signal goes HIGH when the GIC acknowledges the PMU sample request from the CTI
gict_allow_ns	Input	From reset, this tie-off signal controls whether Non-secure software can access the GICT Error Record registers
gicp_allow_ns	Input	From reset, this tie-off signal controls whether Non-secure software can access the GICP PMU registers
gicd_page_offset	Input	From reset, this tie-off signal controls the page address bits[x:16] of the GICD page. Only present in monolithic configurations. See Page offset on page 96.
its_transr_page_offset	Input	From reset, this tie-off signal controls the page address of the GITS_TRANSLATER register. Only present in monolithic configurations. See 4.12 MSI-64 on page 65 and Page offset on page 96.
target_address<n>[ADDR_WIDTH−17:0]	Input	Modifies the address map to ensure only writes to the correct location can trigger MSI requests. Only present when the bypass switch is configured. <n> represents an ITS identifier. Specifies the 64K page address that includes the GITS_TRANSLATER register address, and is matched against the axaddr[ADDR_WIDTH−1:16] signal. See 3.3.1 ITS ACE-Lite subordinate interface on page 36.
msi_translator_page	Input	The target page address of the GITS_TRANSLATER register. The MSI-64 Encapsulator does not support an msi_transalator_page signal value of 0. See 3.4 MSI-64 Encapsulator on page 40.
msi64_translator_page	Input	The target address of the 64-bit GITS_TRANSLATER register. This page must be at a different location to the msi_translator_page signal and at a location that is known only to the hypervisor. The hypervisor must be able to protect the page from accesses from devices and processors that can spoof incorrect DeviceIDs. See 3.4 MSI-64 Encapsulator on page 40 and 4.12 MSI-64 on page 65.
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. See 3.4 MSI-64 Encapsulator on page 40 and 3.4.1 MSI-64 ACE-Lite interfaces on page 40.

A.7 Interblock AXI4-Stream interface signals

The GIC-600AE Distributor uses AXI4-Stream interfaces to communicate with the other GIC components.

The following table shows the GIC-600AE interblock signals.

Signal definitions

Table A-8: Interblock AXI4-Stream signals

Distributor to Redistributor AXI4-Stream interface, icdp bus		
Signal	**Direction**	**Description**
icdptready	Input	AXI4-Stream compliant bus for communication between the Distributor and a Redistributor. The interface is fully credited and can be sent over any free-flowing interconnect. The signal width is set using the `ppi_stream_data_width` configuration option.
icdptvalid	Output	
icdptdata[<ppi_stream_data_width>−1:0]	Output	
icdptlast	Output	
icdptwakeup	Output	Registered wake signal to indicate that a message is arriving or is about to arrive on the icdp bus
icdptkeep[(<ppi_stream_data_width>/8)−1:0]	Output	Indicates the data bytes that must be transferred. This signal is only present on the Distributor.
icdptdest[variable:0]	Output	Specifies the destination Redistributor block. This signal is only present on the Distributor.

Redistributor to Distributor AXI4-Stream interface, icpd bus		
Signal	**Direction**	**Description**
icpdtready	Input	AXI4-Stream compliant bus for communication between a Redistributor and the Distributor. The interface is fully credited and can be sent over any free-flowing interconnect. The signal width is set using the `gic_stream_width` configuration option.
icpdtvalid	Output	
icpdtdata[<gic_stream_width>−1:0]	Output	
icpdtlast	Output	
icpdtwakeup	Output	Registered wake signal to indicate that a message is arriving or is about to arrive on the icpd bus
icpdtkeep[(<gic_stream_width>/8)−1:0]	Output	Indicates the data bytes that must be transferred. This signal is only present on the Redistributor.
icpdtid[variable:0]	Output	Specifies the source Redistributor block. This signal is only present on the Distributor, so the system must provide an icpdtid signal.

Distributor to ITS AXI4-Stream interface, icdi bus		
Signal	**Direction**	**Description**
icditready	Input	AXI4-Stream compliant bus for communication from the Distributor to an ITS. The interface is fully credited and can be sent over any free-flowing interconnect. The signal width is set using the `its_stream_data_width` configuration option.
icditvalid	Output	
icditdata[<its_stream_data_width>−1:0]	Output	
icditlast	Output	
icditwakeup	Output	Indicates that a message is arriving or is about to arrive on the icdi bus
icditkeep[(<its_stream_data_width>/8)−1:0]	Output	Indicates the data bytes that must be transferred. This signal is only present on the Distributor.
icditdest[variable:0]	Output	Specifies the destination ITS block. This signal is only present on the Distributor.

ITS to Distributor AXI4-Stream interface, icid bus		
Signal	**Direction**	**Description**
icidtready	Input	AXI4-Stream compliant bus for communication from an ITS to the Distributor. The interface is fully credited and can be sent over any free-flowing interconnect. The signal width is set using the `stream_data_width` configuration option.
icidtvalid	Output	
icidtdata[<stream_data_width>−1:0]	Output	
icidtlast	Output	
icidtwakeup	Output	Registered wake signal. Indicates that a message is arriving or is about to arrive on the icid bus
icidtkeep[(<stream_data_width>/8)−1:0]	Output	Indicates the data bytes that must be transferred. This signal is only present on the ITS.
icidtid[variable:0]	Output	Specifies the source ITS. This signal is only present on the Distributor, so the system must provide an icidtid signal.

Distributor to Wake Request AXI4-Stream interface, icdw bus		
Signal	**Direction**	**Description**
icdwtready	Input	AXI4-Stream compliant bus for communication from the Distributor to the Wake Request block. The interface is fully credited and can be sent over any free-flowing interconnect. The icdw bus is not exposed when the top level is stitched.
icdwtvalid	Output	
icdwtdata[15:0]	Output	
icdwtwakeup	Output	Registered wake signal to indicate that a message is arriving or is about to arrive on the icdw bus. This signal is not exposed when the top level is stitched.

Distributor to SPI Collator AXI4-Stream interface, icdc bus		
Signal	**Direction**	**Description**
icdctready	Input	AXI4-Stream compliant bus for communication between the Distributor and the SPI Collator. The interface is fully credited and can be sent over any free-flowing interconnect.
icdctvalid	Output	
icdctdata[15:0]	Output	
icdctwakeup	Output	Registered wake signal to indicate that a message is arriving or is about to arrive on the icdc bus
icdctdest	Output	Indicates that the SPI Collator number is always 0. This signal is only present on the Distributor.

SPI Collator to Distributor AXI4-Stream interface, iccd bus		
Signal	**Direction**	**Description**
iccdtready	Input	AXI4-Stream compliant bus for communication between the SPI Collator and the Distributor. The interface is fully credited and can be sent over any free-flowing interconnect.
iccdtvalid	Output	
iccdtdata[15:0]	Output	
iccdtwakeup	Output	Registered wake signal to indicate that a message is arriving or is about to arrive on the iccd bus
iccdtid	Output	Indicates the SPI Collator number. This signal is only present on the Distributor, and must be set LOW.

A.8 Interdomain signals

Interdomain signals are routed between domains.

Signal definitions

Table A-9: Interdomain signals

Signal	Direction	Description
wakeup_sm_*	Input and output, depends on signal name	These signals connect between halves of a CoreLink™ ADB-400. See the *Arm® CoreLink™ ADB-400 AMBA® Domain Bridge User Guide*. 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.
wakeup_ms_*		
*_async_*		

A.9 Interchip AXI4-Stream interface signals

The following table shows the GIC-600AE interchip signals.

Signal definitions

Table A-10: Interchip signals

Distributor to remote chip AXI4-Stream interface, icdr bus		
Signal	**Direction**	**Description**
icdrtready	Input	AXI4-Stream compliant bus for communication between the Distributor and a remote chip. The interface is fully credited and can be sent over any free-flowing interconnect.
icdrtvalid	Output	
icdrtdata[63:0]	Output	
icdrtlast	Output	
icdrtwakeup	Output	Registered wake signal to indicate that a message is arriving or is about to arrive on the icdr bus. The icdrtvalid and icdrtready signals control data transfer.
icdrtdest[<chip_addr_width>−1:0]	Output	Specifies the destination remote chip. The signal width is set using the `chip_addr_width` configuration option. This signal is only present on the Distributor.
icdrtkeep	Output	Indicates the data bytes that must be transferred. This signal is only present on the Distributor.

Remote chip to Distributor AXI4-Stream interface, icrd bus		
Signal	**Direction**	**Description**
icrdtready	Input	AXI4-Stream compliant bus for communication between the remote chip and the Distributor. The interface is fully credited and can be sent over any free-flowing interconnect.
icrdtvalid	Output	
icrdtdata[63:0]	Output	
icrdtlast	Output	
icrdtwakeup	Output	Registered wake signal to indicate that a message is arriving or is about to arrive on the icrd bus. The icrdtvalid and icrdtready signals control data transfer.

Appendix B Implementation-defined features

The GIC-600AE 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-600AE. These features and options are configurable at build time.

The following table summarizes the implementation-defined features of the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4 that the GIC-600AE uses. The table also gives references to sections within this document that provide information about implementation-defined behavior of the GIC-600AE.

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 4.5 SPI routing and 1 of N selection on page 54
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 communication occurs 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 4.15 Reliability, Accessibility, and Serviceability on page 68.
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.

GICv3 architecture feature	Architectural specification reference		Description
	Chapter	Section	
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-600AE 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.
Changing priority	Physical interrupt handling and prioritization	Interrupt prioritization. Changing the priority of enabled PPIs, SGIs, and SPIs.	Reprogramming an IPRIORITYRn register does not change the priority of an active interrupt. Reprogramming an IPRIORITYRn 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 4.10 LPI caching on page 63 and 4.9 ITS on page 60.
LPI Configuration tables	Locality-specific peripheral interrupts and the ITS	LPI Configuration tables	The GIC-600AE 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 GICR_TYPER.CommonLPIAff for more information. 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	See the Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4

Appendix C Revisions

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

Table C-1: Issue 0000-00

Change	Location
First release	-

Table C-2: Differences between issue 0000-00 and issue 0000-01

Change	Location
Changed instances of qactive_clk_col to qactive_col_clk	3.5.4 SPI Collator clock Q-Channel on page 43
Updated bit assignments	• 5.10.2 FMU_ERR<n>CTLR, Error Record Control Register on page 180 • 5.10.3 FMU_ERR<n>STATUS, Error Record Primary Status register on page 182 • 5.10.6 FMU_PINGCTLR, Ping Control Register on page 185 • 5.10.7 FMU_PINGNOW, Ping Now register on page 186
Replaced redundant table with reference to the *Safety Manual*	6.1 Safety Mechanism overview on page 193
Removed MSI-64 Encapsulator Safety Mechanism	6.2.5 Safety Mechanism IDs on page 199
Added list of Safety Mechanisms that cannot be disabled through the FMU_SMEN register	6.2.5.1 Enabling or disabling a Safety Mechanism on page 202
Added software procedure to initiate a directed ping	6.2.6 Ping mechanisms on page 202
Added section	6.2.8 Correctable Error enable on page 206
Added *Prioritized FMU_ERR<n>STATUS registers* content	6.2.9 Software interaction on page 207
Added section	6.10.4 Disabling P-Channel and Q-Channel Safety Mechanisms on page 238
Added section	6.11 PPI and SPI interrupt interface protection on page 244
Added `FUSA_DISABLE_PQCHAN_PROT` parameter. Removed redundant information about FuSa parameters.	6.15 Configuration and parameters on page 248

Table C-3: Differences between issue 0000-01 and issue 0001-02

Change	Location
Changed instances of: • `DEVICEID_WIDTH` to `DID_WIDTH` • `num_cpus` to `cpus` • `num_ppis` to `ppi_count` • `num_its` to `its_count`	Throughout the document
Added information to note	3.3 Interrupt Translation Service on page 34
Updated the ACE_LITE_ACCESS_FAILURE description	4.15.6.7 ITS command and translation error records 13 and beyond on page 81

Change	Location
Corrected the values of the ProductID, Variant, and Revision fields	• 5.2.3 GICD_IIDR, Distributor Implementer Identification Register on page 103 • 5.4.1 GICR_IIDR, Redistributor Implementation Identification Register on page 122 • 5.6.1 GITS_IIDR, ITS Implementer Identification Register on page 138
Corrected the values of the Version field	5.5.5 GICR_CFGID1, Configuration ID1 Register on page 135
Writing `0b10` also clears the CE field	5.10.3 FMU_ERR<n>STATUS, Error Record Primary Status register on page 182
Added the following Safety Mechanisms: • GICD FMU ClkGate override • PPI FMU ClkGate override • ITS FMU ClkGate override	6.2.5 Safety Mechanism IDs on page 199
Added restriction for ClkGate override Safety Mechanisms	6.2.5.2 Injecting an error in a Safety Mechanism on page 202

Table C-4: Differences between issue 0001-02 and issue 0002-03

Change	Location
Removed reference to *Reliability Accessibility Serviceability* (RAS)	Throughout 5.10 FMU register summary on page 179 and including its subsections, and 6. Functional Safety on page 193
Updated the Revision field value	• 5.2.3 GICD_IIDR, Distributor Implementer Identification Register on page 103 • 5.4.1 GICR_IIDR, Redistributor Implementation Identification Register on page 122 • 5.6.1 GITS_IIDR, ITS Implementer Identification Register on page 138
Added registers	• 5.3.1 GICM_TYPER, Message-based Type Register on page 119 • 5.3.2 GICM_IIDR, Message-based Distributor Implementer Identification Register on page 120
Updated the Version field value	5.5.5 GICR_CFGID1, Configuration ID1 Register on page 135
Corrected the EVENT_ID description	Table 5-40: GITS_OPR bit descriptions on page 144
Corrected the FMU_ERR<n>STATUS reset value	5.10 FMU register summary on page 179
Updated the FI and UI bit descriptions	5.10.2 FMU_ERR<n>CTLR, Error Record Control Register on page 180
Updated the Usage constraints and Attributes. Updated the BLKID description.	5.10.3 FMU_ERR<n>STATUS, Error Record Primary Status register on page 182
Updated the enabled description	5.10.6 FMU_PINGCTLR, Ping Control Register on page 185
Corrected the ping_ack_received description	5.10.7 FMU_PINGNOW, Ping Now register on page 186
Added the remote_block_inject_error and gicd_inject_error bits	5.10.7 FMU_PINGNOW, Ping Now register on page 186
Updated the Usage constraints	5.10.10 FMU_PINGMASK, Ping Mask register on page 190

Change	Location
Updated the Usage constraints. Added a note about which combinations of BLK and SMID are not permitted.	5.10.8 FMU_SMEN, Safety Mechanism Enable register on page 187
Updated the Usage constraints. Added a note about which combinations of BLK and SMID are not permitted.	5.10.9 FMU_SMINJERR, Safety Mechanism Inject Error register on page 188
Updated the description of Safety Mechanism ID 0, for all blocks. Added extra content about handling an SMID:0 response.	6.2.5 Safety Mechanism IDs on page 199
Added the BLK and SMID field information. Added information about FMU_STATUS.idle.	6.2.5.2 Injecting an error in a Safety Mechanism on page 202
Added extra content	6.2.6 Ping mechanisms on page 202
Added extra information and an example of a 64-bit write access	6.2.7 Lock and key mechanism on page 205
Added new content	Power management on page 208

Table C-5: Differences between issue 0002-03 and issue 0003-04

Change	Location
Replaced the non-inclusive language for: • The type of ACE-Lite interface. The document now uses manager and subordinate interfaces. • The type of AXI4-Stream interface and GIC Stream interface. The document now uses transmitter and receiver interfaces.	Throughout the document
Corrected a signal name, that is, gits_transr_page_offset becomes its_transr_page_offset	• 3.1.2 Distributor ACE-Lite subordinate interface on page 27 • 4.12 MSI-64 on page 65
Corrected the maximum number of outstanding write acceptance capabilities from 128 to 256	3.3.1 ITS ACE-Lite subordinate interface on page 36
Added Figure 4-3: MSI-64 Encapsulator with DeviceID sent in the data[63:32] bits on page 66	4.12 MSI-64 on page 65
Added more information about programming the GICP_IRQCR.SPIID field	Overflow interrupt on page 68
Corrected the *corrupted SGI number* formula	Table 4-10: SGI RAM errors, records 3-4 on page 78
• Corrected the Type assignment of GICD_CHIPSR from RW to RO • Corrected the number of Reserved registers for GICD_NSACRn when affinity routing is enabled	5.2 Distributor registers (GICD/GICDA) summary on page 98
Corrected the description of the *_busy bits	5.2.6 GICD_CHIPSR, Chip Status Register on page 107
Corrected the GICR_ICFGR1 reset value	5.5 Redistributor registers for SGIs and PPIs summary on page 130
Added the Revision field value for r0p3	• 5.2.3 GICD_IIDR, Distributor Implementer Identification Register on page 103 • 5.4.1 GICR_IIDR, Redistributor Implementation Identification Register on page 122 • 5.6.1 GITS_IIDR, ITS Implementer Identification Register on page 138

Change	Location
Added a usage constraint about 32-bit writes	5.5.3 GICR_SGIDR, SGI Default Register on page 134
Added the Version field value for r0p3	5.5.5 GICR_CFGID1, Configuration ID1 Register on page 135
Corrected the GITS_BASER0 and GITS_BASER1 reset values	5.6 ITS control register summary on page 136
Corrected the DEVICE_ID and EVENT_ID descriptions	Table 5-40: GITS_OPR bit descriptions on page 144
Corrected the Count field description	Table 5-50: GICT_ERR<n>MISC0 bit descriptions on page 154
Renamed GICT_ERRIDR to GICT_DEVID. Corrected the NUM field values.	5.8.9 GICT_DEVID, Device Configuration register on page 161
Corrected the GICP_PMDEVARCH reset value and the GICP_CIDR1 reset value	5.9 GICP register summary on page 163
Corrected the fmu_fault_int signal naming	Figure 6-2: FMU interconnections on page 196
Corrected the information about pslverr	6.2.1 FMU APB4 interface on page 196
Added information about not updating FMU_PINGCTLR[31:1] when background ping is enabled	6.2.6 Ping mechanisms on page 202
Added the ppi_id_chk signal	6.4.1 Non-architected FuSa ports on page 209
Added “Interfaces that use partial duplication do not invert their *_chk signals.”	6.9.1 GIC-rendered partially duplicated interconnect on page 226
Corrected the description and block assignment for fault_icidi* (the first instance in the table)	Table 6-12: fault_* tie-off signals on page 232
Deleted the “Minimum of one cycle.” text from the [<domain>]reset_n description	A.1 Common control signals on page 250
Deleted the iccdtlast and icdctlast signals between the GICD and SPI Collators	A.7 Interblock AXI4-Stream interface signals on page 259
Deleted the “must not backpressure” text	• A.7 Interblock AXI4-Stream interface signals on page 259 • A.9 Interchip AXI4-Stream interface signals on page 262