PrimeCell® Generic Interrupt Controller (PL390)

Revision: r0p0

Technical Reference Manual

ARM DDI 0416B (ID012510)

PrimeCell Generic Interrupt Controller (PL390)
Technical Reference Manual

Release Information

The *Change history* table lists the changes made to this book.

Change history

Date	Issue	Confidentiality	Change
28 April 2008	A	Non-Confidential	First release for r0p0
23 November 2009	B	Non-Confidential	Second release for r0p0

Product Status

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

Web Address

`http://www.arm.com`

Contents

PrimeCell Generic Interrupt Controller (PL390) Technical Reference Manual

List of Tables
PrimeCell Generic Interrupt Controller (PL390) Technical Reference Manual

List of Figures
PrimeCell Generic Interrupt Controller (PL390) Technical Reference Manual

Preface

This preface introduces the *PrimeCell Generic Interrupt Controller (PL390) Technical Reference Manual* (TRM). It contains the following sections:

- *About this book* on page x
- *Feedback* on page xiii.

About this book

This is the TRM for the *PrimeCell Generic Interrupt Controller (PL390)*. The *Generic Interrupt Controller* (GIC) is a configurable interrupt controller that supports uniprocessor or multiprocessor systems.

Product revision status

The r*n*p*n* identifier indicates the revision status of the product described in this book, where:

r*n*	Identifies the major revision of the product.
p*n*	Identifies the minor revision or modification status of the product.

Intended audience

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

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

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

Chapter 2 *Functional Overview*

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

Chapter 3 *Programmers Model*

Read this for a description of the memory map and registers.

Chapter 4 *Programmers Model for Test*

Read this for a description of the test registers.

Appendix A *Signal Descriptions*

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

Appendix B *Interrupt Signaling*

Read this for a description of how the GIC signals interrupts to a processor.

Appendix C *Revisions*

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

Glossary Read this for definitions of terms used in this book.

Conventions

Conventions that this book can use are described in:

- *Typographical* on page xi
- *Timing diagrams* on page xi
- *Signals* on page xi.

Typographical

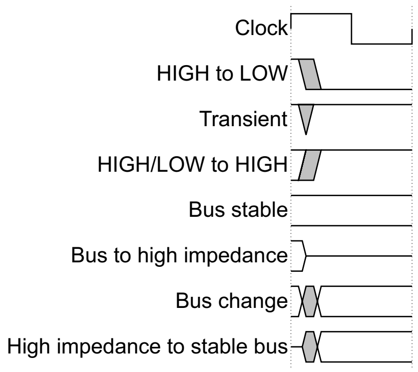
The typographical conventions are:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace`	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
< and >	Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`

Timing diagrams

The figure named *Key to timing diagram conventions* 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.



Key to timing diagram conventions

Signals

The signal conventions are:

Signal level	The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means: • HIGH for active-HIGH signals • LOW for active-LOW signals.
Lower-case n	At the start or end of a signal name denotes an active-LOW signal.

Additional reading

This section lists publications by ARM and by third parties.

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

ARM publications

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

- *PrimeCell Generic Interrupt Controller (PL390) Implementation Guide* (ARM DII 0178)
- *PrimeCell Generic Interrupt Controller (PL390) Integration Manual* (ARM DII 0179)
- *PrimeCell Generic Interrupt Controller (PL390) Supplement to AMBA® Designer (FD001) User Guide* (ARM DSU 0008)
- *AMBA Designer (FD001) User Guide* (ARM DUI 0333)
- *ARM Generic Interrupt Controller Architecture Specification* (ARM IHI 0048)
- *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition* (ARM DDI 0406)
- *AMBA AXI Protocol v1.0 Specification* (ARM IHI 0022)
- *AMBA 3 AHB-Lite Protocol v1.0 Specification* (ARM IHI 0033).

Other publications

This section lists relevant documents published by third parties:

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

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

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

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

Feedback on content

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

- the title
- the number, ARM DDI 0416B
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter introduces the GIC (PL390). It contains the following sections:

1.1 About the GIC

The GIC is an *Advanced Microcontroller Bus Architecture* (AMBA) and ARM Architecture compliant *System-on-Chip* (SoC) peripheral. It is a high-performance, area-optimized interrupt controller with on-chip AMBA bus interfaces that, depending on the configuration, conform to the AMBA *Advanced eXtensible Interface* (AXI) protocol or the AMBA AHB-Lite protocol.

The main product configurations are:

GIC with AHB-Lite slave interfaces

Implements the AHB-Lite protocol and contains a single CPU Interface.

GIC with AXI slave interfaces

Implements the AXI protocol, supports the Security Extensions, and contains up to eight CPU Interfaces.

You can configure the GIC to provide the optimum features, performance, and gate count required for your intended application. For a summary of the configurable features supported, see *Configurable options* on page 1-8.

The GIC implements the ARM Generic Interrupt Controller Architecture. See the *ARM Generic Interrupt Controller Architecture Specification* for information about the:

- architecture and interrupt types
- interrupt prioritization
- programmers model.

Figure 1-1 shows the interfaces that are available on the GIC.

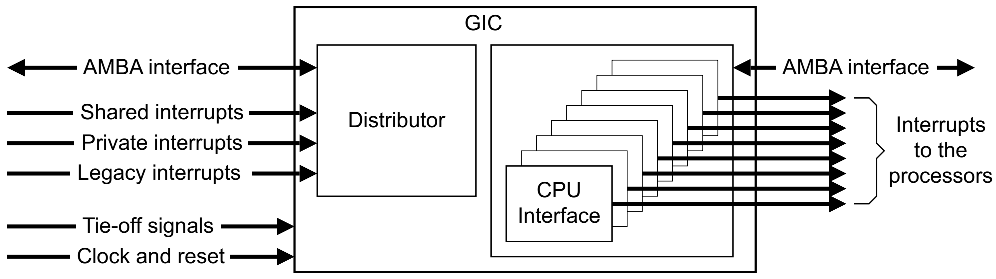


Figure 1-1 Interfaces on the GIC

——— **Note** ———

In Figure 1-1, the configuration of the GIC determines:

- the number of CPU Interfaces
- if both AMBA interfaces are either AXI or AHB-Lite.

———————————

Figure 1-2 on page 1-3 shows the GIC in an example multiprocessor system.

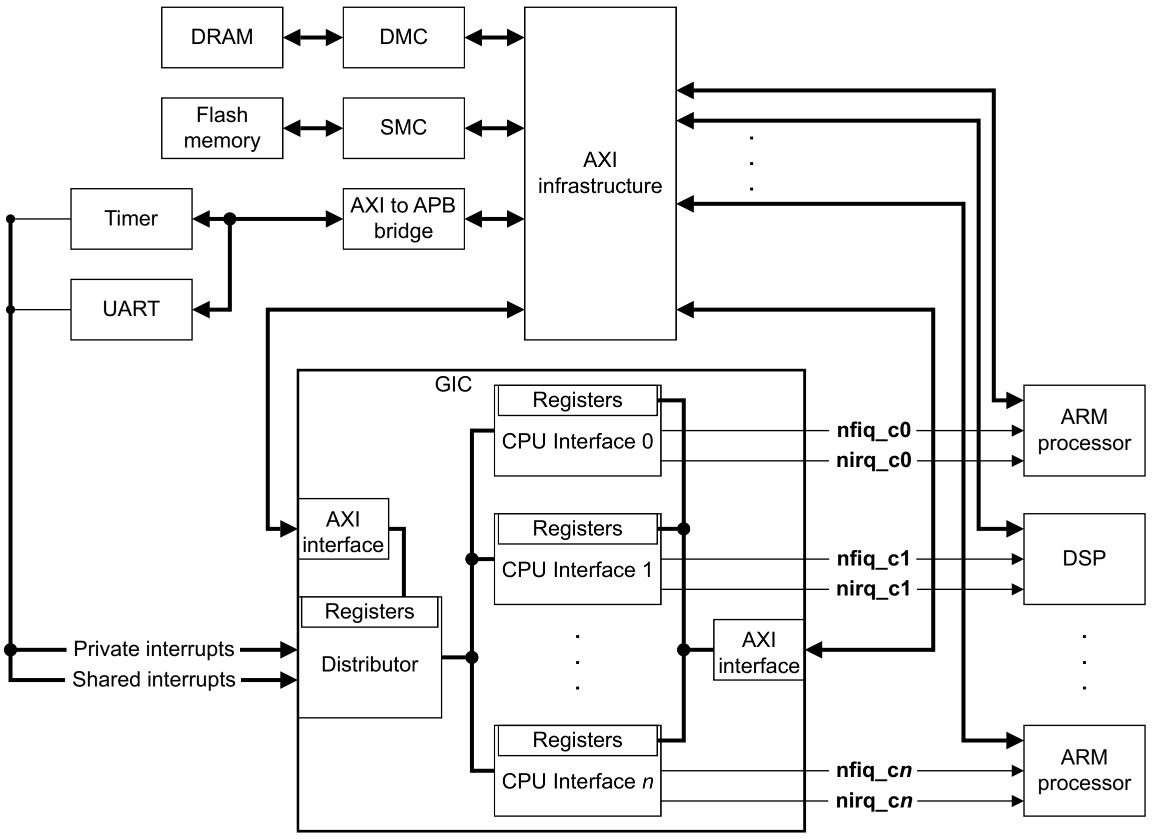


Figure 1-2 Example multiprocessor system

The GIC example multiprocessor system contains:

- A GIC configured to use the AXI protocol. The GIC provides a CPU Interface for each bus master that connects to it.
- AXI bus masters:
  - two ARM processors.
  - a *Digital Signal Processor* (DSP).
  - up to five additional bus masters. For clarity, these are not shown.
- AXI infrastructure component.
- PrimeCell slaves:
  - a *Dynamic Memory Controller* (DMC)
  - a *Static Memory Controller* (SMC)
  - a Timer
  - a *Universal Asynchronous Receiver-Transmitter* (UART).

Figure 1-3 on page 1-4 shows the GIC in an example uniprocessor system.

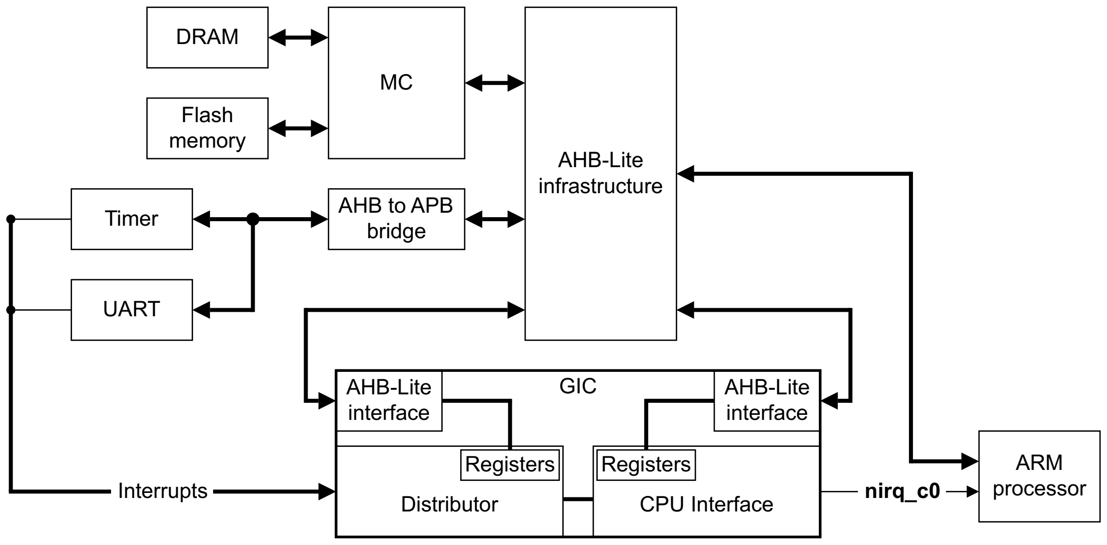


Figure 1-3 Example uniprocessor system

The GIC example uniprocessor system contains:

- a GIC configured to use the AHB-Lite protocol
- an ARM processor
- AHB infrastructure component
- PrimeCell slaves:
  - a *Memory Controller* (MC)
  - a Timer
  - a *Universal Asynchronous Receiver-Transmitter* (UART).

The AHB-Lite interconnect enables the processor to access the slaves. The Timer and UART connect to the AHB-Lite interconnect using an AHB to APB bridge component.

1.2 Compliance

The GIC is compliant with the following standards and protocols:

- AMBA 3 AXI protocol
- AMBA 3 AHB-Lite protocol
- ARM Generic Interrupt Controller Architecture.

1.3 Features

The GIC provides the following features:

- Support for three interrupt types:
  - *Software Generated Interrupt* (SGI)
  - *Private Peripheral Interrupt* (PPI)
  - *Shared Peripheral Interrupt* (SPI).
- Programmable interrupts that enable you to set the:
  - security state for an interrupt
  - priority level of an interrupt
  - enabling or disabling of an interrupt
  - processors that receive an interrupt.
- Enhanced security features, when the GIC is configured to support the Security Extensions.

1.4 Interfaces

The GIC provides separate AMBA slave interfaces that enable you to program the Distributor and the CPU Interfaces. Depending on the configuration, the GIC contains either:

- two AMBA AXI slave interfaces
- two AMBA 3 AHB-Lite slave interfaces.

1.5 Configurable options

During implementation of the GIC, the features that are configurable depend on the configuration as follows:

Uniprocessor configurations

- AMBA protocol, either AXI or AHB-Lite. When set to AXI then the GIC can be configured to support the Security Extensions.

Multiprocessor configurations

- The width of the AXI ID tag signals.
- The number of CPU Interfaces.
- The number of lockable SPIs.
- The number of PPIs for a CPU Interface.
- PPIs can be pulse-sensitive or level-sensitive.
- To synchronize a PPI to **gclk**.
- To register a PPI.
- Security Extensions support, which enables the GIC to operate using the Secure state and the Non-secure state.

All configurations

- The number of SGIs for each processor, from zero to 16.
- The number of SPIs, from one to 988.
- To synchronize an SPI to **gclk**.
- To register an SPI.
- The number of priority levels.
- The number of register slices in the highest pending interrupt logic.
- To include legacy interrupts for all CPU Interfaces in the GIC. Each legacy interrupt can be:
  - either pulse-sensitive or level-sensitive
  - synchronized to **gclk**
  - registered.

1.6 Test features

The GIC provides integration test logic, see Chapter 4 *Programmers Model for Test*.

1.7 Product documentation, design flow, and architecture

This section describes the GIC books, how they relate to the design flow, and the relevant architectural standards and protocols.

See *Additional reading* on page xii for more information about the books described in this section.

1.7.1 Documentation

The GIC documentation is as follows:

Technical Reference Manual

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

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

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

User Guide

The *User Guide* (UG) describes:

- the available build configuration options and related issues in selecting them
- how to use AMBA Designer to:
  - configure the GIC
  - generate the *Register Transfer Level* (RTL).

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

Implementation Guide

The *Implementation Guide* (IG) describes the:

- *Out-Of-Box* instructions
- synthesis constraints.

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

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

Integration Manual

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

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

1.7.2 Design flow

The GIC is delivered as synthesizable RTL. Before it can be used in a product, it must go through the following process:

1. Implementation. The implementer configures and synthesizes the RTL to produce a hard macrocell.
2. Integration. The integrator connects the implemented design into an SoC. This includes connecting it to a memory system and peripherals.
3. Programming. The system programmer develops the software required to control the GIC and tests the required application software.

Each stage of the process:

- can be performed by a different party
- can include options that affect the behavior and features at the next stage:

  Build configuration

  The implementer chooses the options that affect how the RTL source files are pre-processed. They usually include or exclude logic that can affect the area or maximum frequency of the resulting macrocell.

  Configuration inputs

  The integrator configures some features of the GIC by tying inputs to specific values. These configurations affect the start-up behavior prior to the software taking control. They can also limit the options available to the software. See *Miscellaneous signals* on page A-11.

  Software control

  The programmer updates the GIC by programming particular values into software-visible registers. This affects the behavior of the GIC.

1.7.3 ARM architecture and protocol information

The GIC complies with, or implements, the specifications described in:

- *ARM generic interrupt controller architecture*
- *Advanced Microcontroller Bus Architecture*.

ARM generic interrupt controller architecture

The GIC implements the ARM generic interrupt controller architecture. See the *ARM Generic Interrupt Controller Architecture Specification*.

Advanced Microcontroller Bus Architecture

The GIC complies with the:

- AMBA 3 AXI protocol, see the *AMBA AXI Protocol Specification*
- AMBA 3 AHB-Lite protocol, see the *AMBA 3 AHB-Lite Protocol v1.0 Specification*.

1.8 Product revisions

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

r0p0	First release.

Chapter 2
Functional Overview

This chapter describes the GIC operation. It contains the following sections:

2.1 Functional interfaces

Figure 2-1 shows a block diagram of the GIC.

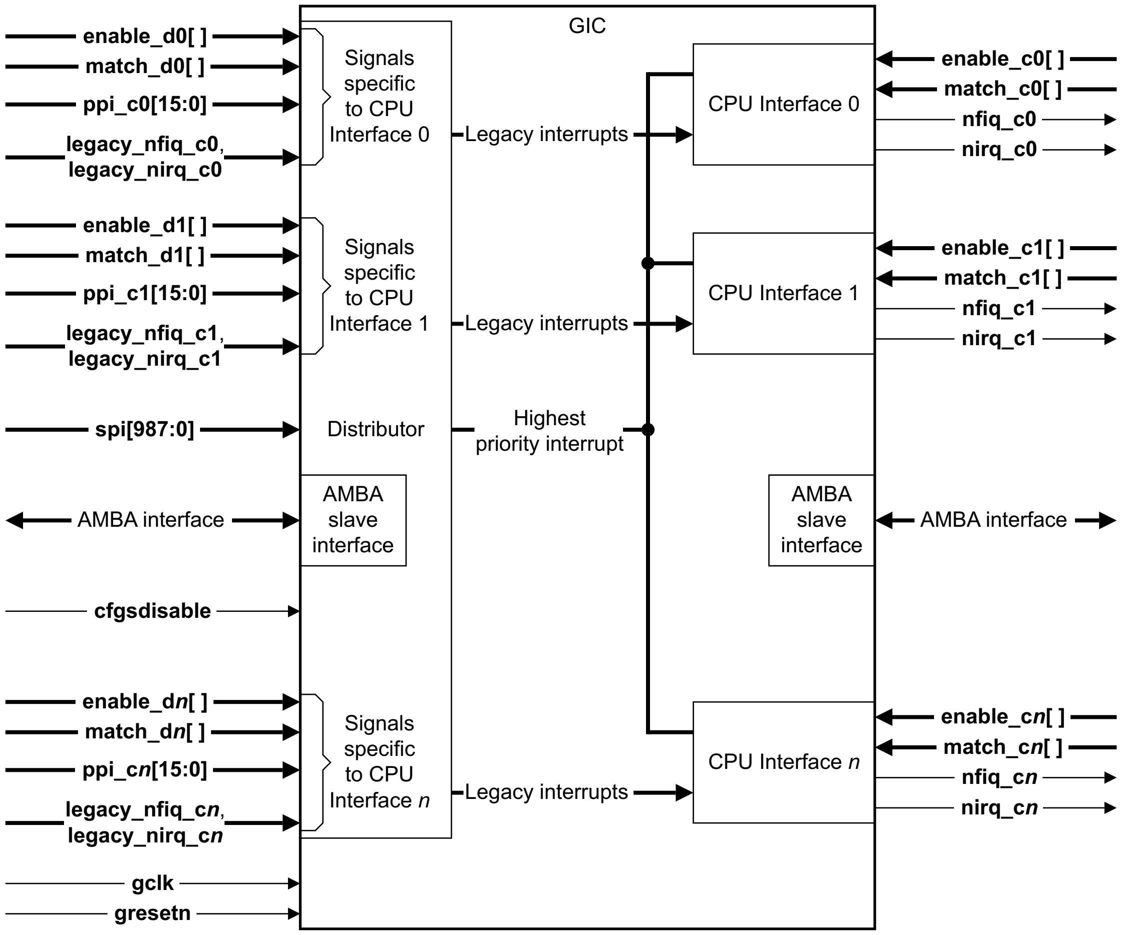


Figure 2-1 GIC block diagram

Note

Some configurations of the GIC might not include all of the signals that Figure 2-1 shows.

The main blocks of the GIC are:

- *AMBA slave interfaces*
- *Distributor* on page 2-5
- *CPU Interface* on page 2-6
- *Clock and reset* on page 2-7
- *enable and match signals* on page 2-7.

2.1.1 AMBA slave interfaces

The AMBA slave interfaces provide access to the GIC registers that enable you to program the system configuration parameters and obtain status information. See Chapter 3 *Programmers Model* for more information.

The GIC provides two AMBA slave interfaces, one for the Distributor and one that the CPU Interfaces share. You can configure the AMBA slave interfaces to be either AXI or AHB-Lite and these are described in:

- *AXI slave interface*
- *AHB-Lite slave interface* on page 2-4.

AXI slave interface

Both AXI slave interfaces use a 32-bit data bus and consist of the following AXI channels:

- Write-Address (AW)
- Write-Data (W)
- Write response (B)
- Read-Address (AR)
- Read-Data (R).

For information about the AXI protocol see the *AMBA AXI Protocol v1.0 Specification*.

Figure 2-2 shows the AXI slave interface external signals.

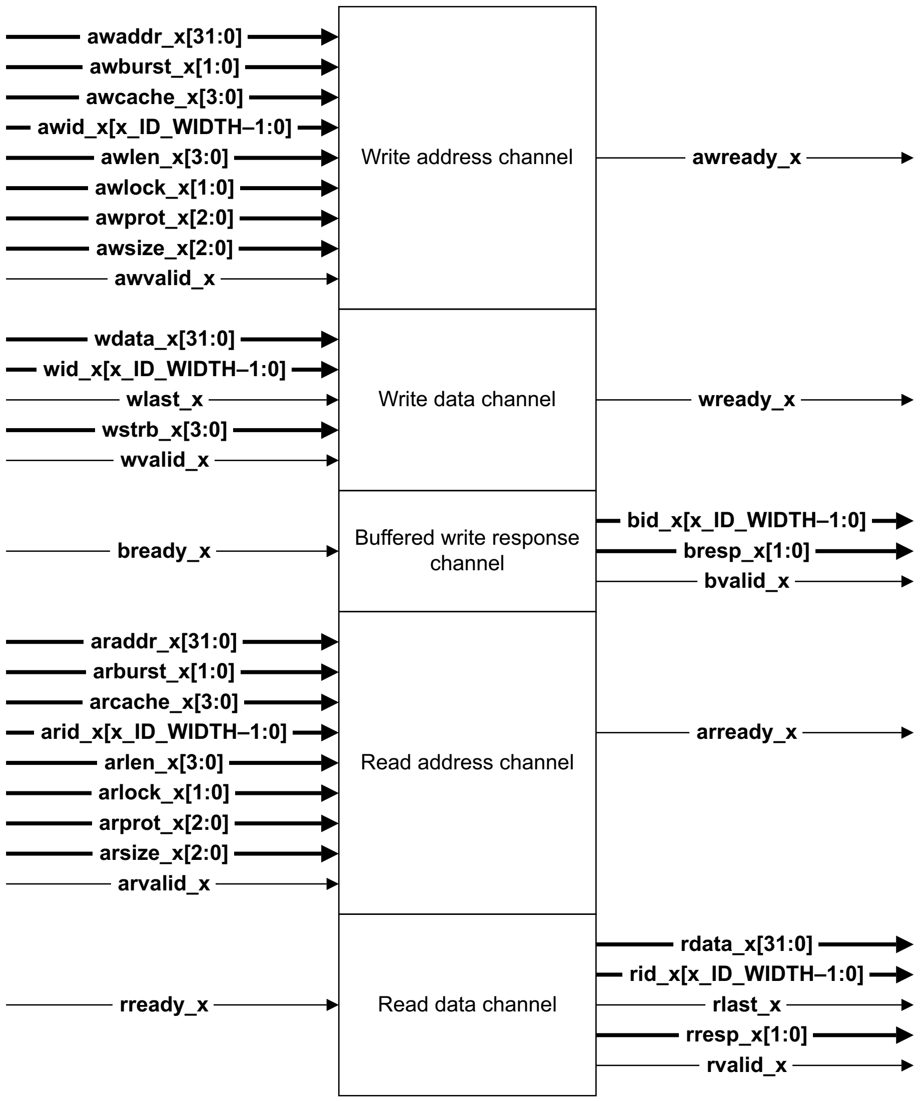


Figure 2-2 AXI slave interface

——— **Note** ———

In Figure 2-2 on page 2-3:

- Each signal uses an **_x** suffix to identify the AXI slave interface, where x is:

d	Distributor
c	CPU Interface(s).

- **x_ID_WIDTH** is the width of the ID tag, see *AXI slave interface signals* on page A-3.
- The clock and reset signals are not shown, see *Clock and reset signals* on page A-2.

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

Table 2-1 Attribute formats

Attribute[a]	Value
Combined acceptance capability	1
Write interleave depth	1
Read data reordering depth	1

a. See *Glossary* for a description of these AXI attributes.

AHB-Lite slave interface

When a GIC is configured to support the AHB-Lite protocol, then both AHB-Lite slave interfaces use a 32-bit data bus and are fully-compliant AHB-Lite slaves. For information about the AHB-Lite protocol see the *AMBA 3 AHB-Lite Protocol v1.0 Specification*.

Figure 2-3 shows the AHB-Lite external signals.

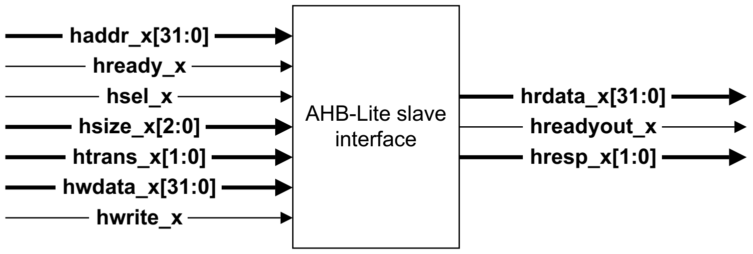


Figure 2-3 AHB-Lite slave interface

——— **Note** ———

In Figure 2-3:

- Each signal uses an **_x** suffix to identify the AHB-Lite interface, where x is:

d	Distributor
c	CPU Interface.

- The clock and reset signals are not shown, see *Clock and reset signals* on page A-2.

2.1.2 Distributor

The Distributor receives interrupts and provides the highest priority interrupt to the corresponding CPU Interface. An interrupt with a lower priority is forwarded to the appropriate CPU Interfaces when it becomes the highest priority pending interrupt.

Figure 2-4 shows the Distributor.

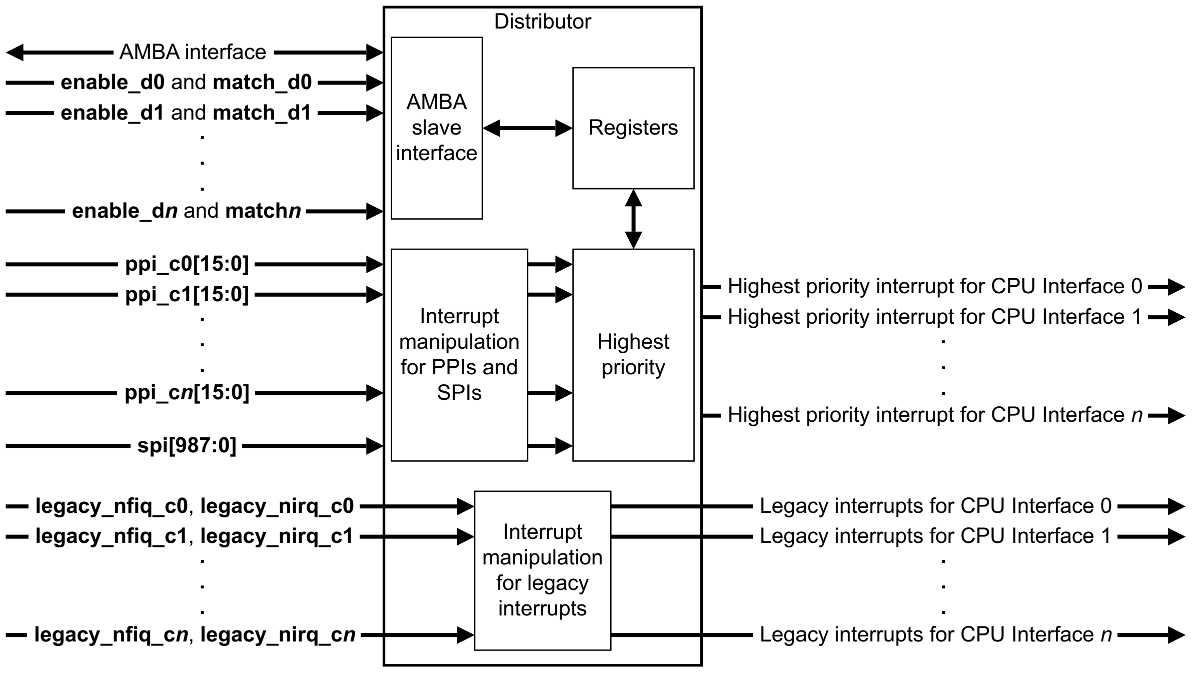

Distributor
AMBA interface
enable_d0 and match_d0
enable_d1 and match_d1
enable_dn and matchn
ppi_c0[15:0]
ppi_c1[15:0]
ppi_cn[15:0]
spi[987:0]
legacy_nfiq_c0, legacy_nirq_c0
legacy_nfiq_c1, legacy_nirq_c1
legacy_nfiq_cn, legacy_nirq_cn
AMBA slave interface
Registers
Interrupt manipulation for PPIs and SPIs
Highest priority
Interrupt manipulation for legacy interrupts
Highest priority interrupt for CPU Interface 0
Highest priority interrupt for CPU Interface 1
Highest priority interrupt for CPU Interface n
Legacy interrupts for CPU Interface 0
Legacy interrupts for CPU Interface 1
Legacy interrupts for CPU Interface n

Figure 2-4 Distributor

In multiprocessor configurations, the Distributor provides up to 16 *Private Peripheral Interrupts* (PPIs) for each CPU Interface. The Distributor only enables these interrupts to be forwarded to the corresponding CPU Interface.

The Distributor provides from 1 to 988 *Shared Peripheral Interrupts* (SPIs). For each SPI, you can program the Distributor to control how many CPU Interfaces it routes the interrupt to.

Interrupt manipulation block

The interrupt manipulation block enables the Distributor to manipulate an external interrupt as follows:

- the first stage provides the option of registering or synchronizing the interrupt
- the second stage provides an edge-detect option. This option is selected when the interrupt is set to be pulse-sensitive.

Figure 2-5 on page 2-6 shows the interrupt manipulation logic.

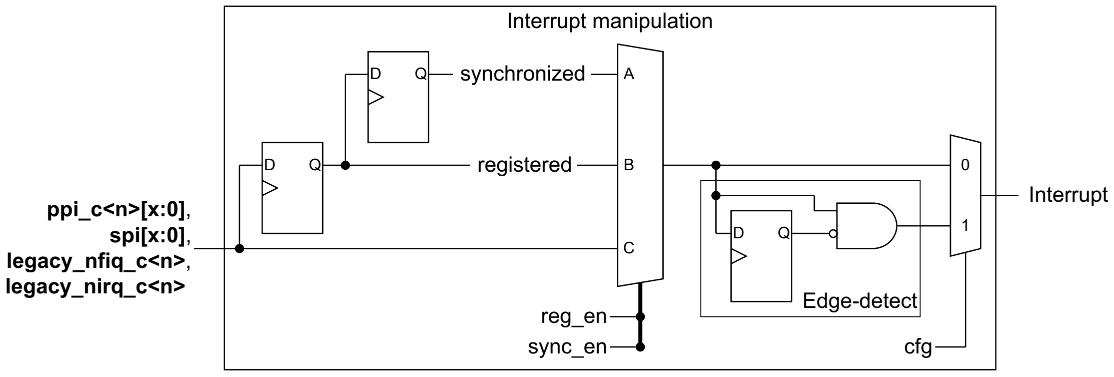


Figure 2-5 Interrupt manipulation logic

When you configure an external interrupt to be synchronized then the interrupt incurs an additional latency of two **gclk** cycles.

——— **Note** ———

The GIC enables you to configure the synchronization of any combination of PPIs, SPIs, and legacy interrupts.

———

When you configure a legacy interrupt or PPI, or program an SPI, to be pulse-sensitive then the GIC controls the 2-input multiplexor, in Figure 2-5, so that it selects the signal that passes through the edge-detect logic. The Distributor recognizes a pulse when the input is observed LOW and then HIGH on two consecutive rising edges of **gclk**. A pulse interrupt must be asserted for at least one **gclk** cycle to enable the Distributor to observe it.

——— **Note** ———

When configuring the legacy interrupts for CPU Interface *n*, you must set them to be level-sensitive if CPU Interface *n* connects to an ARM processor.

———

2.1.3 CPU Interface

A CPU Interface contains a programmable interrupt priority mask and it only accepts Pending interrupts if the priority of the interrupt is higher than the:

- programmed interrupt priority mask
- interrupts that the processor is currently servicing.

Figure 2-6 shows a CPU Interface.

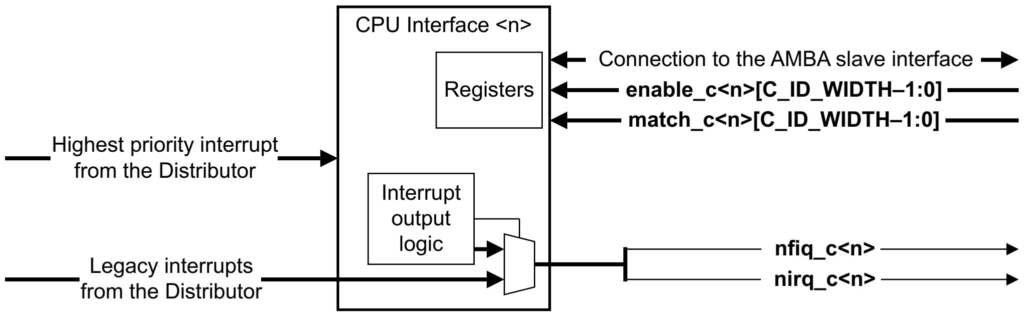

CPU Interface <n>
Registers
Connection to the AMBA slave interface
enable_c<n>[C_ID_WIDTH–1:0]
match_c<n>[C_ID_WIDTH–1:0]
Highest priority interrupt from the Distributor
Interrupt output logic
Legacy interrupts from the Distributor
nfiq_c<n>
nirq_c<n>

Figure 2-6 CPU Interface

2.1.4 Clock and reset

This section describes:

- *Clock*
- *Reset*.

Clock

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

Reset

The GIC provides a single reset input, **gresetn**. See *Clock and reset signals* on page A-2.

2.1.5 enable and match signals

When the GIC contains two or more CPU Interfaces then it provides the following signals:

- **enable_d<n>[D_ID_WIDTH–1:0]**
- **match_d<n>[D_ID_WIDTH–1:0]**
- **enable_c<n>[C_ID_WIDTH–1:0]**
- **match_c<n>[C_ID_WIDTH–1:0]**.

Where:

<n>	Is a number, from 0 to 7, that identifies a CPU Interface
D_ID_WIDTH	Is the width of the AXI ID signals on the Distributor interface.
C_ID_WIDTH	Is the width of the AXI ID signals on the AMBA interface that enables access to the CPU Interfaces.

The **enable** signals function as a mask select on the AXI ID signals and the result is compared with the **match** signals. If the comparison is true then the GIC provides access to the registers for the relevant CPU Interface, otherwise the GIC ignores writes and reads return zero.

For the Distributor, it provides access to the banked registers for a CPU Interface when:

- (**arid_d** & **enable_d<n>**) == **match_d<n>**
- (**awid_d** & **enable_d<n>**) == **match_d<n>**

——— **Note** ———

When an access to a non-banked register in the Distributor occurs then it provides access to the register irrespective of the state of the **enable_d** and **match_d** signals.

———————————

The GIC provides access to a CPU Interface and its registers when:

- (**arid_c** & **enable_c<n>**) == **match_c<n>**
- (**awid_c** & **enable_c<n>**) == **match_c<n>**

——— **Note** ———

- By changing the state of the **match** and **enable** signals, a processor in Secure state can then access any of the banked registers for all other CPU Interfaces in the GIC. Similarly, a processor in Non-secure state can also modify the **match** and **enable** signals but the GIC only provides access to the banked registers, for all other CPU Interfaces, that are accessible in Non-secure state.

- You must ensure that only one set of **enable_d<n>** and **match_d<n>** signals are valid for an active AXI ID tag to the Distributor. If the value of the AXI ID tag enables the Distributor to select multiple banked registers then the behavior is Unpredictable.
- You must ensure that only one set of **enable_c<n>** and **match_c<n>** signals are valid for an active AXI ID tag to the AXI slave interface that accesses the CPU Interfaces. If the value of the AXI ID tag enables the GIC to select multiple banked registers then the behavior is Unpredictable.

2.2 Implementation-defined behavior

This section describes the behavior of the GIC which the *ARM Generic Interrupt Controller Architecture Specification* defines as being Implementation-defined. It contains the following sections:

- *Number of CPU Interfaces*
- *Number of interrupt inputs*
- *Legacy interrupts*
- *Programmable trigger-mode of PPIs* on page 2-10
- *Lockable SPI (LSPI) support* on page 2-10
- *Number of priority levels* on page 2-10
- *Effect of updating the Priority Level Register for Active interrupts* on page 2-10
- *Interrupt prioritization of interrupts with equal priority* on page 2-10
- *Arbitration of SPIs which target multiple processors* on page 2-11
- *Initial value of the Interrupt Security Register* on page 2-11
- *Access restrictions of registers that are specific to a CPU Interface* on page 2-11
- *Minimum supported value in the Binary Point Register* on page 2-11.

2.2.1 Number of CPU Interfaces

The number of CPU Interfaces is configurable from one to eight inclusive. The AMBA Designer documentation provides information about configuring the GIC, see *Additional reading* on page xii.

2.2.2 Number of interrupt inputs

The GIC provides the following types of interrupt:

- *Software Generated Interrupt (SGI)*
- *Private Peripheral Interrupt (PPI)*
- *Shared Peripheral Interrupt (SPI)*.

Software Generated Interrupt (SGI)

You can configure the GIC to provide up to 16 SGIs for each CPU Interface. Each interrupt is assigned an interrupt ID number, INTID[15:0].

Private Peripheral Interrupt (PPI)

When you configure the GIC to contain more than one CPU Interface then the Distributor can provide from zero to 16 PPIs for each CPU Interface. The Distributor ensures that the interrupts it receives on the **ppi_c<n>[15:0]** signals are only accessible to the relevant CPU Interface, that is, CPU Interface *n*.

Shared Peripheral Interrupt (SPI)

You can configure the GIC to support up to 988 SPIs. The Distributor assigns each SPI an interrupt ID number, INTID[1019:32], and it distributes interrupts it receives on the **spi[x:0]** signals to any CPU Interface that the programming of the Target Register specifies.

2.2.3 Legacy interrupts

Depending on the GIC configuration, the Distributor can provide up to two legacy interrupt inputs, for each processor that connects to the GIC.

Note

- When the GIC is configured to support the Secure state only then you can configure the GIC to provide a **legacy_nirq_c<n>** for each CPU Interface that the GIC contains.
- When the GIC is configured to support the Security Extensions then you can configure the GIC to provide a **legacy_nfiq_c<n>** and **legacy_nirq_c<n>** for each CPU Interface that the GIC contains.

See the *ARM Generic Interrupt Controller Architecture Specification* for information about how to program the CPU Interface Control Register to route the legacy interrupt inputs to the interrupt outputs of a CPU Interface, that is, **nfiq_c<n>** and **nirq_c<n>**.

2.2.4 Programmable trigger-mode of PPIs

The GIC does not permit use of the Interrupt Configuration Register to program the trigger mode of PPIs.

Note

- The interrupt configuration of a PPI is set during configuration and can be either:
  - level-sensitive active HIGH
  - pulse-sensitive.
- The AMBA Designer documentation provides information about configuring the triggering mechanism of PPIs, see *Additional reading* on page xii.

2.2.5 *Lockable SPI* (LSPI) support

The GIC supports a configurable number of LSPIs. The AMBA Designer documentation provides information about configuring the number of LSPIs you require, see *Additional reading* on page xii.

Note

The LSPI field in the Interrupt Controller Type Register defines which INTIDs are LSPIs. See *Interrupt Controller Type Register (ICDICTR)* on page 3-6.

2.2.6 Number of priority levels

The number of priority levels that the GIC supports is configurable. The AMBA Designer documentation provides information about configuring the GIC, see *Additional reading* on page xii.

2.2.7 Effect of updating the Priority Level Register for Active interrupts

When you change the priority level of an interrupt that is either Active or Active-and-pending then the GIC immediately uses the new priority level for the Active interrupt.

2.2.8 Interrupt prioritization of interrupts with equal priority

If two or more interrupts, with the same priority level, are the highest Pending interrupts then depending on the interrupt type the Distributor arbitrates as follows:

PPI, SPI	The Distributor issues the interrupt with the lowest INTID.

SGI The Distributor issues the SGI with the lowest INTID. In multiprocessor systems, if a priority level conflict remains, the Distributor issues the SGI that was requested by the processor with the lowest CPUID. Therefore when a priority level conflict occurs, an SGI request from processor 0 has the highest priority and an SGI request from processor 7 has the lowest priority.

——— **Note** ———

The Distributor excludes INTIDs from entering the prioritization logic if they are in the Active-and-pending state.

2.2.9 Arbitration of SPIs which target multiple processors

The GIC performs no arbitration when an SPI signals an interrupt to multiple processors. Under these circumstances, all the targeted processors are signaled the SPI at the same time and it is the first processor to respond by reading their Interrupt Acknowledge Register that receives the INTID of the SPI. The GIC ensures that all of the remaining processors that were targeted receive a spurious interrupt response.

2.2.10 Initial value of the Interrupt Security Register

The GIC resets all bits in the Interrupt Security Register to zero.

2.2.11 Access restrictions of registers that are specific to a CPU Interface

When a GIC contains multiple CPU Interfaces then the GIC permits a processor in:

- Secure state to access all the registers for any other CPU Interface
- Non-secure state to access registers for any other CPU Interface, but only those registers that permits non-secure accesses.

See *enable and match signals* on page 2-7 for more information.

2.2.12 Minimum supported value in the Binary Point Register

The minimum value in the Binary Point Register depends on the number of priority levels that the GIC is configured to support.

Chapter 3
Programmers Model

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

3.1 About the programmers model

The GIC provides the following register maps:

- *Distributor register map*
- *CPU Interface register map* on page 3-4.

The following information applies to the GIC registers:

- The base address of the GIC is not fixed, and can be different for any particular system implementation. The offset of each register from the base address is fixed.
- Do not attempt to access reserved or unused address locations. Attempting to access these location can result in Unpredictable behavior of the GIC.
- Unless otherwise stated in the accompanying text:
  - do not modify undefined register bits
  - ignore undefined register bits on reads
  - all register bits are reset to a logic 0 by a system or power-on reset.
- Accesses can be byte, halfword, or word.
- The GIC only supports data in little-endian format.
- The Type column in Table 3-1 on page 3-5 and Table 3-18 on page 3-30 describes the access types as follows:

RW	Read and write.
RO	Read only.
WO	Write only.

3.1.1 Distributor register map

The register map of the Distributor spans a 4KB region, as Figure 3-1 on page 3-3 shows.

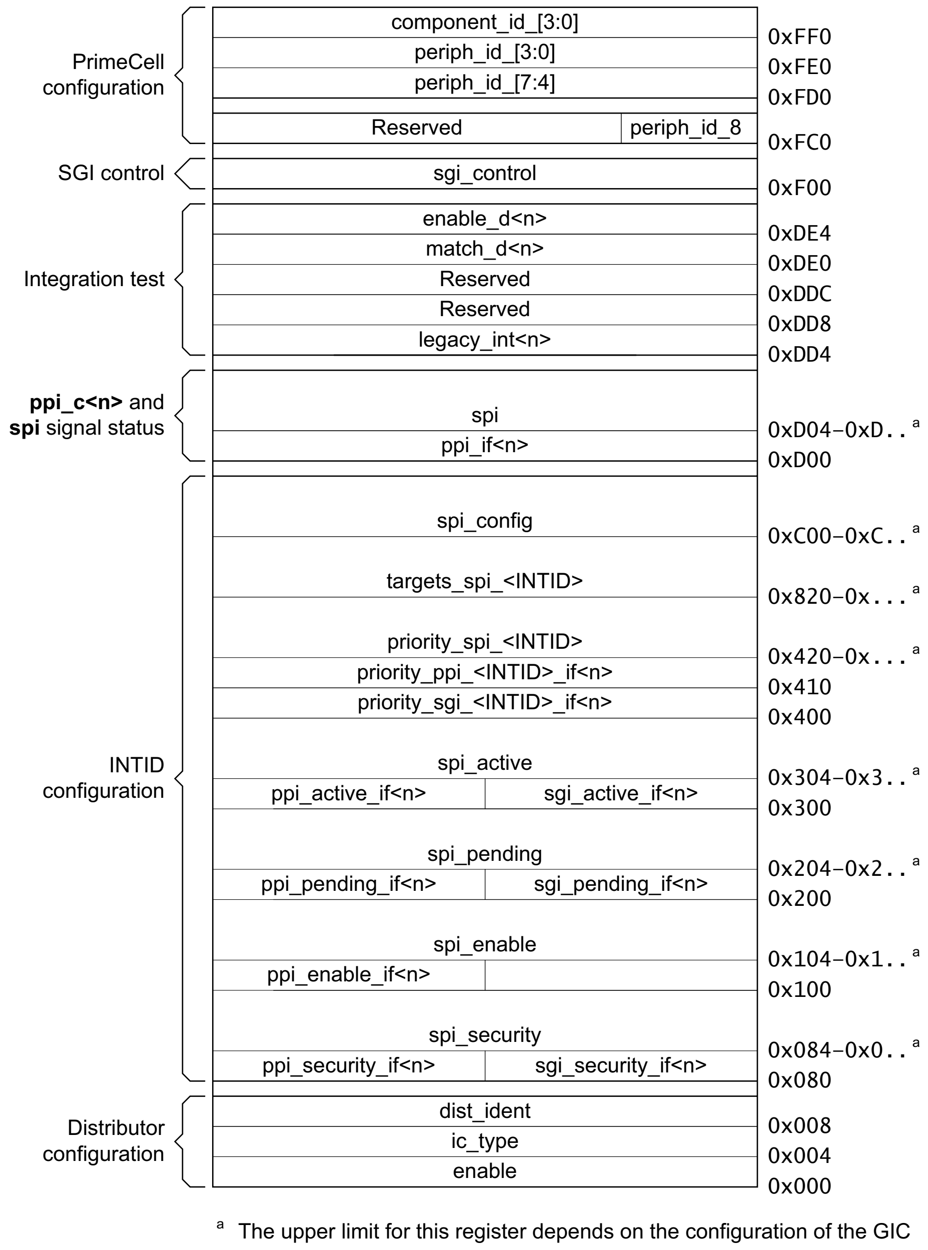


Figure 3-1 Distributor register map

In Figure 3-1, the register map consists of the following regions:

Distributor configuration

Use these registers to determine the global configuration of the Distributor and control its operating state.

INTID configuration

These registers provide the operating parameters for each INTID.

ppi_c<n> and spi status

These registers return the present logic status of the **ppi_c<n>** and **spi** inputs.

Integration test

Use these registers when testing the integration of the GIC in a *System-on-Chip* (SoC). See *Distributor integration test registers* on page 4-3 for more information.

SGI control Use this register to generate an SGI.

PrimeCell configuration

These registers enable the identification of system components by software.

See *Distributor register descriptions* on page 3-5 for information about the registers.

3.1.2 CPU Interface register map

The register map for each CPU Interface spans a 4KB region, as Figure 3-2 shows.

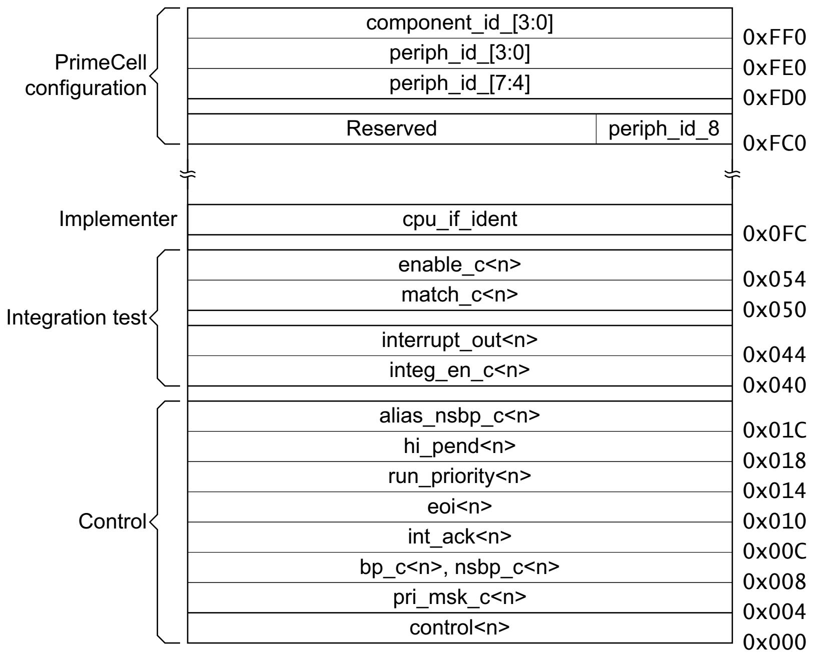


Figure 3-2 CPU Interface register map

In Figure 3-2, the register map consists of the following regions:

Control Use these registers to control the operating state of the CPU Interface.

Integration test

Use these registers to test the integration of the GIC in an SoC. See *CPU Interface integration test registers* on page 4-7 for more information.

Implementer

Identifies the implementer, and revision, of the CPU Interface.

PrimeCell configuration

These registers enable the identification of system components by software.

See *CPU Interface register descriptions* on page 3-30 for information about the registers.

3.2 Distributor register descriptions

This section describes the registers that the Distributor provides. Table 3-1 lists the Distributor registers and provides a reference to the register description, which is either in this book or the *ARM Generic Interrupt Controller Architecture Specification*.

Table 3-1 Distributor register summary

Offset	Name	Type	Reset	Width	Description
`0x000`	enable	RW	`0x00000000`	32	Distributor Control Register (ICDDCR)[a]
`0x004`	ic_type	RO	-[b]	32	*Interrupt Controller Type Register (ICDICTR)* on page 3-6
`0x008`	dist_ident	RO	`0x00---43B`[c]	32	*Distributor Implementer Identification Register (ICDIIDR)* on page 3-8
`0x00C-0x07C`	-	-	-	-	Reserved

`0x080`
`0x082`
`0x084-0x0FC`
	sgi_security_if<n>[d]
	ppi_security_if<n>[d]
	spi_security
		RW	`0x0000`
			`0x0000`
			`0x00000000`
				16
				16
				32
					Interrupt Security Registers (ICDISRn) on page 3-8

`0x100`
`0x102`
`0x104-0x17C`
	-
	ppi_enable_if<n>[d]
	spi_enable
		RO
		RW
		RW
			-[b]
			-[b]
			`0x00000000`
				16
				16
				32
					Enable Set Registers (ICDISERn) on page 3-9

`0x180`
`0x182`
`0x184-0x1FC`
	-
	ppi_enable_if<n>[d]
	spi_enable
		RO
		RW
		RW
			-[b]
			-[b]
			`0x00000000`
				16
				16
				32
					Enable Clear Registers (ICDICERn) on page 3-11

`0x200`
`0x202`
`0x204-0x27C`
	sgi_pending_if<n>[d]
	ppi_pending_if<n>[d]
	spi_pending
		RO
		RW
		RW
			`0x0000`
			`0x0000`
			`0x00000000`
				16
				16
				32
					Pending Set Registers (ICDISPRn) on page 3-12

`0x280`
`0x282`
`0x284-0x2FC`
	sgi_pending_if<n>[d]
	ppi_pending_if<n>[d]
	spi_pending
		RO
		RW
		RW
			`0x0000`
			`0x0000`
			`0x00000000`
				16
				16
				32
					Pending Clear Registers (ICDICPRn) on page 3-13

`0x300`
`0x302`
`0x304-0x37C`
	sgi_active_if<n>[d]
	ppi_active_if<n>[d]
	spi_active
		RO	`0x0000`
			`0x0000`
			`0x00000000`
				16
				16
				32
					Active Status Registers (ICDABRn) on page 3-15

`0x380-0x3FC`	-	-	-	-	Reserved

`0x400-0x40F`
`0x410-0x41F`
`0x420-0x7FB`
	priority_sgi_<INTID>_if<n>[d]
	priority_ppi_<INTID>_if<n>[d]
	priority_spi_<INTID>
		RW	`0x00`	8	*Priority Level Registers (ICDIPRn)* on page 3-16

`0x7FC`	-	-	-	32	Reserved

`0x800-0x81F`
`0x820-0xBFB`
	-
	targets_spi_<INTID>
		RO
		RW
			-[b]
			`0x00`
				8	*Target Registers (ICDIPTRn)* on page 3-18

`0xBFC`	-	-	-	32	Reserved

`0xC00-0xC04`
`0xC08-0xCFC`
	-
	spi_config
		RO
		RW
			-[b]	32	*Interrupt Configuration Registers (ICDICRn)* on page 3-19

Table 3-1 Distributor register summary (continued)

Offset	Name	Type	Reset	Width	Description
`0xD00`	ppi_if<n>[d]	RO[e]	`0x00000000`	32	*PPI Status Register* on page 3-20
`0xD04-0xD7C`	spi	RO[e]	`0x00000000`	32	*SPI Status Registers* on page 3-21
`0xD80-0xDD0`	-	-	-	-	Reserved

`0xDD4`
`0xDD8-0xDDC`
`0xDE0`
`0xDE4`
	legacy_int<n>[d]
	-
	match_d<n>[d]
	enable_d<n>[d]
		See Chapter 4 *Programmers Model for Test* for information about these registers.			

`0xDE8-0xEFC`	-	-	-	-	Reserved
`0xF00`	sgi_control	WO	-	32	Software Generated Interrupt Register (ICDSGIR)[a]
`0xF04-0xFBC`	-	-	-	-	Reserved
`0xFC0`	periph_id_8	RO	-[b]	8	*Peripheral Identification Registers* on page 3-22
`0xFC4-0xFCC`	-	-	-	-	Reserved

`0xFD0-0xFDC`
`0xFE0-0xFEC`
	periph_id_[7:4]
	periph_id_[3:0]
		RO	-[b]	8	*Peripheral Identification Registers* on page 3-22

`0xFF0-0xFFC`	component_id_[3:0]	RO	`0xB105F00D`	8	*PrimeCell Identification Registers* on page 3-29

a. See the *ARM Generic Interrupt Controller Architecture Specification*.
b. The reset value depends on the configuration of the GIC.
c. The reset value depends on the revision of the GIC. See *Distributor Implementer Identification Register (ICDIIDR)* on page 3-8.
d. <n> corresponds to the number of a CPU Interface. If the GIC contains two or more CPU Interfaces then the **enable_d<n>** and **match_d<n>** signals control which banked register is selected.
e. Only processors in Secure state can access this Implementation-defined register.

3.2.1 Interrupt Controller Type Register (ICDICTR)

The ic_type Register characteristics are:

Purpose	Provides information about the configuration of the GIC.
Usage constraints	No usage constraints.
Configurations	This register is available in all configurations of the GIC.
Attributes	See the register summary in Table 3-1 on page 3-5.

Figure 3-3 shows the ic_type Register bit assignments.

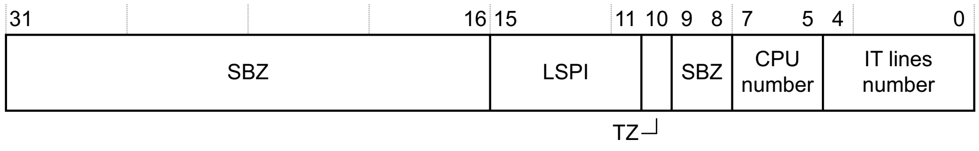


Figure 3-3 ic_type Register bit assignments

Table 3-2 shows the ic_type Register bit assignments.

Table 3-2 ic_type Register bit assignments

Bits	Name	Function
[31:16]	-	SBZ.
[15:11]	LSPI	Returns the number of *Lockable Shared Peripheral Interrupts* (LSPIs) that the GIC contains: b00000 = no LSPIs b00001 = 1 LSPI. INTID32 b00010 = 2 LSPIs, INTID32 - INTID33 b00011 = 3 LSPIs, INTID32 - INTID34 . . . b11110 = 30 LSPIs, INTID32 - INTID61 b11111 = 31 LSPIs, INTID32 - INTID62. When **cfgsdisable** is HIGH then the GIC prevents writes to any register locations that control the operating state of an LSPI that is programmed to be secure. See the *ARM Generic Interrupt Controller Architecture Specification* for more information about **cfgsdisable**. ——— **Note** ——— The AMBA Designer documentation provides information about configuring the number of LSPIs you require, see *Additional reading* on page xii. ————————
[10]	TZ	Returns the number of security states that the GIC supports: 0 = the GIC supports the Secure state 1 = the GIC supports a Secure state and a Non-secure state.
[9:8]	-	SBZ.

[7:5]	CPU number	Returns the number of CPU Interfaces that the GIC provides. The GIC provides either:
		b000 = one CPU Interface
		b001 = two CPU Interfaces
		b010 = three CPU Interfaces
		b011 = four CPU Interfaces
		b100 = five CPU Interfaces
		b101 = six CPU Interfaces
		b110 = seven CPU Interfaces
		b111 = eight CPU Interfaces.

[4:0]	IT lines number	Returns the number of INTIDs, to the nearest 32, that the Distributor provides: b00000 = the Distributor provides 1-32 INTIDs[a] b00001 = the Distributor provides 33-64 INTIDs b00010 = the Distributor provides 65-96 INTIDs b00011 = the Distributor provides 97-128 INTIDs . . . b11110 = the Distributor provides 961-992 INTIDs b11111 = the Distributor provides 993-1020 INTIDs. ——— **Note** ——— Software can use the *Enable Set Registers (ICDISERn)* on page 3-9 to discover exactly how many INTIDs the GIC contains. ————————

a. The Distributor always uses INTID0 to INTID31, to control any SGIs and PPIs that a configured GIC contains.

3.2.2 Distributor Implementer Identification Register (ICDIIDR)

The dist_ident Register characteristics are:

Purpose	Provides information about the implementer of the Distributor and the revision of the GIC.
Usage constraints	No usage constraints.
Configurations	This register is available in all configurations of the GIC.
Attributes	See the register summary in Table 3-1 on page 3-5.

Figure 3-4 shows the dist_ident Register bit assignments.

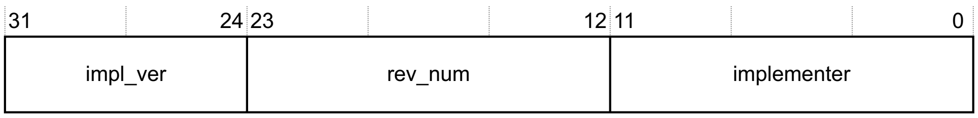


Figure 3-4 dist_ident Register bit assignments

Table 3-3 shows the dist_ident Register bit assignments.

Table 3-3 dist_ident Register bit assignments

Bits	Name	Function
[31:24]	impl_ver	Returns the product identifier. For a GIC (PL390) this field returns `0x00`.
[23:12]	rev_num	Returns the revision number of the GIC. For revision r0p0, this field returns `0x000`.
[11:0]	implementer	Returns the JEP106 code of the company that implemented the Distributor RTL, that is, ARM. It uses the following bit format: • [11:8] = `0x4`, that is, the JEP106 continuation code for ARM • [7] = 0 • [6:0] = b0111011, that is, the JEP106 code [6:0] for ARM. See the *JEDEC Standard Manufacturer's Identification Code* for information about JEP106.

3.2.3 Interrupt Security Registers (ICDISRn)

Figure 3-5 on page 3-9 shows the address map that the Distributor provides for the following security registers:

- sgi_security_if<n> Register
- ppi_security_if<n> Register
- spi_security Registers.

——— **Note** ———

The *ARM Generic Interrupt Controller Architecture Specification* describes the behavior of the Interrupt Security Registers (ICDISRn).

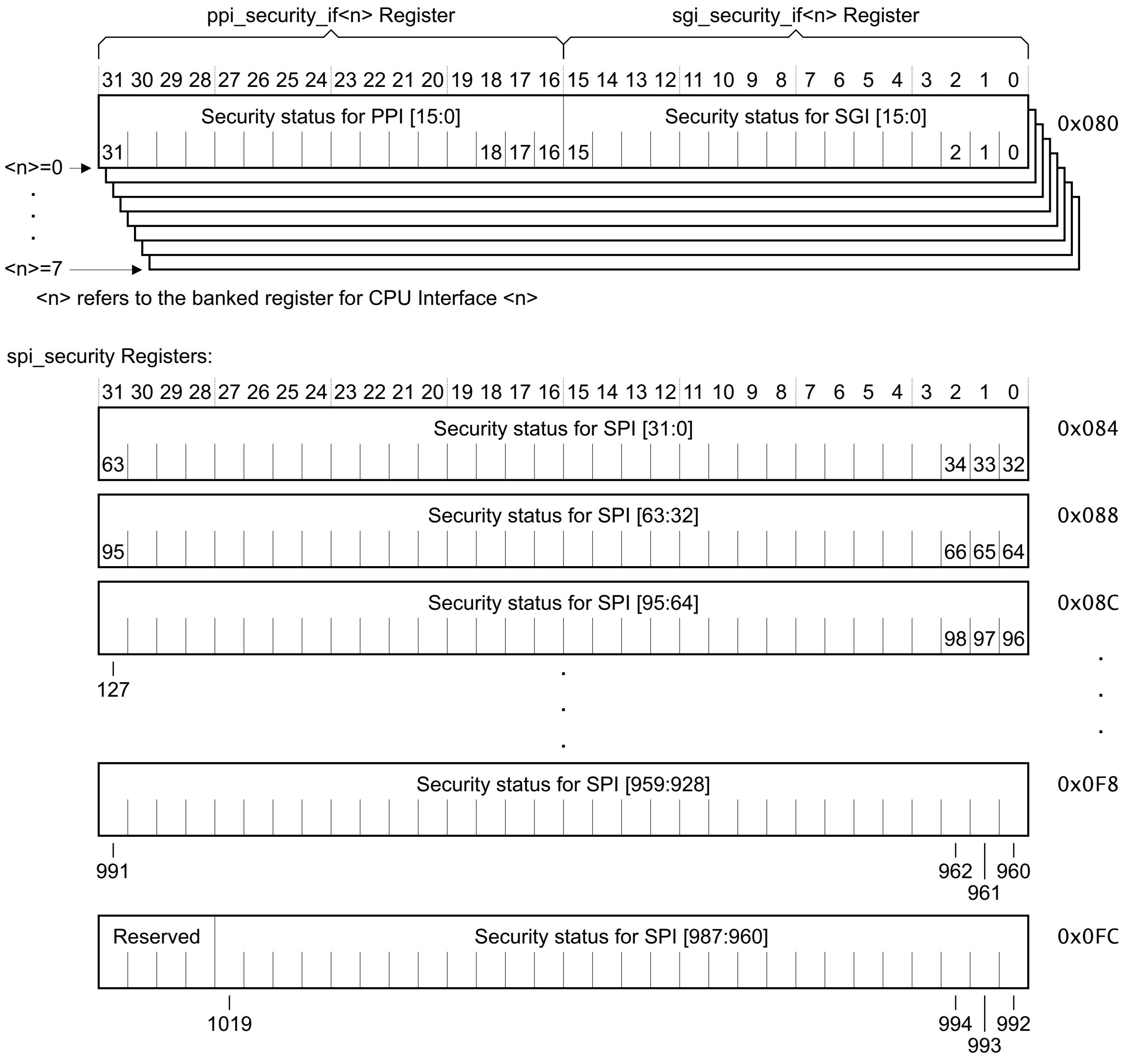

ppi_security_if<n> Register
sgi_security_if<n> Register
Security status for PPI [15:0]
Security status for SGI [15:0]
0x080
<n>=0
<n>=7
<n> refers to the banked register for CPU Interface <n>
spi_security Registers:
Security status for SPI [31:0]
0x084
Security status for SPI [63:32]
0x088
Security status for SPI [95:64]
0x08C
Security status for SPI [959:928]
0x0F8
Reserved
Security status for SPI [987:960]
0x0FC

Figure 3-5 Interrupt Security Register address map

In Figure 3-5:

- If the GIC is configured to contain more than one CPU Interface then the Distributor provides banked registers at address offset `0x080`, that contain the INTIDs for the PPIs and SGIs for the corresponding CPU Interface. The Distributor uses the ID tag that it receives on **arid_d** or **awid_d**, with the settings of **enable_d<n>** and **match_d<n>**, to select the appropriate register bank.
- If you configure the GIC to support ≤960 SPIs, then it reduces the number of spi_security Registers accordingly.

3.2.4 Enable Set Registers (ICDISERn)

Figure 3-6 on page 3-10 shows the address map that the Distributor provides for the following Enable Set Registers:

- ppi_enable_if<n> Register
- spi_enable Registers.

Note

The *ARM Generic Interrupt Controller Architecture Specification* describes the behavior of the Interrupt Set-Enable Registers (ICDISERn).

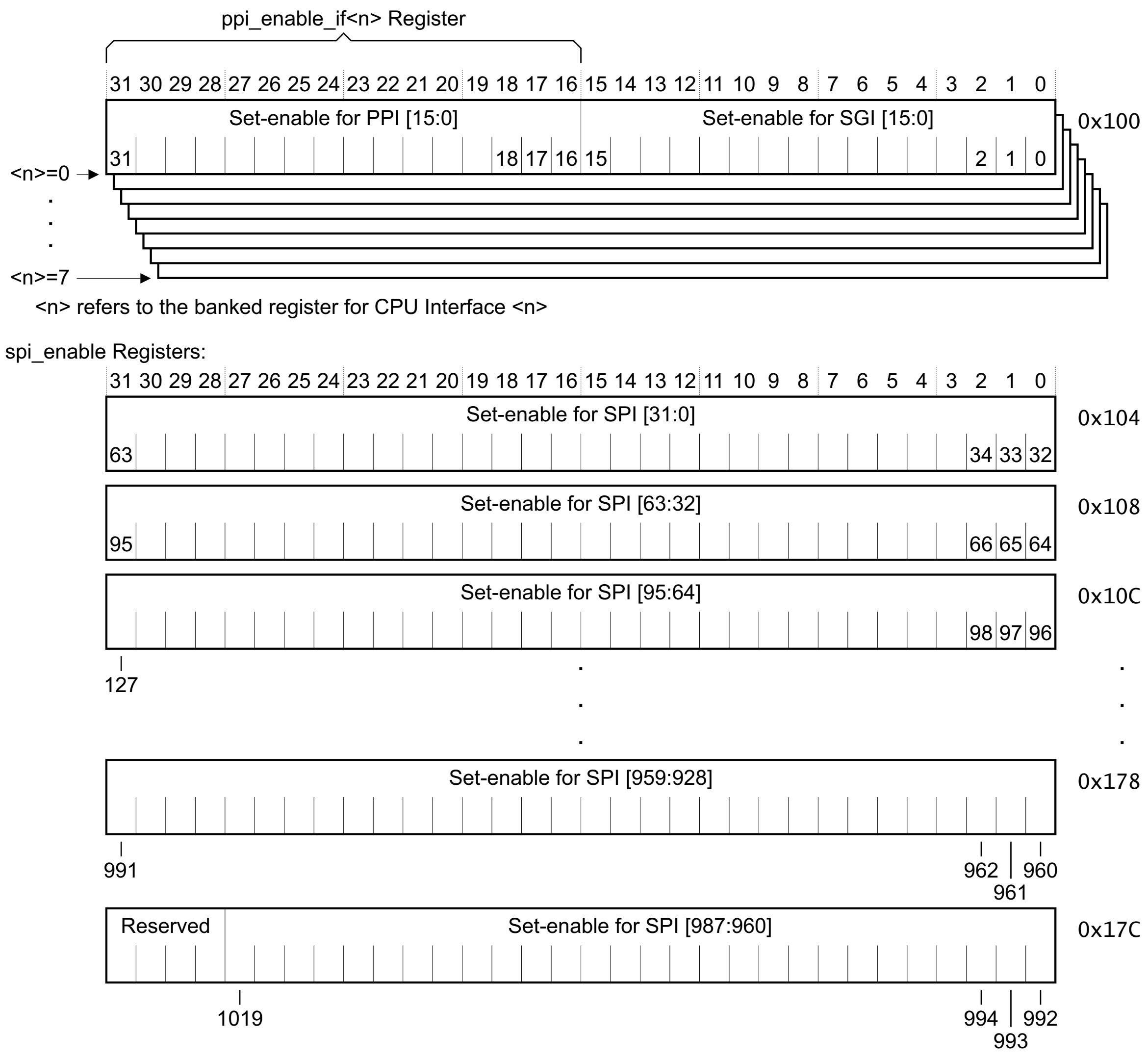

ppi_enable_if<n> Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Set-enable for PPI [15:0]
Set-enable for SGI [15:0]
31
18 17 16 15
2 1 0
0x100
<n>=0
<n>=7
<n> refers to the banked register for CPU Interface <n>
spi_enable Registers:
Set-enable for SPI [31:0]
63
34 33 32
0x104
Set-enable for SPI [63:32]
95
66 65 64
0x108
Set-enable for SPI [95:64]
98 97 96
0x10C
127
Set-enable for SPI [959:928]
0x178
991
962 961 960
Reserved
Set-enable for SPI [987:960]
0x17C
1019
994 993 992

Figure 3-6 Enable Set Register address map

In Figure 3-6:

- The Distributor does not provide registers for INTIDs < 16 because SGIs are always enabled.
- If the GIC is configured to contain more than one CPU Interface then the Distributor provides banked registers at address offset `0x100`, that contain the INTIDs for the PPIs and SGIs for the corresponding CPU Interface. The Distributor uses the ID tag that it receives on **arid_d** or **awid_d**, with the settings of **enable_d<n>** and **match_d<n>**, to select the appropriate register.
- If you configure the GIC to support ≤960 SPIs, then it reduces the number of spi_enable Registers accordingly.

3.2.5 Enable Clear Registers (ICDICERn)

Figure 3-7 shows the address map that the Distributor provides for the following Enable Clear Registers:

- ppi_enable_if<n> Register
- spi_enable Registers.

——— **Note** ———

The *ARM Generic Interrupt Controller Architecture Specification* describes the behavior of the Interrupt Clear-Enable Registers (ICDICERn).

———————————

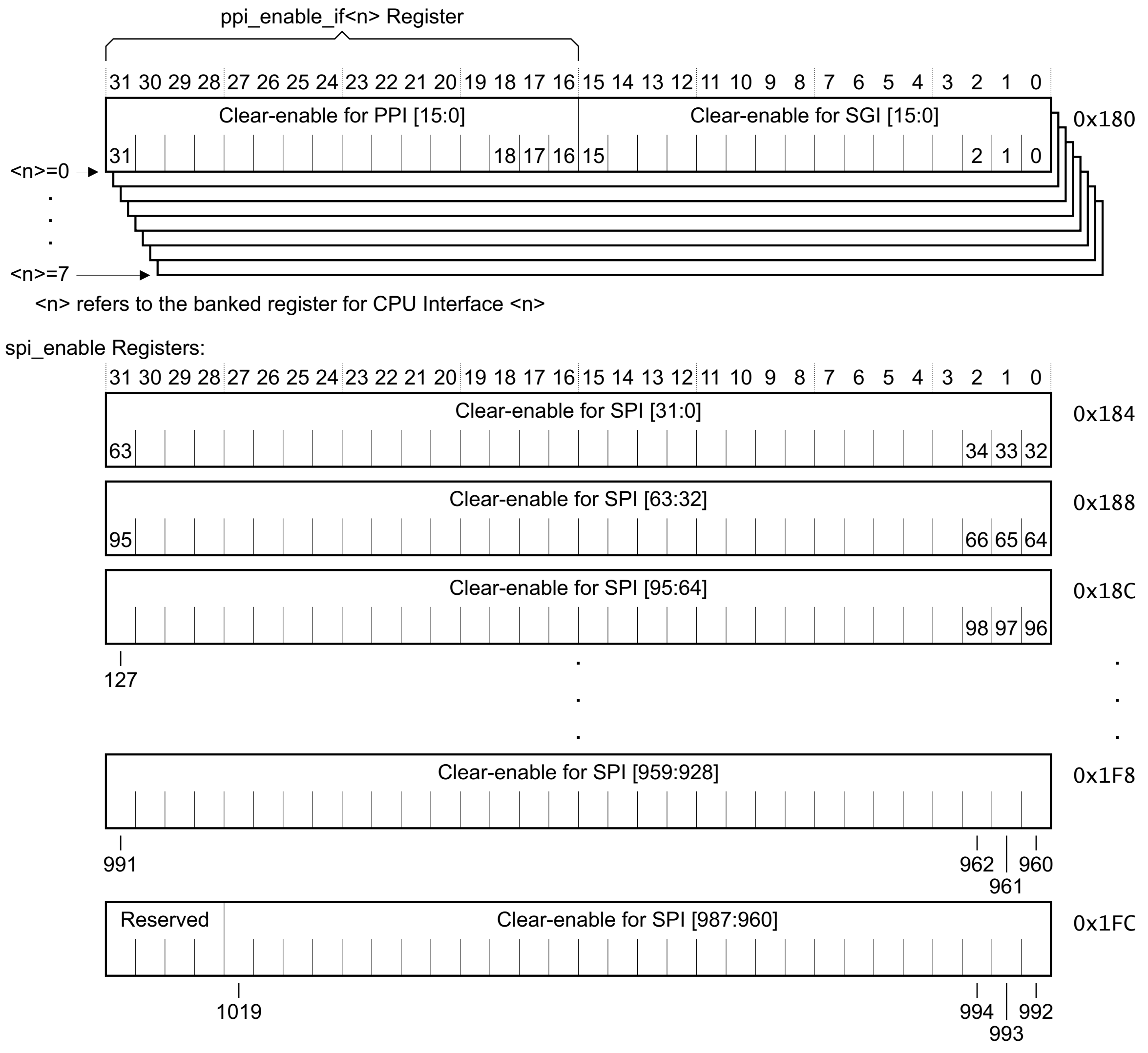

ppi_enable_if<n> Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Clear-enable for PPI [15:0]
Clear-enable for SGI [15:0]
31
18 17 16 15
2 1 0
0x180
<n>=0
<n>=7
<n> refers to the banked register for CPU Interface <n>
spi_enable Registers:
Clear-enable for SPI [31:0]
63
34 33 32
0x184
Clear-enable for SPI [63:32]
95
66 65 64
0x188
Clear-enable for SPI [95:64]
98 97 96
0x18C
127
Clear-enable for SPI [959:928]
0x1F8
991
962
961
960
Reserved
Clear-enable for SPI [987:960]
0x1FC
1019
994
993
992

Figure 3-7 Enable Clear Register address map

In Figure 3-7:

- The Distributor does not provide registers for INTIDs < 16 because SGIs are always enabled.

- If the GIC is configured to contain more than one CPU Interface then the Distributor provides banked registers at address offset `0x180`, that contain the INTIDs for the PPIs and SGIs for the corresponding CPU Interface. The Distributor uses the ID tag that it receives on **arid_d** or **awid_d**, with the settings of **enable_d<n>** and **match_d<n>**, to select the appropriate register.
- If you configure the GIC to support ≤960 SPIs, then it reduces the number of spi_enable Registers accordingly.

3.2.6 Pending Set Registers (ICDISPRn)

Figure 3-8 on page 3-13 shows the address map that the Distributor provides for the following Pending Set Registers:

- sgi_pending_if<n> Register
- ppi_pending_if<n> Register
- spi_pending Registers.

Note

The *ARM Generic Interrupt Controller Architecture Specification* describes the behavior of the Interrupt Set-Pending Registers (ICDISPRn).

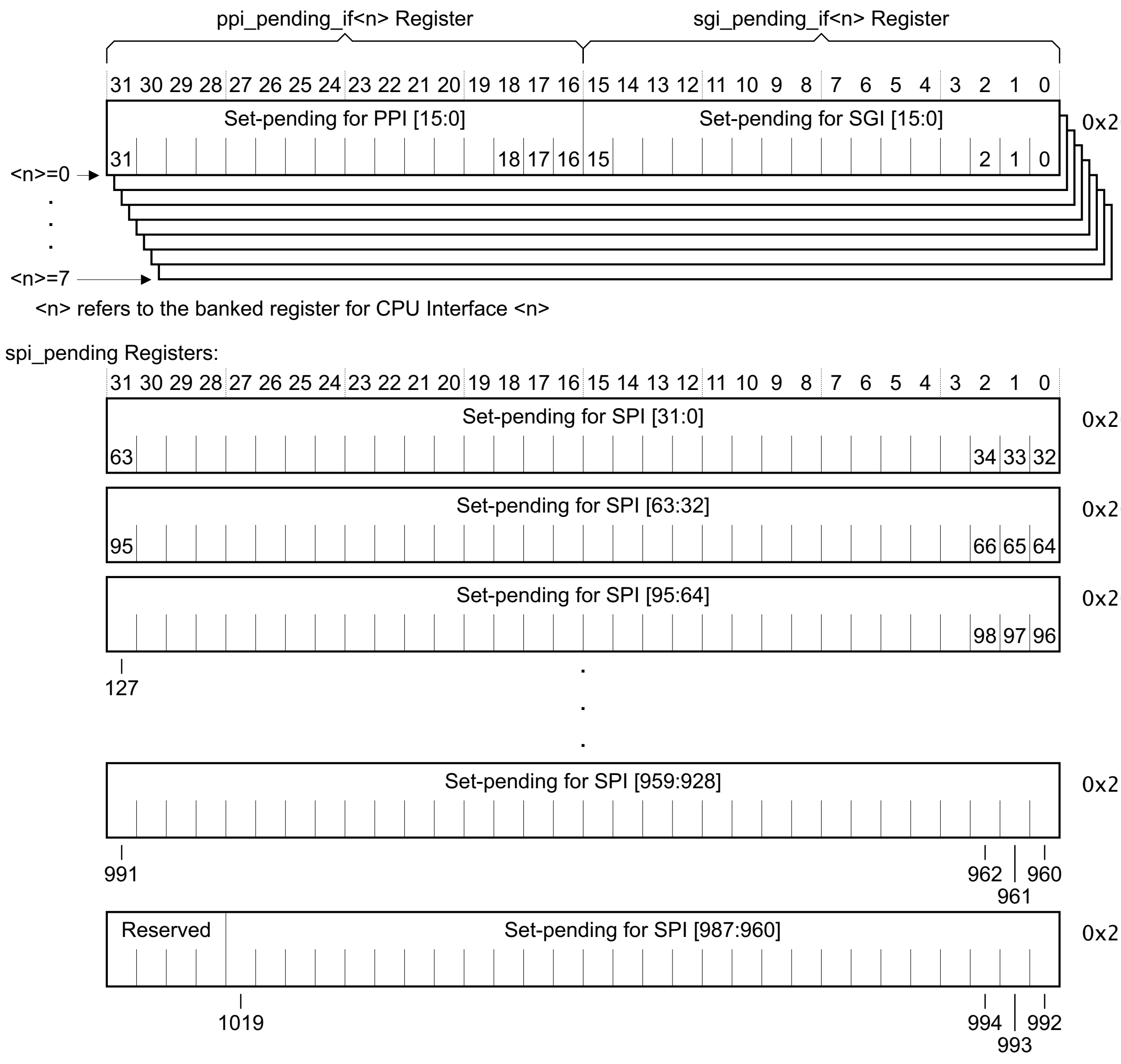

ppi_pending_if<n> Register
sgi_pending_if<n> Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Set-pending for PPI [15:0]
Set-pending for SGI [15:0]
0x200
<n>=0
<n>=7
<n> refers to the banked register for CPU Interface <n>
spi_pending Registers:
Set-pending for SPI [31:0]
0x204
Set-pending for SPI [63:32]
0x208
Set-pending for SPI [95:64]
0x20C
Set-pending for SPI [959:928]
0x278
Reserved
Set-pending for SPI [987:960]
0x27C

Figure 3-8 Pending Set Register address map

In Figure 3-8:

- The INTIDs for the SGIs are read-only. The Distributor updates these bits by using information from the sgi_control Register.
- If the GIC is configured to contain more than one CPU Interface then the Distributor provides banked registers at address offset `0x200`, that contain the INTIDs for the PPIs and SGIs for the corresponding CPU Interface. The Distributor uses the ID tag that it receives on **arid_d** or **awid_d**, with the settings of **enable_d<n>** and **match_d<n>**, to select the appropriate register.
- If you configure the GIC to support ≤960 SPIs then it reduces the number of spi_pending Registers accordingly.

3.2.7 Pending Clear Registers (ICDICPRn)

Figure 3-9 on page 3-14 shows the address map that the Distributor provides for the following Pending Clear Registers:

- sgi_pending_if<n> Register
- ppi_pending_if<n> Register

- spi_pending Registers.

Note

The *ARM Generic Interrupt Controller Architecture Specification* describes the behavior of the Interrupt Clear-Pending Registers (ICDICPRn).

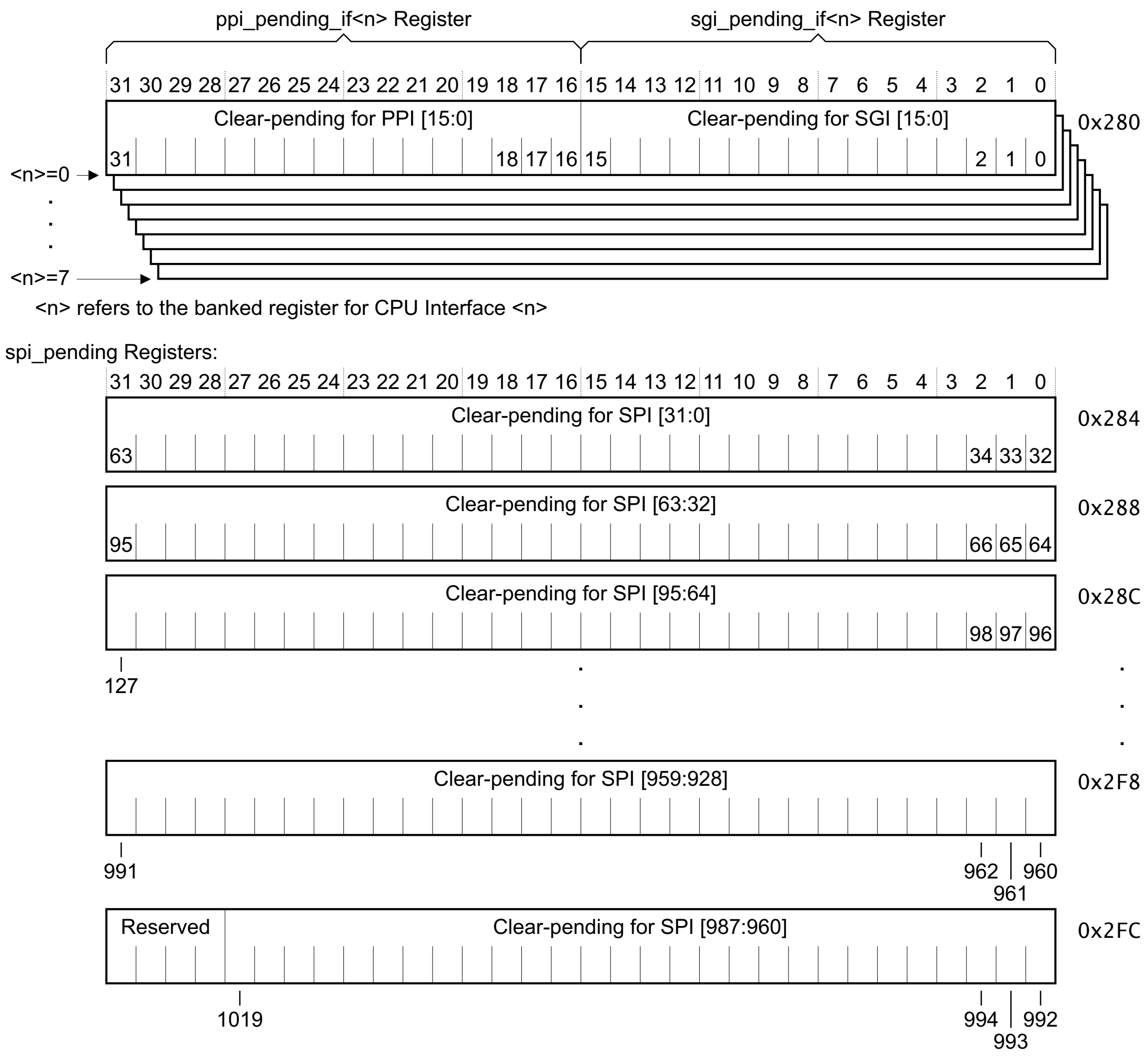

ppi_pending_if<n> Register
sgi_pending_if<n> Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Clear-pending for PPI [15:0]
Clear-pending for SGI [15:0]
0x280
<n>=0
<n>=7
<n> refers to the banked register for CPU Interface <n>
spi_pending Registers:
Clear-pending for SPI [31:0]
0x284
Clear-pending for SPI [63:32]
0x288
Clear-pending for SPI [95:64]
0x28C
Clear-pending for SPI [959:928]
0x2F8
Reserved
Clear-pending for SPI [987:960]
0x2FC

Figure 3-9 Pending Clear Register address map

In Figure 3-9:

- The INTIDs for the SGIs are read-only. The Distributor updates these bits by using information from the sgi_control Register.
- If the GIC is configured to contain more than one CPU Interface then the Distributor provides banked registers at address offset `0x280`, that contain the INTIDs for the PPIs and SGIs for the corresponding CPU Interface. The Distributor uses the ID tag that it receives on **arid_d** or **awid_d**, with the settings of **enable_d<n>** and **match_d<n>**, to select the appropriate register.
- If you configure the GIC to support ≤960 SPIs then it reduces the number of spi_pending Registers accordingly.

3.2.8 Active Status Registers (ICDABRn)

Figure 3-10 shows the address map that the Distributor provides for the following Active Status Registers:

- sgi_active_if<n> Register
- ppi_active_if<n> Register
- spi_active Registers.

——— **Note** ———

The *ARM Generic Interrupt Controller Architecture Specification* describes the behavior of the Active Bit Registers (ICDABRn).

————————

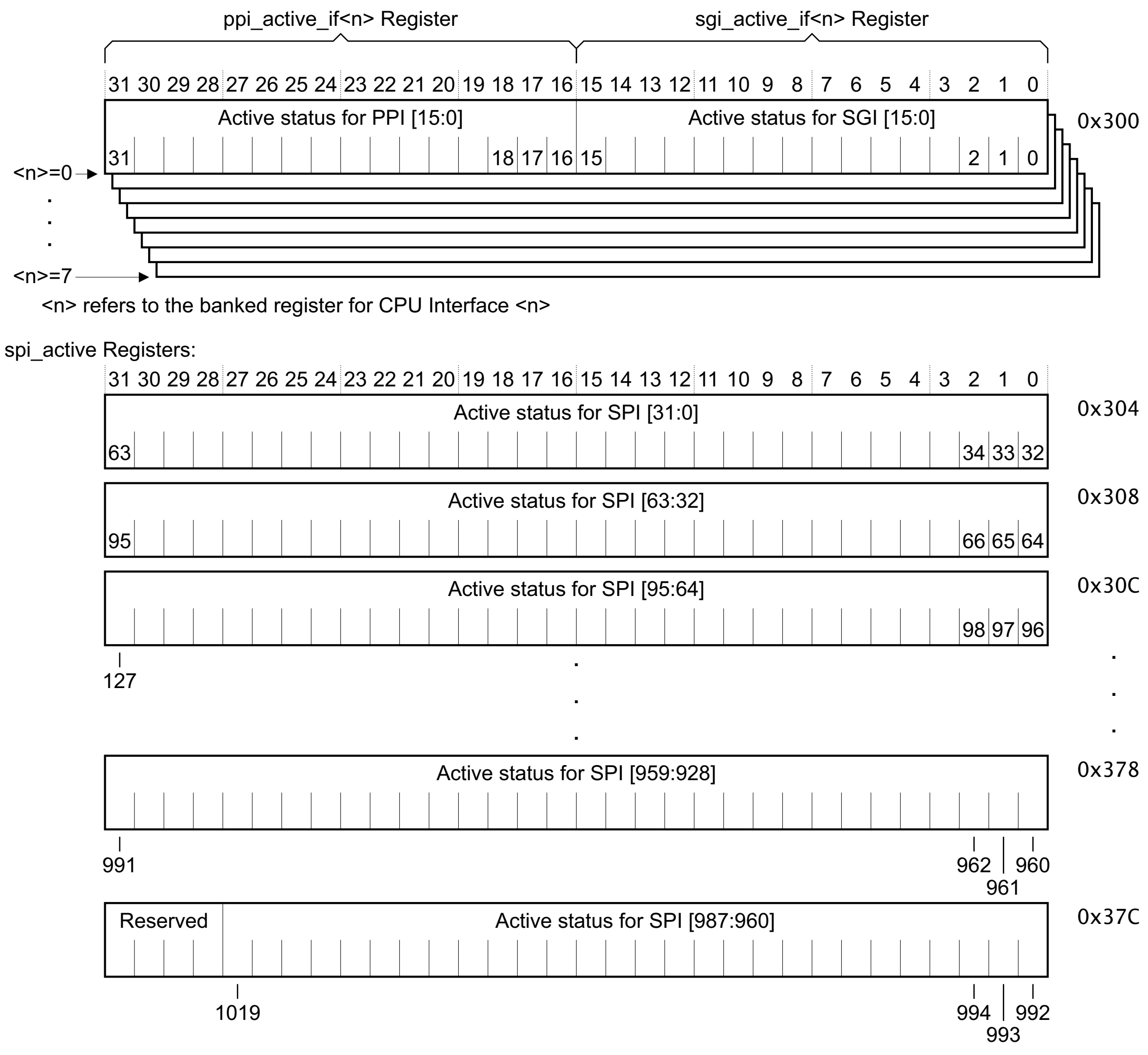

ppi_active_if<n> Register
sgi_active_if<n> Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Active status for PPI [15:0]
Active status for SGI [15:0]
0x300
<n>=0
<n>=7
<n> refers to the banked register for CPU Interface <n>
spi_active Registers:
Active status for SPI [31:0]
0x304
Active status for SPI [63:32]
0x308
Active status for SPI [95:64]
0x30C
127
Active status for SPI [959:928]
0x378
991
962
961
960
Reserved
Active status for SPI [987:960]
0x37C
1019
994
993
992

Figure 3-10 Active Status Register address map

In Figure 3-10 on page 3-15:

- If the GIC is configured to contain more than one CPU Interface then the Distributor provides banked registers at address offset `0x300`, that contain the INTIDs for the PPIs and SGIs for the corresponding CPU Interface. The Distributor uses the ID tag that it receives on **arid_d**, with the settings of **enable_d<n>** and **match_d<n>**, to select the appropriate register.
- If you configure the GIC to support ≤960 SPIs then it reduces the number of spi_active Registers accordingly.

3.2.9 Priority Level Registers (ICDIPRn)

Figure 3-11 on page 3-17 shows the address map that the Distributor provides for the following Priority Level Registers:

- priority_sgi_<INTID>_if<n> Register
- priority_ppi_<INTID>_if<n> Register
- priority_spi_<INTID> Register.

Note

- The *ARM Generic Interrupt Controller Architecture Specification* describes the behavior of the Interrupt Priority Registers (ICDIPRn).
- The AMBA Designer documentation provides information about configuring the number of priority levels, see *Additional reading* on page xii.

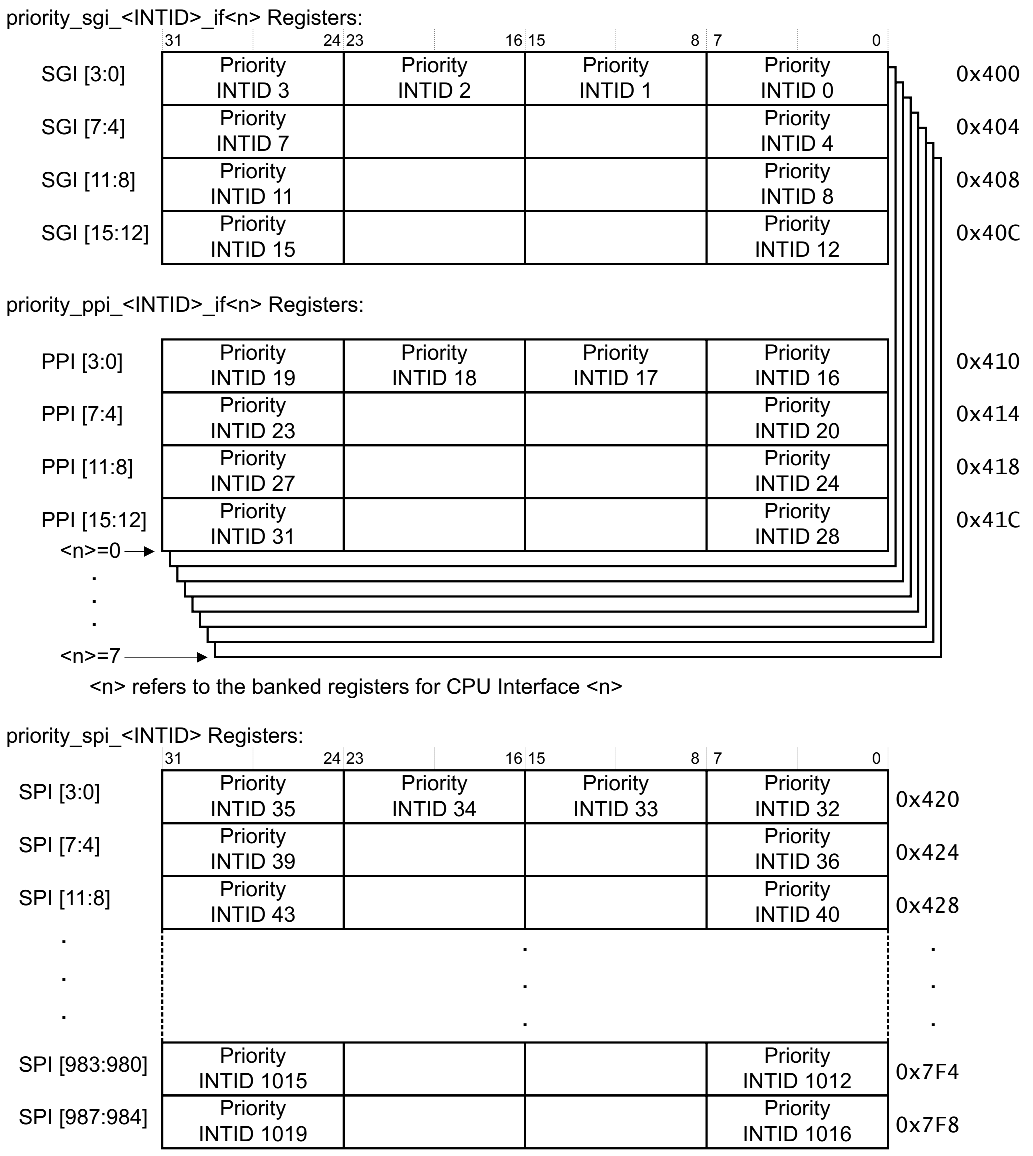


Figure 3-11 Priority Level Register address map

In Figure 3-11:

- If the GIC is configured to contain more than one CPU Interface then the Distributor provides banked registers at address offset `0x400`-`0x41C`, that contain the INTIDs for the PPIs and SGIs for the corresponding CPU Interface. The Distributor uses the ID tag that it receives on **arid_d** or **awid_d**, with the settings of **enable_d<n>** and **match_d<n>**, to select the appropriate register.
- If you configure the GIC to support ≤987 SPIs then it reduces the number of priority_spi_<INTID> Registers accordingly.

3.2.10 Target Registers (ICDIPTRn)

Figure 3-12 shows the address map that the Distributor provides for the targets_spi_<INTID> Registers.

——— **Note** ———

The *ARM Generic Interrupt Controller Architecture Specification* describes the behavior of the Interrupt Processor Targets Registers (ICDIPTRn).

———————

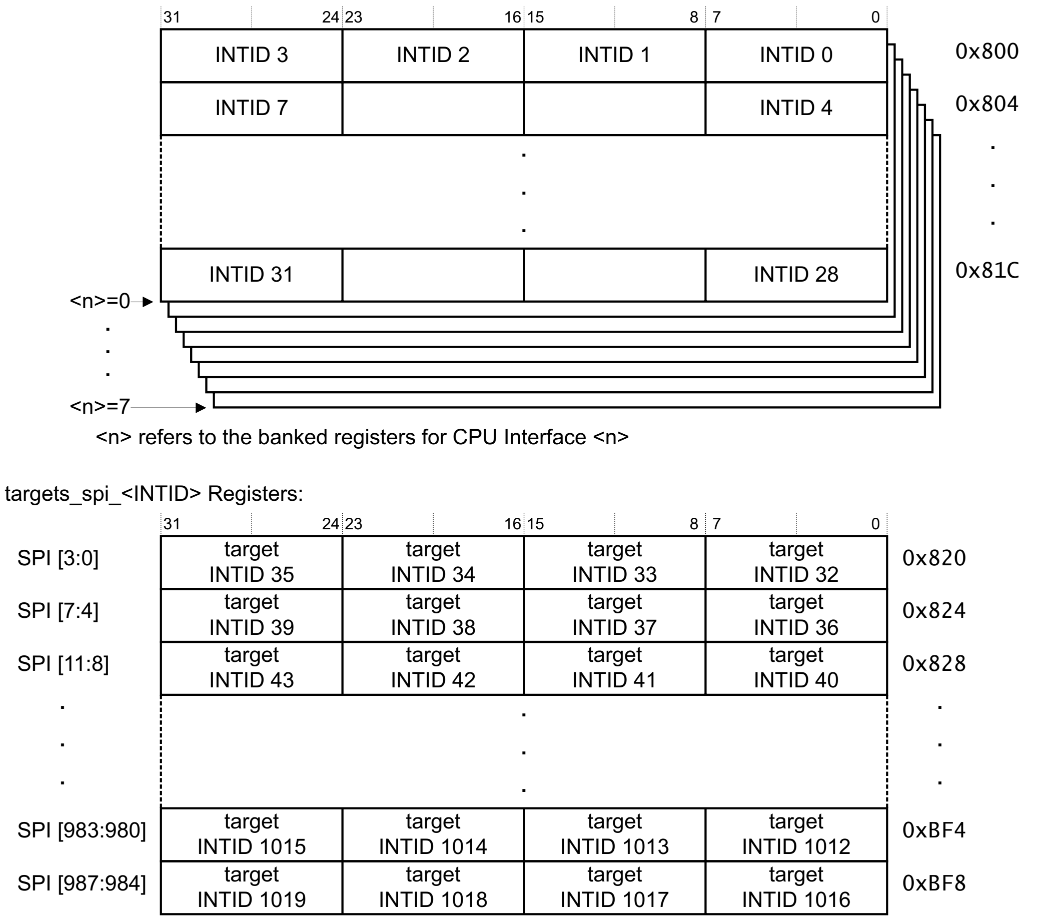


Figure 3-12 targets_spi_<INTID> Register address map

In Figure 3-12:

- The Distributor does not provide registers for INTIDs < 32. If a processor reads from a location where an SGI or PPI is implemented then the Distributor returns:

`0x01`	Processor read a targets_spi_<INTID> Register for CPU Interface 0.
`0x02`	Processor read a targets_spi_<INTID> Register for CPU Interface 1.
`0x04`	Processor read a targets_spi_<INTID> Register for CPU Interface 2.
`0x08`	Processor read a targets_spi_<INTID> Register for CPU Interface 3.
`0x10`	Processor read a targets_spi_<INTID> Register for CPU Interface 4.
`0x20`	Processor read a targets_spi_<INTID> Register for CPU Interface 5.
`0x40`	Processor read a targets_spi_<INTID> Register for CPU Interface 6.
`0x80`	Processor read a targets_spi_<INTID> Register for CPU Interface 7.

If a processor reads from a location where an SGI or PPI is not implemented then the Distributor returns `0x00`.

——— Note ———

You must use the sgi_control Register to program the target CPU Interfaces for SGIs.

———————

- If you configure the GIC to support ≤987 SPIs then it reduces the number of targets_spi_<INTID> Registers accordingly.

3.2.11 Interrupt Configuration Registers (ICDICRn)

Figure 3-13 shows the address map that the Distributor provides for the spi_config Registers.

——— Note ———

The *ARM Generic Interrupt Controller Architecture Specification* describes the behavior of the Interrupt Configuration Registers (ICDICRn).

———————

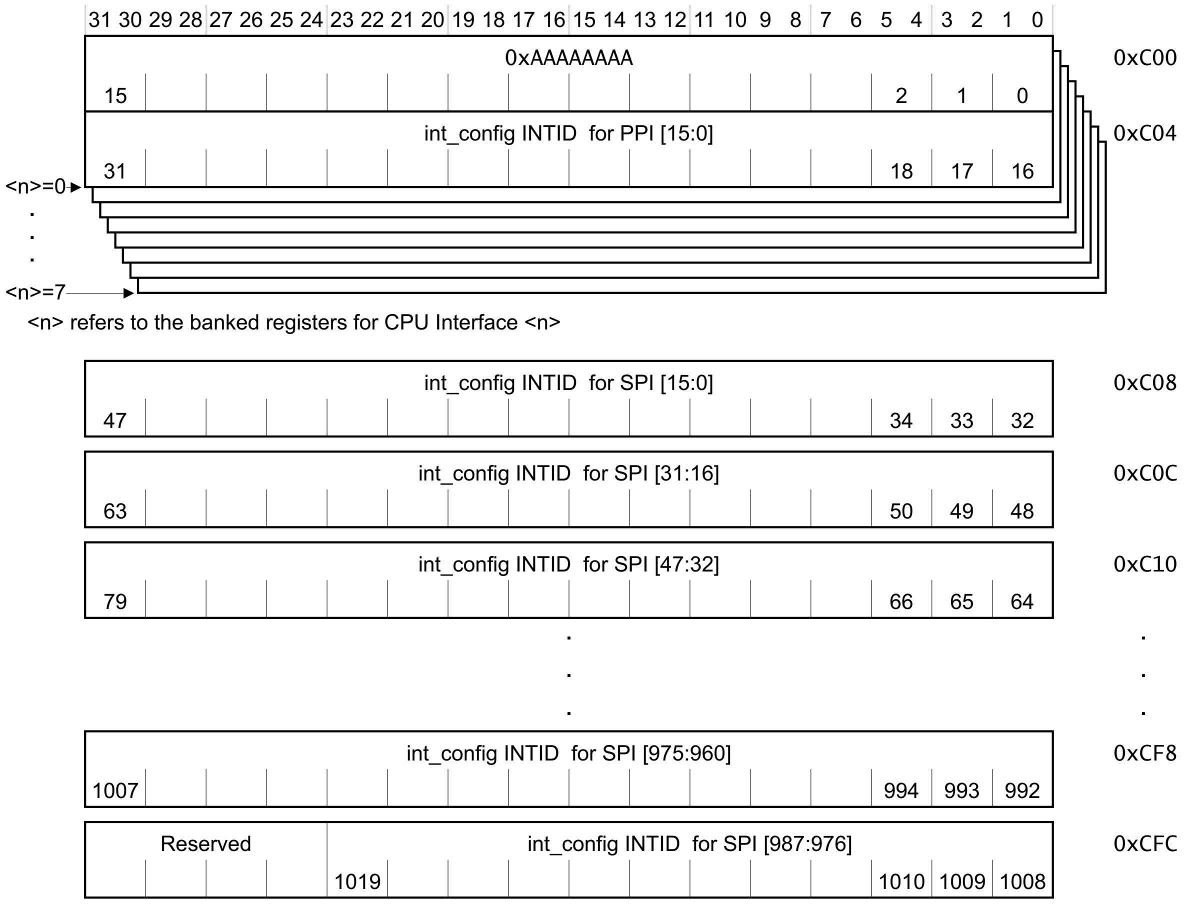

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0xAAAAAAAA
15 2 1 0
0xC00
int_config INTID for PPI [15:0]
31 18 17 16
0xC04
<n>=0
<n>=7
<n> refers to the banked registers for CPU Interface <n>
int_config INTID for SPI [15:0]
47 34 33 32
0xC08
int_config INTID for SPI [31:16]
63 50 49 48
0xC0C
int_config INTID for SPI [47:32]
79 66 65 64
0xC10
int_config INTID for SPI [975:960]
1007 994 993 992
0xCF8
Reserved
int_config INTID for SPI [987:976]
1019 1010 1009 1008
0xCFC

Figure 3-13 spi_config Register address map

In Figure 3-13:

- The Distributor does not provide a register for:
  - — INTID [15:0] because SGIs are always pulse-sensitive and use the N-N software model. The Distributor returns b10, for each bit-pair, when address `0xC00` is read.

— INTID [31:16] because PPIs are configured during the GIC configuration process. The Distributor returns either b01 or b11, for each bit-pair, when address `0xC04` is read.

- If the GIC is configured to contain more than one CPU Interface then the Distributor provides banked registers at address offset `0xC04`, that contain the INTIDs for the PPIs for the corresponding CPU Interface. The Distributor uses the ID tag that it receives on **arid_d**, with the settings of **enable_d<n>** and **match_d<n>**, to select the appropriate register.
- If you configure the GIC to support ≤976 SPIs then it reduces the number of registers accordingly. For locations where INTIDs are not implemented then the Distributor ignores writes and reads return zero.

3.2.12 PPI Status Register

The ppi_if<n> Register characteristics are:

Purpose	Each bit returns the status of the **ppi_c<n>[15:0]** inputs for CPU Interface <n>.
Usage constraints	Only accessible to processors in Secure state.
Configurations	This register is only available when the GIC is configured to provide two or more CPU Interfaces.

Note

If the GIC is configured to provide a single CPU Interface then the Distributor returns `0x00000000`.

Attributes	See the register summary in Table 3-1 on page 3-5.

Figure 3-14 shows the ppi_if<n> Register bit assignments.

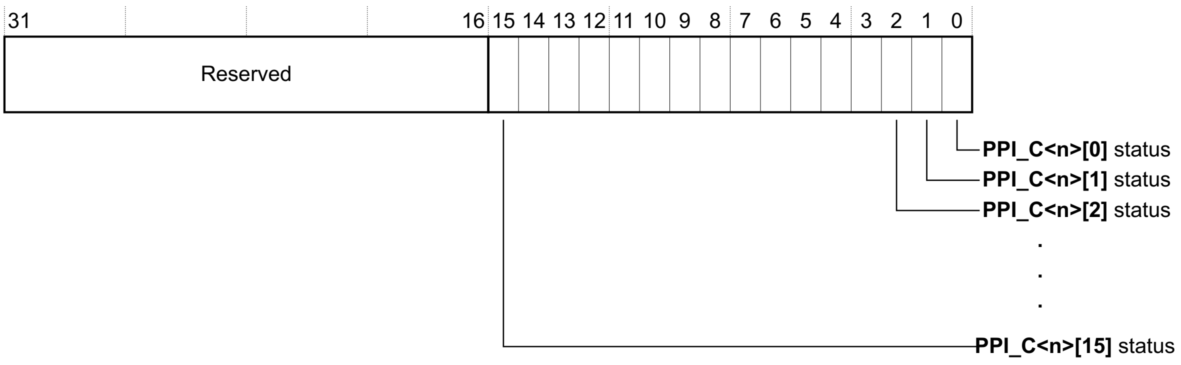


Figure 3-14 ppi_if<n> Register bit assignments

Table 3-4 shows the ppi_if<n> Register bit assignments.

Table 3-4 ppi_if<n> Register bit assignments

Bits	Name	Function
[31:16]	-	Reserved
[15:0]	ppi_status	Returns the status of the **ppi_c<n>[15:0]** inputs on the Distributor: **Bit [*X*] = 0** **ppi_c<n>[x]** is LOW **Bit [*X*] = 1** **ppi_c<n>[x]** is HIGH. **Note** These bits return the actual status of the **ppi_c<n>[15:0]** signals. The *Pending Set Registers (ICDISPRn)* on page 3-12 and *Pending Clear Registers (ICDICPRn)* on page 3-13 also provide the **ppi_c<n>[15:0]** status but because you can write to these registers then they might not contain the actual status of the **ppi_c<n>[15:0]** signals.

3.2.13 SPI Status Registers

The spi Register characteristics are:

Purpose	Each bit returns the status of an **spi[987:0]** input.
Usage constraints	Only accessible to processors in Secure state.
Configurations	This register is available in all configurations of the GIC.
Attributes	See the register summary in Table 3-1 on page 3-5.

Figure 3-15 shows the spi Register bit assignments.

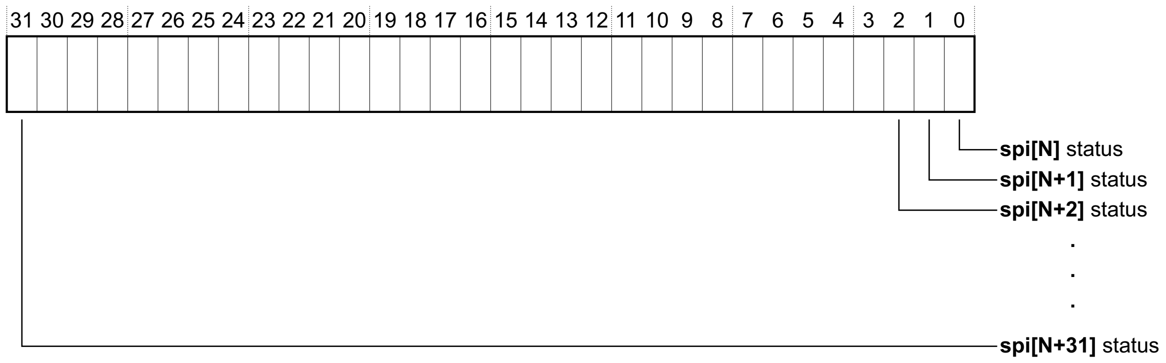


Figure 3-15 spi Register bit assignments

Table 3-5 shows the spi Register bit assignments.

Table 3-5 spi Register bit assignments

Bits	Name	Function
[31:0]	spi_status	Returns the status of the **spi[987:0]** inputs on the Distributor: **Bit [*x*] = 0** **spi[x]** is LOW **Bit [*x*] = 1** **spi[x]** is HIGH. —— **Note** —— The **spi** that *x* refers to, depends on its bit position and the base address offset of the spi Register as Figure 3-16 shows. These bits return the actual status of the **spi** signals. The *Pending Set Registers (ICDISPRn)* on page 3-12 and *Pending Clear Registers (ICDICPRn)* on page 3-13 also provide the **spi** status but because you can write to these registers then they might not contain the actual status of the **spi** signals.

Figure 3-16 shows the address map that the Distributor provides for the SPIs.

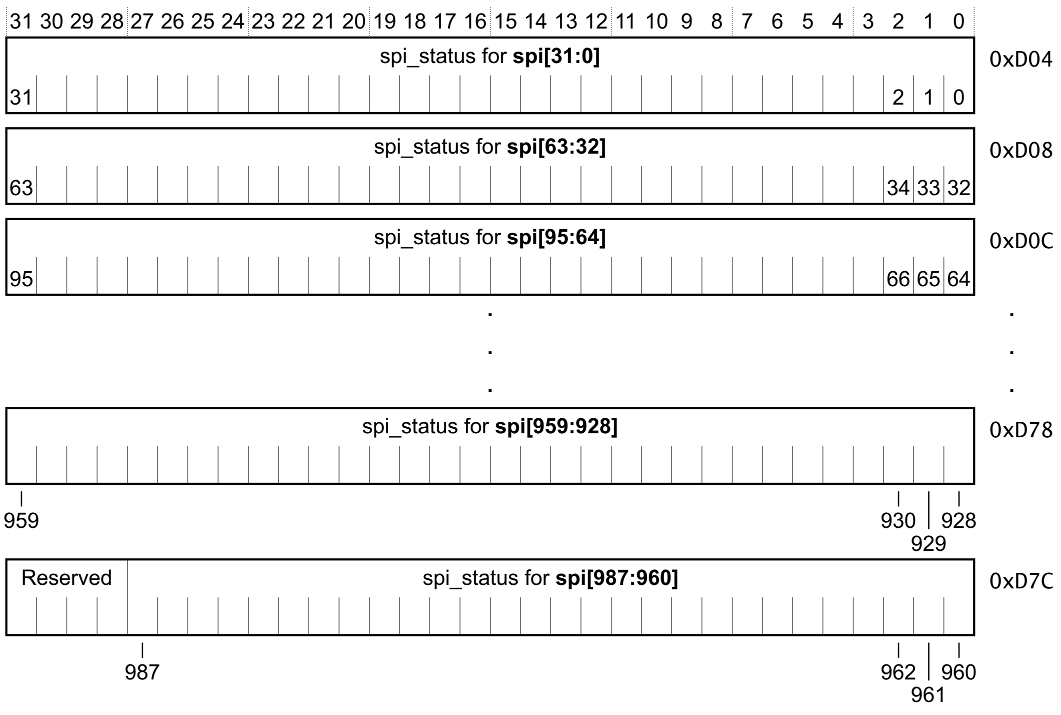


Figure 3-16 spi Register address map

If you configure the GIC to support ≤960 SPIs then it reduces the number of registers accordingly.

3.2.14 Peripheral Identification Registers

The periph_id_[8:0] Registers provide information about the configuration of the peripheral. Table 3-1 on page 3-5 shows the address base offset, reset value, and access type for these registers.

Each register provides eight bits of data but because some fields span across two adjacent periph_id registers then the following sections describe them:

- *periph_id_[3:0] register group* on page 3-23

- *periph_id_[7:4] register group* on page 3-24
- *periph_id_8 Register* on page 3-28.

periph_id_[3:0] register group

Figure 3-17 shows the periph_id_[3:0] register group bit assignments.

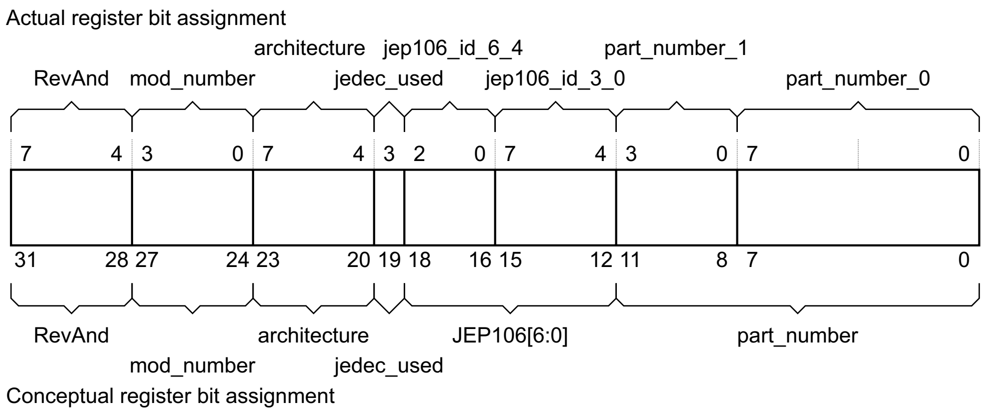


Figure 3-17 periph_id_[3:0] Register bit assignments

Table 3-6 shows the periph_id_[3:0] register group bit assignments.

Table 3-6 periph_id_[3:0] Register bit assignments

Bits	Name	Function
[31:28]	RevAnd	Identifies the manufacturer revision number.
[27:24]	mod_number	Identifies data that is relevant to the ARM partner.
[23:20]	architecture	Identifies the architecture version of the GIC.
[19]	jedec_used	Identifies if the GIC uses the JEP106 manufacturer's identity code.
[18:12]	JEP106[6:0]	Identifies the designer. This is set to b0111011, to indicate that ARM designed the peripheral.
[11:0]	part_number	Identifies the peripheral. The part number for the GIC is `0x390`.

The following subsections describe the periph_id_[3:0] registers:

- *Peripheral Identification Register 0*
- *Peripheral Identification Register 1* on page 3-24
- *Peripheral Identification Register 2* on page 3-24
- *Peripheral Identification Register 3* on page 3-24.

Peripheral Identification Register 0

The periph_id_0 Register is hard-coded and the fields in the register control the reset value. Table 3-7 shows the register bit assignments.

Table 3-7 periph_id_0 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	part_number_0	Returns `0x90`

Peripheral Identification Register 1

The periph_id_1 Register is hard-coded and the fields in the register control the reset value. Table 3-8 shows the register bit assignments.

Table 3-8 periph_id_1 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined.
[7:4]	jep106_id_3_0	JEP106 identity code [3:0]. See the *JEDEC Standard Manufacturer's Identification Code*. These bits read back as `0xB` because ARM is the designer of the peripheral.
[3:0]	part_number_1	Returns `0x3`.

Peripheral Identification Register 2

The periph_id_2 Register is hard-coded and the fields in the register control the reset value. Table 3-9 shows the register bit assignments.

Table 3-9 periph_id_2 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined.
[7:4]	architecture	Identifies the architecture version of the GIC. For revision r0p0, this field returns `0x0`. However, the GIC implements version 1.0 of the *ARM Generic Interrupt Controller Architecture Specification*.
[3]	jedec_used	This indicates that the GIC uses a manufacturer's identity code that was allocated by JEDEC according to JEP106. This bit always returns `0x1`.
[2:0]	jep106_id_6_4	JEP106 identity code [6:4]. See the *JEDEC Standard Manufacturer's Identification Code*. These bits read back as b011 because ARM is the designer of the peripheral.

Peripheral Identification Register 3

The periph_id_3 Register is hard-coded and the fields in the register control the reset value. Table 3-10 shows the register bit assignments.

Table 3-10 periph_id_3 Register bit assignments

Bit	Name	Description
[31:8]	-	Undefined.
[7:4]	RevAnd	The top-level RTL provides four AND gates that are tied-off to provide an output value of `0x0`. Once silicon is available, if metal fixes are necessary then the manufacturer can modify the tie-offs to indicate that a revision of the silicon has occurred.
[3:0]	mod_number	The customer can update this field if they modify the RTL of the GIC. ARM set this to `0x0`.

periph_id_[7:4] register group

Figure 3-18 on page 3-25 shows the periph_id_[7:4] register group bit assignments.

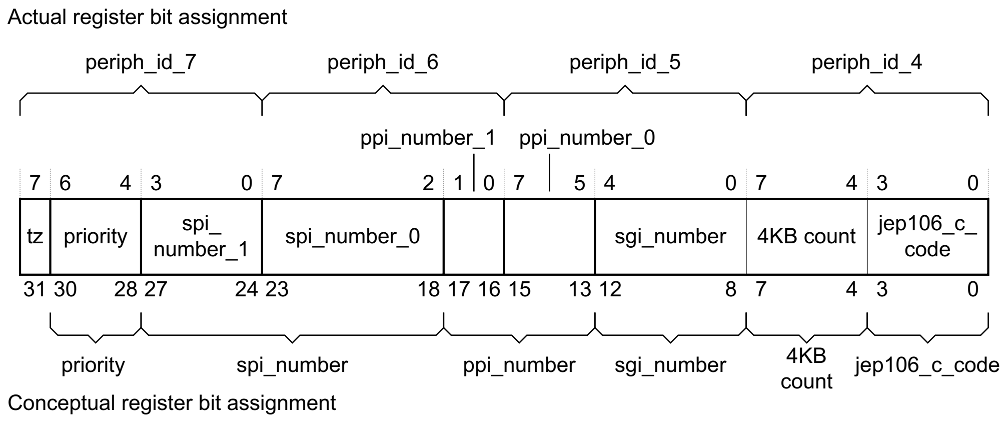


Figure 3-18 periph_id_[7:4] Register bit assignments

Table 3-11 shows the periph_id_[7:4] register group bit assignments.

Table 3-11 periph_id_[7:4] Register bit assignments

Bits	Name	Function
[31]	tz	Identifies the number of security states that the GIC supports. See Table 3-15 on page 3-27.
[30:28]	priority	Identifies the number of priority levels that the GIC provides. See Table 3-15 on page 3-27.

[27:18]	spi_number	Identifies the number of SPIs that the GIC provides:
		b0000000000 = reserved
		b0000000001 = one SPI, INTID32
		b0000000010 = two SPIs, INTID[33:32]
		b0000000011 = three SPIs, INTID[34:32]
		.
		.
		.
		b1111011100 = 988 SPIs, INTID[1019:32]
		b1111011101-b1111111111 = reserved.

[17:13]	ppi_number	Identifies the number of PPIs that the GIC provides:
		b00000 = no PPIs
		b00001 = one PPI, INTID16
		b00010 = two PPIs, INTID[17:16]
		.
		.
		.
		b10000 = 16 PPIs, INTID[31:16]
		b10001-b11111 = reserved.

[12:8]	sgi_number	Identifies the number of SGIs that the GIC provides. See Table 3-13 on page 3-26.
[7:4]	4KB count	Identifies the address space that the registers occupy. See Table 3-12 on page 3-26.
[3:0]	jep106_c_code	Identifies the JEP106 continuation code. See Table 3-12 on page 3-26.

The following subsections describe the periph_id_[7:4] registers:

- *Peripheral Identification Register 4* on page 3-26
- *Peripheral Identification Register 5* on page 3-26

- *Peripheral Identification Register 6* on page 3-27
- *Peripheral Identification Register 7* on page 3-27.

Peripheral Identification Register 4

The periph_id_4 Register is hard-coded and the fields in the register control the reset value. Table 3-12 shows the register bit assignments.

Table 3-12 periph_id_4 Register bit assignments

Bits	Name	Function
[31:8]	-	Undefined.
[7:4]	4KB count	The number of 4KB address blocks you require, to access the registers, expressed in powers of 2. These bits read back as `0x0`.
[3:0]	jep106_c_code	The JEP106 continuation code value represents how many `0x7F` continuation characters occur in the manufacturer's identity code. See *JEDEC Standard Manufacturer's Identification Code*. These bits return `0x4`.

Peripheral Identification Register 5

The periph_id_5 Register is hard-coded and the fields in the register control the reset value. Table 3-13 shows the register bit assignments.

Table 3-13 periph_id_5 Register bit assignments

Bits	Name	Function
[31:8]	-	Undefined.
[7:5]	ppi_number_0	The LSBs of the number of PPIs that the GIC provides. See Figure 3-18 on page 3-25 and Table 3-11 on page 3-25 for information about how to concatenate this field to create the ppi_number field.

[4:0]	sgi_number	The number of SGIs that the GIC provides.
		b00000 = no SGIs
		b00001 = one SGI, INTID0
		b00010 = two SGIs, INTID[1:0]
		.
		.
		.
		b10000 = 16 SGIs, INTID[15:0]
		b10001-b11111 = reserved.

Peripheral Identification Register 6

The periph_id_6 Register is hard-coded and the fields in the register control the reset value. Table 3-14 shows the bit assignments for this register.

Table 3-14 periph_id_6 Register bit assignments

Bits	Name	Function
[31:8]	-	Undefined.
[7:2]	spi_number_0	The LSBs of the number of SPIs that the GIC provides. See Figure 3-18 on page 3-25 and Table 3-11 on page 3-25 for information about how to concatenate this field to create the spi_number field.
[1:0]	ppi_number_1	The MSBs of the number of PPIs that the GIC provides. See Figure 3-18 on page 3-25 and Table 3-11 on page 3-25 for information about how to concatenate this field to create the ppi_number field.

Peripheral Identification Register 7

The periph_id_7 Register is hard-coded and the fields in the register control the reset value. Table 3-15 shows the bit assignments for this register.

Table 3-15 periph_id_7 Register bit assignments

Bits	Name	Function
[31:8]	-	Undefined
[7]	tz	Identifies the number of security states that the GIC supports: 0 = one security state, that is, the Secure state 1 = two security states, that is, the Secure state and the Non-secure state.

[6:4]	priority	The number of priority levels that the GIC provides:
		b000 = 16 priority levels
		b001 = 32 priority levels
		b010 = 64 priority levels
		b011 = 128 priority levels
		b100 = 256 priority levels
		b101-b111 = reserved.

[3:0]	spi_number_1	The MSBs of the number of SPIs that the GIC provides. See Figure 3-18 on page 3-25 and Table 3-11 on page 3-25 for information about how to concatenate this field to create the spi_number field.

periph_id_8 Register

Figure 3-19 shows the periph_id_8 Register bit assignments.

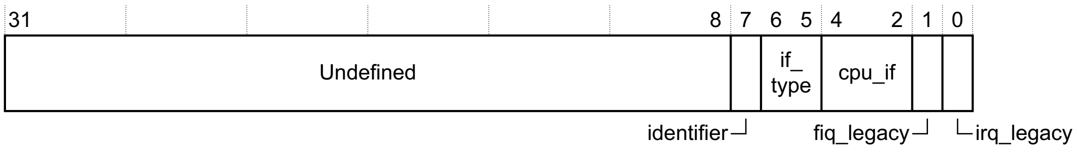


Figure 3-19 periph_id_8 Register bit assignments

Table 3-16 shows the periph_id_8 Register bit assignments.

Table 3-16 periph_id_8 Register bit assignments

Bits	Name	Function
[31:8]	-	Undefined.
[7]	identifier	Identifies the AMBA interface that this register belongs to: 0 = Distributor 1 = CPU Interface.

[6:5]	if_type	Identifies the AMBA protocol that the GIC supports:
		b00 = AXI
		b01 = AHB-Lite
		b10-b11 = reserved.

[4:2]	cpu_if	Identifies the number of CPU Interfaces that the GIC contains:
		b000 = 1
		b001 = 2
		b010 = 3
		b011 = 4
		b100 = 5
		b101 = 6
		b110 = 7
		b111 = 8.

[1]	fiq_legacy	Identifies if the GIC provides a legacy FIQ input signal for each CPU Interface: 0 = not supported 1 = **legacy_nfiq_c<n>** inputs are supported.
[0]	irq_legacy	Identifies if the GIC provides a legacy IRQ input signal for each CPU Interface: 0 = not supported 1 = **legacy_nirq_c<n>** inputs are supported.

3.2.15 PrimeCell Identification Registers

The component_id_[3:0] Register characteristics are:

Purpose	When concatenated, these four registers return `0xB105F00D`.
Usage constraints	Not accessible in the Reset state.
Configurations	Available in all configurations of the GIC.
Attributes	See the register summary in Table 3-1 on page 3-5.

These registers can be treated conceptually as a single register that holds a 32-bit component identification value. You can use the register for automatic BIOS configuration.

Figure 3-20 shows the register bit assignments.

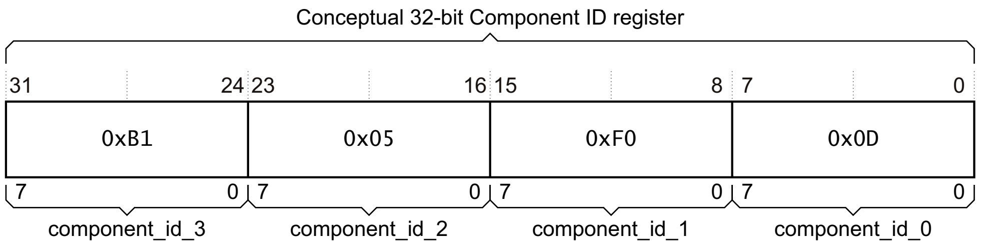


Figure 3-20 PrimeCell ID Register bit assignments

Table 3-17 shows the register bit assignments.

Table 3-17 component_id Register bit assignments

component_id Register		component_id_[3:0] registers		
Bits	**Reset value**	**Register**	**Bits**	**Description**
[31:24]	`0xB1`	component_id_3	[31:8]	Read undefined
			[7:0]	Returns `0xB1`
[23:16]	`0x05`	component_id_2	[31:8]	Read undefined
			[7:0]	Returns `0x05`
[15:8]	`0xF0`	component_id_1	[31:8]	Read undefined
			[7:0]	Returns `0xF0`
[7:0]	`0x0D`	component_id_0	[31:8]	Read undefined
			[7:0]	Returns `0x0D`

3.3 CPU Interface register descriptions

This section describes the registers that each CPU Interface provides. Table 3-18 shows the CPU Interface registers and provides a reference to the register description, which is either in this book or the *ARM Generic Interrupt Controller Architecture Specification*.

Table 3-18 CPU Interface register summary

Offset	Name	Type	Reset	Width	Description
`0x000`	control<n>[a]	RW	`0x00000000`	32	*CPU Interface Control Register (ICCICR)* on page 3-31
`0x004`	pri_msk_c<n>[a]	RW	`0x00000000`	32	Priority Mask Register (ICCPMR)[b]
`0x008`	bp_c<n>[a], nsbp_c<n>[a]	RW	-[c]	32	Binary Point Register (ICCBPR)[b]
`0x00C`	int_ack<n>[a]	RO	`0x000003FF`	32	Interrupt Acknowledge Register (ICCIAR)[b]
`0x010`	eoi<n>[a]	WO	-	32	End of Interrupt Register (ICCEOIR)[b]
`0x014`	run_priority<n>[a]	RO	`0x000000FF`	32	Running Priority Register (ICCRPR)[b]
`0x018`	hi_pend<n>[a]	RO	`0x000003FF`	32	Highest Pending Interrupt Register (ICCHPIR)[b]
`0x01C`[d]	alias_nsbp_c<n>[a]	RW	`0x00000000`	32	Aliased Binary Point Register (ICCABPR)[b]
`0x020-0x03C`	-	-	-	-	Reserved
`0x040` `0x044`	integ_en_c<n>[a] interrupt_out<n>[a]	See Chapter 4 *Programmers Model for Test* for information about these registers			
`0x048-0x04C`	-	-	-	-	Reserved
`0x050` `0x054`	match_c<n>[a] enable_c<n>[a]	See Chapter 4 *Programmers Model for Test* for information about these registers			
`0x058-0x0F8`	-	-	-	-	Reserved
`0x0FC`	cpu_if_ident	RO	`0x390--43B`[e]	32	*CPU Interface Implementer Identification Register (ICCIIDR)* on page 3-31
`0x100-0xFBC`	-	-	-	-	Reserved
`0xFC0`	periph_id_8	RO	-[c]	8	*Peripheral Identification Registers* on page 3-32
`0xFC4-0xFCC`	-	-	-	-	Reserved
`0xFD0-0xFDC` `0xFE0-0xFEC`	periph_id_[7:4] periph_id_[3:0]	RO	-[c]	8	*Peripheral Identification Registers* on page 3-32
`0xFF0-0xFFC`	component_id_[3:0]	RO	`0xB105F00D`	8	*PrimeCell Identification Registers* on page 3-32

a. <n> corresponds to the number of a CPU Interface. If the GIC contains two or more CPU Interfaces then the **enable_c<n>** and **match_c<n>** signals control which CPU Interface is selected.
b. See the *ARM Generic Interrupt Controller Architecture Specification*.
c. The reset value depends on the configuration of the GIC.
d. This address location is only accessible when a processor in Secure state performs a secure access.
e. The reset value depends on the architecture version and the revision of the GIC. See *CPU Interface Implementer Identification Register (ICCIIDR)* on page 3-31.

3.3.1 CPU Interface Control Register (ICCICR)

The *ARM Generic Interrupt Controller Architecture Specification* describes the behavior of the CPU Interface Control Register (ICCICR).

Note

- The setting of the FIQEn, EnableNS, and EnableS bits control how the GIC responds to legacy interrupts.
- If you program the EnableNS bit to 0 then, under certain conditions, denial of service might occur because this disables the CPU Interface from issuing non-secure interrupts. See *Risk of Denial of Service (DoS) when the GIC supports the Security Extensions* on page 3-36 for more information.

3.3.2 CPU Interface Implementer Identification Register (ICCIIDR)

The cpu_if_ident Register characteristics are:

Purpose	Returns information about the implementer of the CPU Interface and the revision of the GIC.
Usage constraints	No usage constraints.
Configurations	This register is available in all configurations of the GIC.
Attributes	See the register summary in Table 3-18 on page 3-30.

Figure 3-21 shows the cpu_if_ident Register bit assignments.

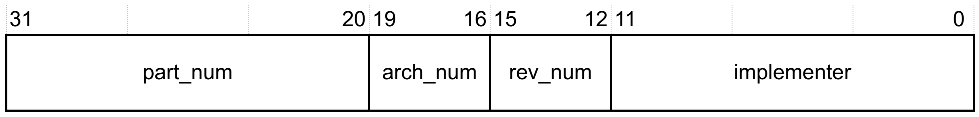


Figure 3-21 cpu_if_ident Register bit assignments

Table 3-19 shows the cpu_if_ident Register bit assignments.

Table 3-19 cpu_if_ident Register bit assignments

Bits	Name	Function
[31:20]	part_num	Identifies the peripheral. The part number of the GIC is `0x390`.
[19:16]	arch_num	Identifies the version of the *ARM Generic Interrupt Controller Architecture Specification* that the GIC implements. For version 1.0, this field returns `0x1`.
[15:12]	rev_num	Returns the revision number of the GIC. For revision r0p0, this field returns `0x0`.
[11:0]	implementer	Returns the JEP106 code of the company that implemented the Distributor RTL, that is, ARM. It uses the following bit format: [11:8] = `0x4`, that is, the JEP106 continuation code for ARM [7] = 0 [6:0] = b0111011, that is, the JEP106 code [6:0] for ARM. See the *JEDEC Standard Manufacturer's Identification Code*.

3.3.3 Peripheral Identification Registers

The periph_id_[8:0] Registers are identical to those that the GIC provides for the Distributor. See *Peripheral Identification Registers* on page 3-22.

Note

Because periph_id_8 Register contains a bit that identifies the name of the AMBA interface, then this is the only periph_id Register to alter in value. The periph_id_[7:0] registers for the CPU Interface contain the same data as the periph_id_[7:0] registers for the Distributor.

Table 3-18 on page 3-30 shows the address base offset and access type for these registers.

3.3.4 PrimeCell Identification Registers

The component_id_[3:0] Registers are identical to those that the GIC provides for the Distributor. See *PrimeCell Identification Registers* on page 3-29.

Table 3-18 on page 3-30 shows the address base offset and access type for these registers.

3.4 Additional programming information

The following sections describe:

- *Software initialization process for GICs that support a single security state*
- *Software initialization process for GICs that support the Security Extensions* on page 3-34
- *Risk of Denial of Service (DoS) when the GIC supports the Security Extensions* on page 3-36.

3.4.1 Software initialization process for GICs that support a single security state

After deassertion of **gresetn**, the Distributor and CPU Interfaces are disabled. Figure 3-22 shows the initialization that software must perform to enable the GIC to route the interrupts to the appropriate processors.

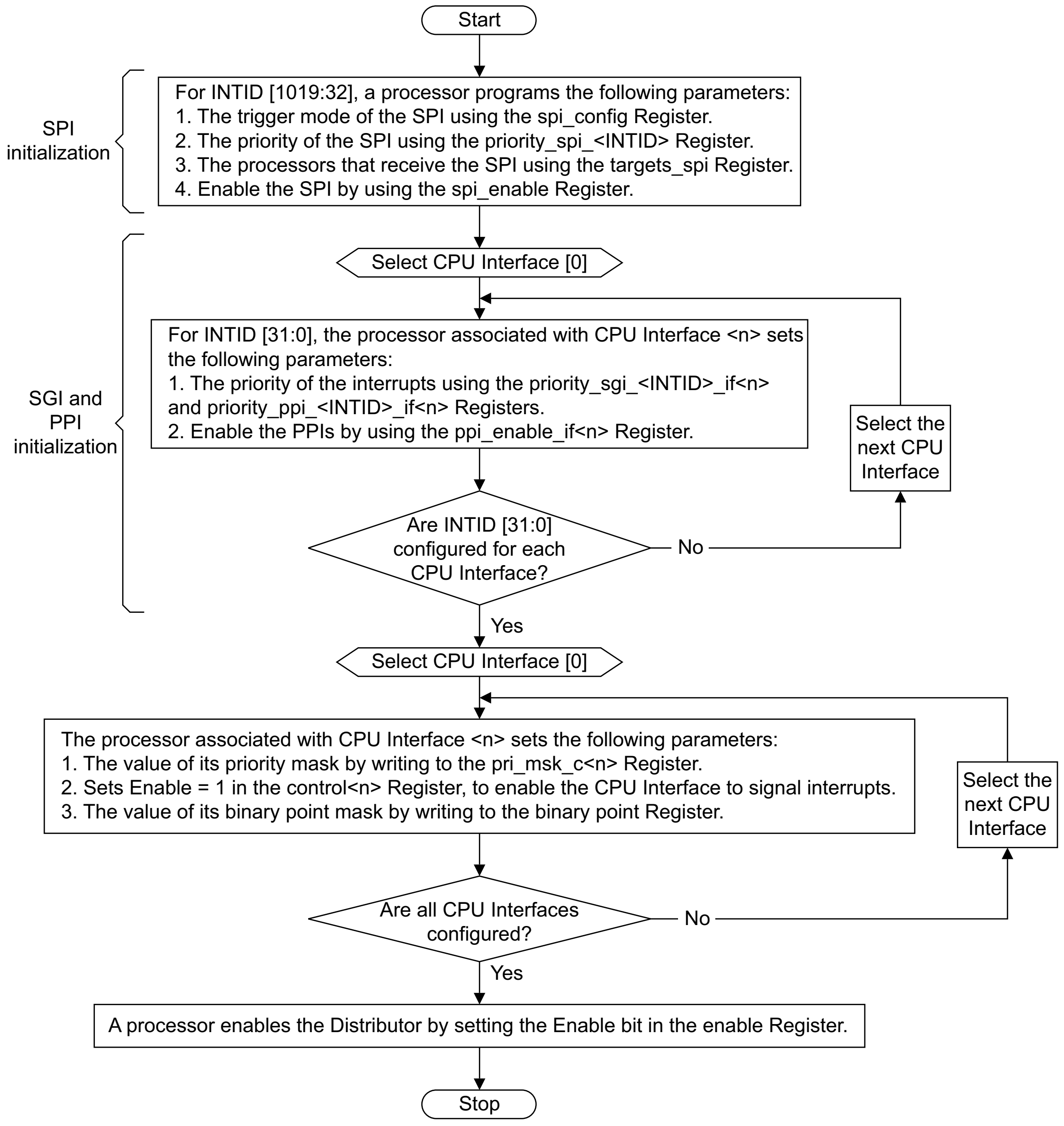

Start
SPI initialization
For INTID [1019:32], a processor programs the following parameters:
1. The trigger mode of the SPI using the spi_config Register.
2. The priority of the SPI using the priority_spi_<INTID> Register.
3. The processors that receive the SPI using the targets_spi Register.
4. Enable the SPI by using the spi_enable Register.
SGI and PPI initialization
Select CPU Interface [0]
For INTID [31:0], the processor associated with CPU Interface <n> sets the following parameters:
1. The priority of the interrupts using the priority_sgi_<INTID>_if<n> and priority_ppi_<INTID>_if<n> Registers.
2. Enable the PPIs by using the ppi_enable_if<n> Register.
Select the next CPU Interface
Are INTID [31:0] configured for each CPU Interface?
No
Yes
Select CPU Interface [0]
The processor associated with CPU Interface <n> sets the following parameters:
1. The value of its priority mask by writing to the pri_msk_c<n> Register.
2. Sets Enable = 1 in the control<n> Register, to enable the CPU Interface to signal interrupts.
3. The value of its binary point mask by writing to the binary point Register.
Select the next CPU Interface
Are all CPU Interfaces configured?
No
Yes
A processor enables the Distributor by setting the Enable bit in the enable Register.
Stop

Figure 3-22 Initialization process for GICs that support a single security state

3.4.2 Software initialization process for GICs that support the Security Extensions

After deassertion of **gresetn**, the Distributor and CPU Interfaces are disabled. Figure 3-23 to Figure 3-24 on page 3-35 show the initialization that software must perform to enable the GIC to route the interrupts to the appropriate processors.

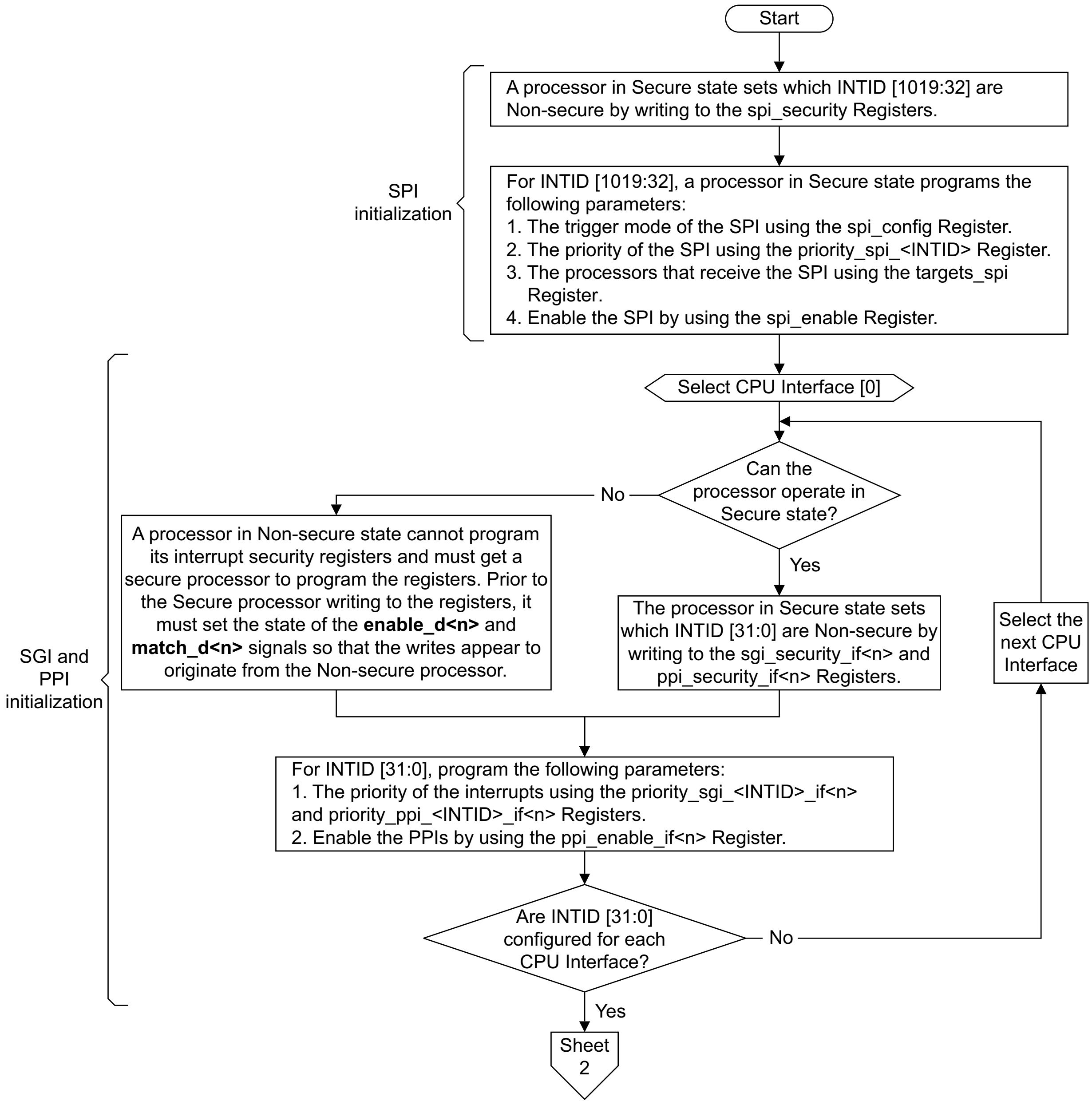

Start
A processor in Secure state sets which INTID [1019:32] are Non-secure by writing to the spi_security Registers.
For INTID [1019:32], a processor in Secure state programs the following parameters:
1. The trigger mode of the SPI using the spi_config Register.
2. The priority of the SPI using the priority_spi_<INTID> Register.
3. The processors that receive the SPI using the targets_spi Register.
4. Enable the SPI by using the spi_enable Register.
SPI initialization
Select CPU Interface [0]
Can the processor operate in Secure state?
No
Yes
A processor in Non-secure state cannot program its interrupt security registers and must get a secure processor to program the registers. Prior to the Secure processor writing to the registers, it must set the state of the enable_d<n> and match_d<n> signals so that the writes appear to originate from the Non-secure processor.
The processor in Secure state sets which INTID [31:0] are Non-secure by writing to the sgi_security_if<n> and ppi_security_if<n> Registers.
Select the next CPU Interface
SGI and PPI initialization
For INTID [31:0], program the following parameters:
1. The priority of the interrupts using the priority_sgi_<INTID>_if<n> and priority_ppi_<INTID>_if<n> Registers.
2. Enable the PPIs by using the ppi_enable_if<n> Register.
Are INTID [31:0] configured for each CPU Interface?
No
Yes
Sheet 2

Figure 3-23 Initialization process for GICs that support the Security Extensions, sheet 1 of 2

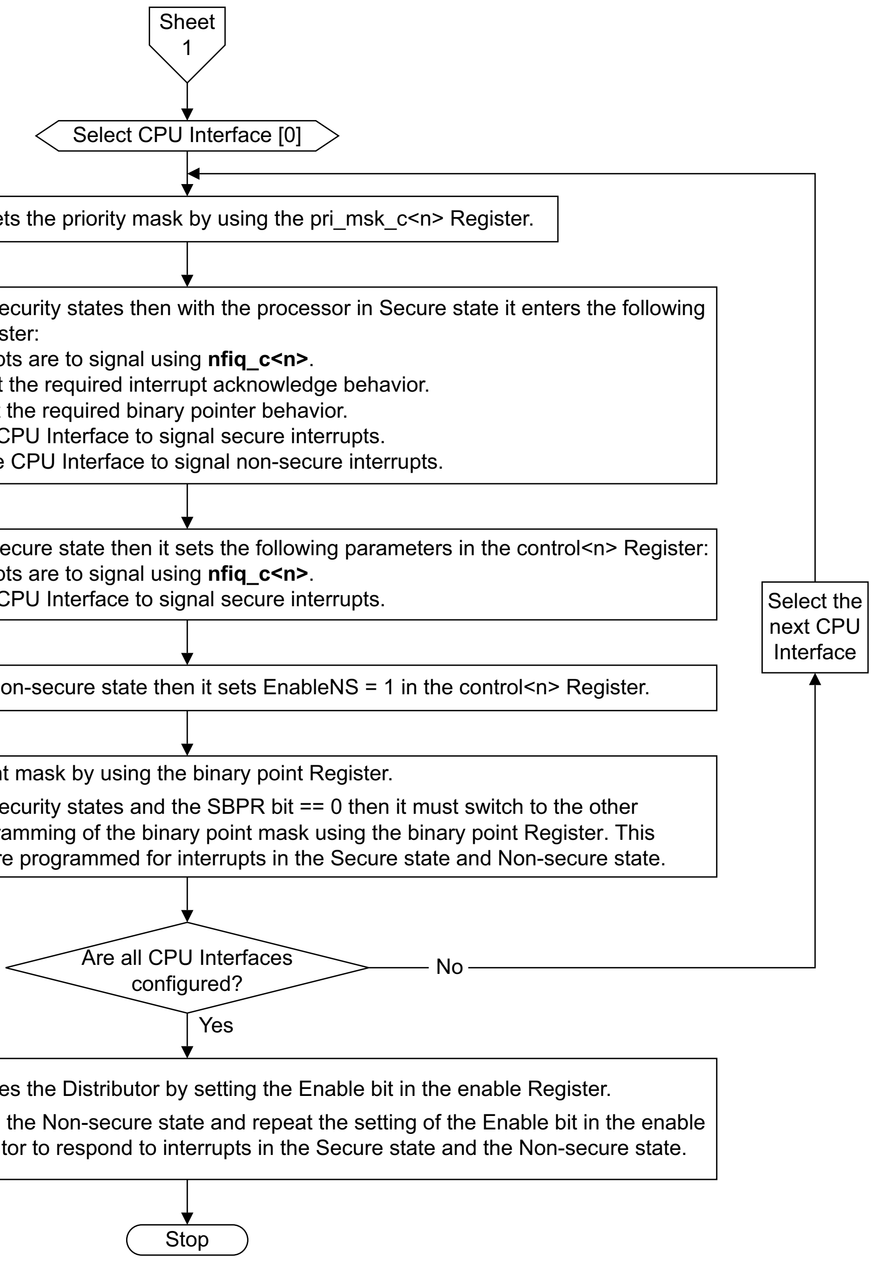

Sheet 1
Select CPU Interface [0]
The processor sets the priority mask by using the pri_msk_c<n> Register.
If the processor operates in both security states then with the processor in Secure state it enters the following parameters in the control<n> Register:
1. Set FIQEn = 1, if secure interrupts are to signal using **nfiq_c<n>**.
2. Program the AckCtl bit, to select the required interrupt acknowledge behavior.
3. Program the SBPR bit, to select the required binary pointer behavior.
4. Set EnableS = 1, to enable the CPU Interface to signal secure interrupts.
5. Set EnableNS = 1, to enable the CPU Interface to signal non-secure interrupts.
If the processor only operates in Secure state then it sets the following parameters in the control<n> Register:
1. Set FIQEn = 1, if secure interrupts are to signal using **nfiq_c<n>**.
2. Set EnableS = 1, to enable the CPU Interface to signal secure interrupts.
If the processor only operates in Non-secure state then it sets EnableNS = 1 in the control<n> Register.
The processor sets the binary point mask by using the binary point Register.
If the processor operates in both security states and the SBPR bit == 0 then it must switch to the other security state and repeat the programming of the binary point mask using the binary point Register. This ensures that binary point masks are programmed for interrupts in the Secure state and Non-secure state.
Are all CPU Interfaces configured?
No
Select the next CPU Interface
Yes
A processor in Secure State enables the Distributor by setting the Enable bit in the enable Register.
The processor must then switch to the Non-secure state and repeat the setting of the Enable bit in the enable Register. This enables the Distributor to respond to interrupts in the Secure state and the Non-secure state.
Stop

Figure 3-24 Initialization process for GICs that support the Security Extensions, sheet 2 of 2

3.4.3 Risk of *Denial of Service* (DoS) when the GIC supports the Security Extensions

It is possible to program the GIC so that a Pending non-secure interrupt prevents a CPU Interface from issuing secure interrupts to a processor as Example 3-1 shows.

Example 3-1 Denial of service

---

1. A processor in Secure state programs a non-secure interrupt to have a priority level that is higher than all the secure interrupts.
2. A processor in Secure state programs the EnableNS bit in the *CPU Interface Control Register (ICCICR)* on page 3-31 so that it prevents the CPU Interface from issuing non-secure interrupts.
3. When the non-secure interrupt from step 1 moves to the Pending state then it pre-empts the current running secure interrupt.
4. Because step 2 disables the CPU Interface from issuing non-secure interrupts to the processor then all secure interrupts are subject to a DoS.

You can avoid creating a possible DoS by programming secure interrupts to always have a higher priority than non-secure interrupts.

Chapter 4
Programmers Model for Test

This chapter describes the registers for functional verification and integration testing. It contains the following sections:

4.1 About the programmers model for test

The following information applies to the GIC registers:

- The base address of the GIC is not fixed, and can be different for any particular system implementation. The offset of each register from the base address is fixed.
- Do not attempt to access reserved or unused address locations. Attempting to access these location can result in Unpredictable behavior of the GIC.
- Unless otherwise stated in the accompanying text:
  - do not modify undefined register bits
  - ignore undefined register bits on reads
  - all register bits are reset to a logic 0 by a system or power-on reset.
- Accesses can be byte, halfword, or word.
- The GIC only supports data in little-endian format.
- The Type column in Table 4-1 on page 4-3 and Table 4-5 on page 4-7 describes the access types as follows:

RW	Read and write.
RO	Read only.
WO	Write only.

4.2 Distributor integration test registers

Table 4-1 lists the integration test registers that the Distributor provides.

Table 4-1 Distributor test register summary

Offset	Name	Type	Reset	Width	Description
`0xDD0`	-	-	-	-	Reserved
`0xDD4`	legacy_int<n>	RO	`0x00000000`	32	*Legacy Interrupt Registers* on page 4-4
`0xDD8-0xDDC`	-	-	-	-	Reserved
`0xDE0`	match_d<n>	RO	`0x00000000`	32	*Match Register* on page 4-5
`0xDE4`	enable_d<n>	RO	`0x00000000`	32	*Enable Register* on page 4-6
`0xDE8-0xEFC`	-	-	-	-	Reserved

4.2.1 Legacy Interrupt Registers

The legacy_int<n> Register characteristics are:

Purpose Enables an external AMBA master to access the status of the:
- **legacy_nirq_c<n>** and **legacy_nfiq_c<n>** inputs for CPU Interface <n>
- **cfgsdisable** tie-off signal.

Usage constraints Only accessible to processors in Secure state.

Configurations This register is available in all configurations of the GIC. The Distributor provides a legacy_int<n> Register for each CPU Interface that the GIC contains.

Attributes See the register summary in Table 4-1 on page 4-3.

Figure 4-1 shows the legacy_int<n> Register bit assignments.

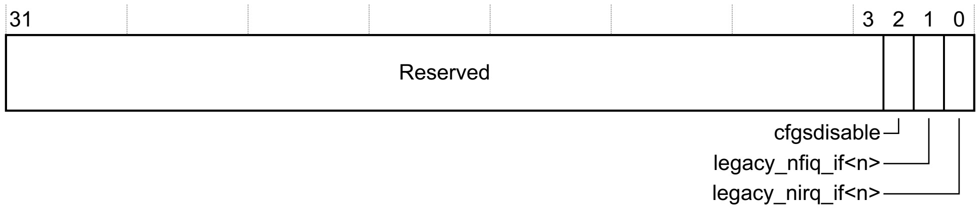


Figure 4-1 legacy_int<n> Register bit assignments

Table 4-2 shows the legacy_int<n> Register bit assignments.

Table 4-2 legacy_int<n> Register bit assignments

Bits	Name	Function
[31:2]	-	Reserved.
[2]	cfgsdisable	Returns the status of the **cfgsdisable** tie-off signal: 0 = **cfgsdisable** is LOW 1 = **cfgsdisable** is HIGH.
[1]	legacy_nfiq_if<n>	Returns the status of the legacy FIQ input signal for CPU Interface <n>: 0 = **legacy_nfiq_c<n>**[a] is LOW, or the GIC does not provide **legacy_nfiq_c<n>** 1 = **legacy_nfiq_c<n>**[a] is HIGH.
[0]	legacy_nirq_if<n>	Returns the status of the legacy IRQ input signal for CPU Interface <n>: 0 = **legacy_nirq_c<n>**[a] is LOW, or the GIC does not provide **legacy_nirq_c<n>** 1 = **legacy_nirq_c<n>**[a] is HIGH.

a. Where <n> is a number, from 0 to 7, that identifies one of the CPU Interfaces.

4.2.2 Match Register

The match_d<n> Register characteristics are:

Purpose	Returns the status of the **match_d<n>[D_ID_WIDTH–1:0]** tie-off signals for CPU Interface <n>.
Usage constraints	Only accessible to processors in Secure state.
Configurations	This register is only available if the GIC contains two or more CPU Interfaces.
Attributes	See the register summary in Table 4-1 on page 4-3.

The Distributor provides a match_d<n> Register for each CPU Interface that the GIC contains. Figure 4-2 shows the match_d<n> Register bit assignments.

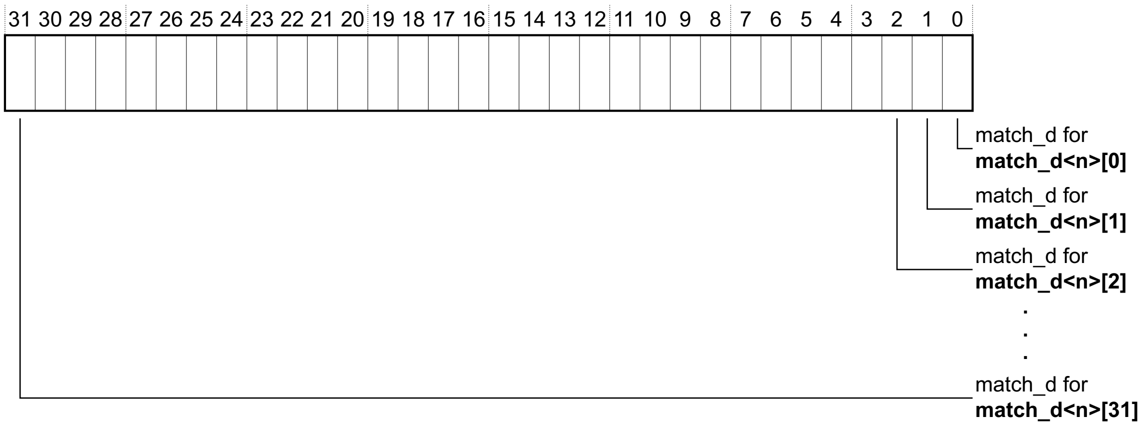

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
match_d for **match_d<n>[0]**
match_d for **match_d<n>[1]**
match_d for **match_d<n>[2]**
.
.
.
match_d for **match_d<n>[31]**

Figure 4-2 match_d<n> Register bit assignments

Table 4-3 shows the match_d<n> Register bit assignments.

Table 4-3 match_d<n> Register bit assignments

Bits	Name	Function
[31:0]	match_d	Returns the status of the **match_d<n>[31:0]** inputs on the Distributor: **Bit [*X*] = 0** **match_d<n>[x]** is LOW **Bit [*X*] = 1** **match_d<n>[x]** is HIGH. Where <n> is a number, from 0 to 7, that identifies one of the CPU Interfaces. **Note** If **D_ID_WIDTH** is configured to be less than 32 then the Distributor returns 0 for each **match_d<n>** signal that is not available.

4.2.3 Enable Register

The enable_d<n> Register characteristics are:

Purpose	Returns the status of the **enable_d<n>[D_ID_WIDTH–1:0]** tie-off signals for CPU Interface <n>.
Usage constraints	Only accessible to processors in Secure state.
Configurations	This register is only available if the GIC contains two or more CPU Interfaces.
Attributes	See the register summary in Table 4-1 on page 4-3.

The Distributor provides an enable_d<n> Register for each CPU Interface that the GIC contains. Figure 4-3 shows the enable_d<n> Register bit assignments.

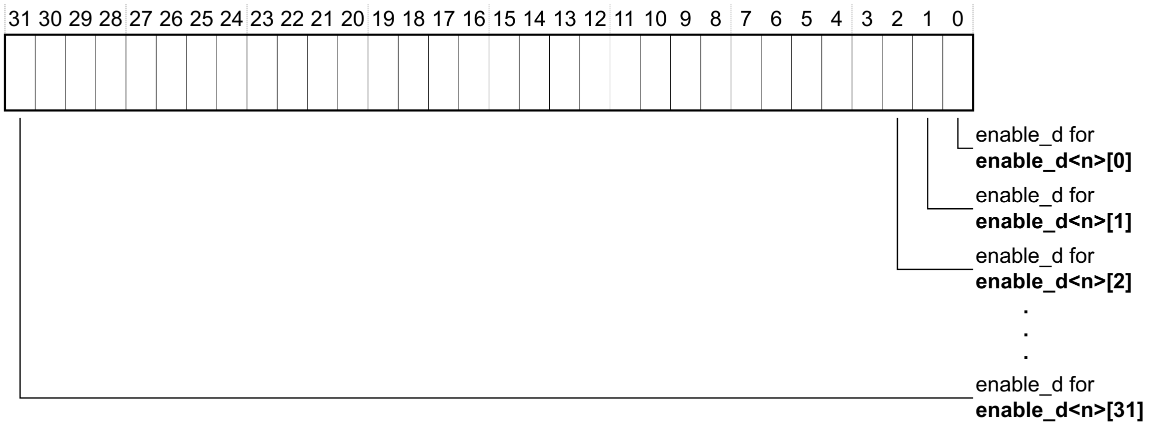

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
enable_d for **enable_d<n>[0]**
enable_d for **enable_d<n>[1]**
enable_d for **enable_d<n>[2]**
.
.
.
enable_d for **enable_d<n>[31]**

Figure 4-3 enable_d<n> Register bit assignments in the Distributor

Table 4-4 shows the enable_d<n> Register bit assignments.

Table 4-4 enable_d<n> Register bit assignments in the Distributor

Bits	Name	Function
[31:0]	enable_d	Returns the status of the **enable_d<n>[31:0]** inputs on the Distributor: **Bit [*X*] = 0** **enable_d<n>[x]** is LOW **Bit [*X*] = 1** **enable_d<n>[x]** is HIGH. Where <n> is a number, from 0 to 7, that identifies one of the CPU Interfaces. ——— **Note** ——— If **D_ID_WIDTH** is configured to be less than 32 then the Distributor returns 0 for each **enable_d<n>** signal that is not available.

4.3 CPU Interface integration test registers

Table 4-5 lists the integration test registers that a CPU Interface provides.

Table 4-5 CPU Interface test register summary

Offset	Name	Type	Reset	Width	Description
`0x040`	integ_en_c<n>	RW	-	32	*Integration Test Enable Register*
`0x044`	interrupt_out<n>	RW	-	32	*Interrupt Output Register* on page 4-8
`0x048-0x04C`	-	-	-	-	Reserved
`0x050`	match_c<n>	RO	`0x00000000`	32	*Match Register* on page 4-9
`0x054`	enable_c<n>	RO	`0x00000000`	32	*Enable Register* on page 4-10

4.3.1 Integration Test Enable Register

The integ_en_c<n> Register characteristics are:

Purpose	Enables the integration test logic to modify the status of the **nfiq_c<n>** and **nirq_c<n>** signals.
Usage constraints	Only accessible to processors in Secure state.
Configurations	This register is available in all configurations of the GIC.
Attributes	See the register summary in Table 4-5.

Figure 4-4 shows the integ_en_c<n> Register bit assignments.

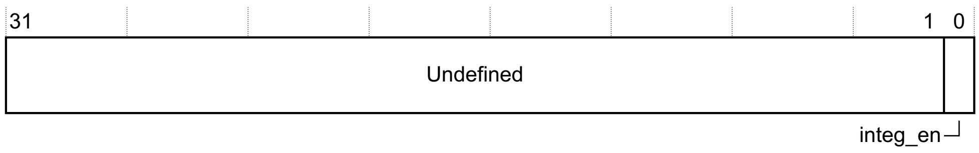


Figure 4-4 integ_en_c<n> Register bit assignments

Table 4-6 shows the integ_en_c<n> Register bit assignments.

Table 4-6 integ_en_c<n> Register bit assignments

Bits	Name	Function
[31:1]	-	Undefined. Write as zero.
[0]	integ_en	Enables the integration test logic: 0 = disables the integration test logic 1 = integration test logic controls the status of the following output signals: • **nfiq_c<n>** • **nirq_c<n>**. **Note** Where <n> is a number, from 0 to 7, that identifies one of the CPU Interfaces.

4.3.2 Interrupt Output Register

The interrupt_out<n> Register characteristics are:

Purpose	Enables a processor to read, or set, the status of the **nirq_c<n>** and **nfiq_c<n>** outputs for CPU Interface <n>.
Usage constraints	Only accessible to processors in Secure state.
Configurations	This register is available in all configurations of the GIC.
Attributes	See the register summary in Table 4-5 on page 4-7.

Figure 4-5 shows the interrupt_out<n> Register bit assignments.

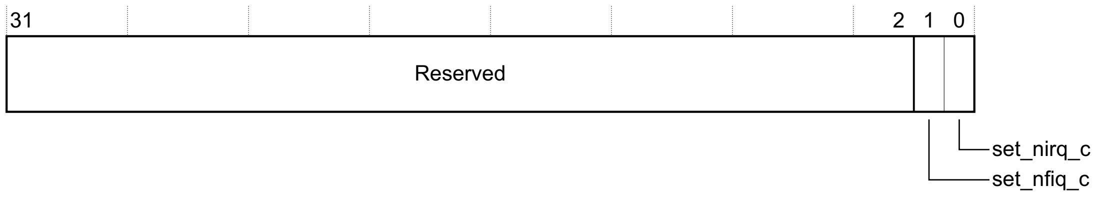


Figure 4-5 interrupt_out<n> Register bit assignments

Table 4-7 shows the interrupt_out<n> Register bit assignments.

Table 4-7 interrupt_out<n> Register bit assignments

Bits	Name	Function
[31:2]	-	Reserved.
[1]	set_nfiq_c	For CPU Interface <n>, if the GIC supports the Security Extensions then reads return the status of **nfiq_c<n>** and writes set the status of **nfiq_c<n>**: 0 = **nfiq_c<n>**[a] is LOW 1 = **nfiq_c<n>**[a] is HIGH. **Note** If the GIC does not support the Security Extensions then it returns zero when read and it ignores writes.
[0]	set_nirq_c	For CPU Interface <n>, reads return the status of **nirq_c<n>** and writes set the status of **nirq_c<n>**: 0 = **nirq_c<n>**[a] is LOW 1 = **nirq_c<n>**[a] is HIGH.

a. Where <n> is a number, from 0 to 7, that identifies one of the CPU Interfaces.

4.3.3 Match Register

The match_c<n> Register characteristics are:

Purpose	Returns the status of the **match_c<n>[C_ID_WIDTH–1:0]** tie-off signals for CPU Interface <n>.
Usage constraints	Only accessible to processors in Secure state.
Configurations	This register is only available if the GIC contains two or more CPU Interfaces.
Attributes	See the register summary in Table 4-5 on page 4-7.

Figure 4-6 shows the match_c<n> Register bit assignments.

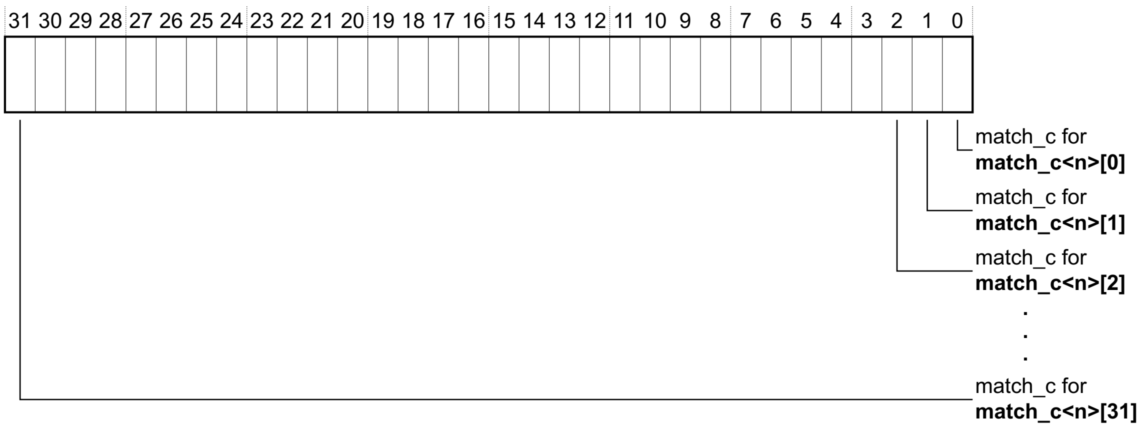

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
match_c for **match_c<n>[0]**
match_c for **match_c<n>[1]**
match_c for **match_c<n>[2]**
match_c for **match_c<n>[31]**

Figure 4-6 match_c<n> Register bit assignments in a CPU Interface

Table 4-8 shows the match_c<n> Register bit assignments.

Table 4-8 match_c<n> Register bit assignments in a CPU Interface

Bits	Name	Function
[31:0]	match_c	Returns the status of the **match_c<n>[31:0]** inputs for CPU Interface <n>: **Bit [*X*] = 0** **match_c<n>[x]** is LOW **Bit [*X*] = 1** **match_c<n>[x]** is HIGH. Where <n> is a number, from 0 to 7, that identifies one of the CPU Interfaces. ——— **Note** ——— If **C_ID_WIDTH** is configured to be less than 32 then a CPU Interface returns 0 for each **match_c<n>** signal that is not available. ———————

4.3.4 Enable Register

The enable_c<n> Register characteristics are:

Purpose	Returns the status of the **enable_c<n>[C_ID_WIDTH–1:0]** tie-off signals for CPU Interface <n>.
Usage constraints	Only accessible to processors in Secure state.
Configurations	This register is only available if the GIC contains two or more CPU Interfaces.
Attributes	See the register summary in Table 4-5 on page 4-7.

Figure 4-7 shows the enable_c<n> Register bit assignments.

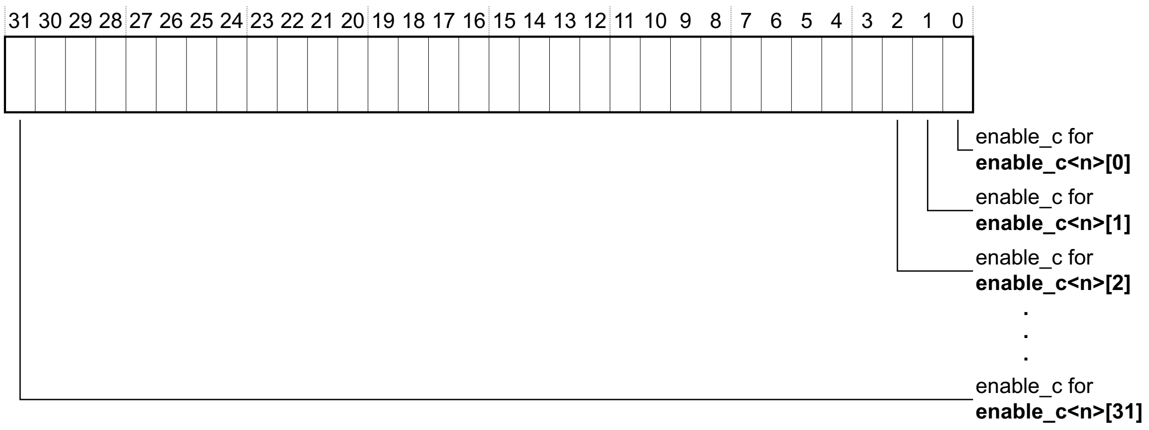

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
enable_c for **enable_c<n>[0]**
enable_c for **enable_c<n>[1]**
enable_c for **enable_c<n>[2]**
enable_c for **enable_c<n>[31]**

Figure 4-7 enable_c<n> Register bit assignments in a CPU Interface

Table 4-9 shows the enable_c<n> Register bit assignments.

Table 4-9 enable_c<n> Register bit assignments in a CPU Interface

Bits	Name	Function
[31:0]	enable_c	Returns the status of the **enable_c<n>[31:0]** inputs for CPU Interface <n>: **Bit [*X*] = 0** **enable_c<n>[x]** is LOW **Bit [*X*] = 1** **enable_c<n>[x]** is HIGH. Where <n> is a number, from 0 to 7, that identifies one of the CPU Interfaces. **Note** If **C_ID_WIDTH** is configured to be less than 32 then a CPU Interface returns 0 for each **enable_c<n>** signal that is not available.

Appendix A
Signal Descriptions

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

A.1 Clock and reset signals

Table A-1 shows the clock and reset signals.

Table A-1 Clock and reset signals

Signal	Type	Source	Description
gclk	Input	Clock source	Clock for the GIC.
gresetn	Input	Reset source	Reset for the GIC. This signal is active LOW.

A.2 AXI slave interface signals

The following sections describe the AXI slave interface signals:

- *Write address (AXI-AW) channel signals*
- *Write data (AXI-W) channel signals* on page A-4
- *Write response (AXI-B) channel signals* on page A-5
- *Read address (AXI-AR) channel signals* on page A-5
- *Read data (AXI-R) channel signals* on page A-6.

A.2.1 Write address (AXI-AW) channel signals

Table A-2 shows the AXI write address signals for the Distributor.

Table A-2 AXI-AW signals for the Distributor

Signal	AMBA equivalent[a]
awaddr_d[31:0]	**AWADDR[31:0]**
awburst_d[1:0]	**AWBURST[1:0]**
awcache_d[3:0][b]	**AWCACHE[3:0]**
awid_d[D_ID_WIDTH−1:0][c]	**AWID[3:0]**
awlen_d[3:0]	**AWLEN[3:0]**
awlock_d[1:0][b]	**AWLOCK[1:0]**
awprot_d[2:0]	**AWPROT[2:0]**
awready_d	**AWREADY**
awsize_d[2:0]	**AWSIZE[2:0]**
awvalid_d	**AWVALID**

a. See the *AMBA AXI Protocol v1.0 Specification* for a description of these signals.
b. The GIC ignores any information that it receives on these signals.
c. The value of **D_ID_WIDTH** is set during configuration of the GIC.

Table A-3 shows the AXI write address signals for a CPU Interface.

Table A-3 AXI-AW signals for a CPU Interface

Signal	AMBA equivalent[a]
awaddr_c[31:0]	**AWADDR[31:0]**
awburst_c[1:0]	**AWBURST[1:0]**
awcache_c[3:0][b]	**AWCACHE[3:0]**
awid_c[C_ID_WIDTH−1:0][c]	**AWID[3:0]**
awlen_c[3:0]	**AWLEN[3:0]**
awlock_c[1:0][b]	**AWLOCK[1:0]**
awprot_c[2:0]	**AWPROT[2:0]**

Table A-3 AXI-AW signals for a CPU Interface (continued)

Signal	AMBA equivalent[a]
awready_c	**AWREADY**
awsize_c[2:0]	**AWSIZE[2:0]**
awvalid_c	**AWVALID**

a. See the *AMBA AXI Protocol v1.0 Specification* for a description of these signals.
b. The GIC ignores any information that it receives on these signals.
c. The value of **C_ID_WIDTH** is set during configuration of the GIC.

A.2.2 Write data (AXI-W) channel signals

Table A-4 shows the AXI write data signals for the Distributor.

Table A-4 AXI-W signals for the Distributor

Signal	AMBA equivalent[a]
wdata_d[31:0]	**WDATA[31:0]**
wid_d[D_ID_WIDTH−1:0][b]	**WID[3:0]**
wlast_d	**WLAST**
wready_d	**WREADY**
wstrb_d[3:0]	**WSTRB[3:0]**
wvalid_d	**WVALID**

a. See the *AMBA AXI Protocol v1.0 Specification* for a description of these signals.
b. The value of **D_ID_WIDTH** is set during configuration of the GIC.

Table A-5 shows the AXI write data signals for a CPU Interface.

Table A-5 AXI-W signals for a CPU Interface

Signal	AMBA equivalent[a]
wdata_c[31:0]	**WDATA[31:0]**
wid_c[C_ID_WIDTH−1:0][b]	**WID[3:0]**
wlast_c	**WLAST**
wready_c	**WREADY**
wstrb_c[3:0]	**WSTRB[3:0]**
wvalid_c	**WVALID**

a. See the *AMBA AXI Protocol v1.0 Specification* for a description of these signals.
b. The value of **C_ID_WIDTH** is set during configuration of the GIC.

A.2.3 Write response (AXI-B) channel signals

Table A-6 shows the AXI write response signals for the Distributor.

Table A-6 AXI-B signals for the Distributor

Signal	AMBA equivalent[a]
bid_d[D_ID_WIDTH−1:0][b]	**BID[3:0]**
bready_d	**BREADY**
bresp_d[1:0]	**BRESP[1:0]**
bvalid_d	**BVALID**

a. See the *AMBA AXI Protocol v1.0 Specification* for a description of these signals.
b. The value of **D_ID_WIDTH** is set during configuration of the GIC.

Table A-7 shows the AXI write response signals for a CPU Interface.

Table A-7 AXI-B signals for a CPU Interface

Signal	AMBA equivalent[a]
bid_c[C_ID_WIDTH−1:0][b]	**BID[3:0]**
bready_c	**BREADY**
bresp_c[1:0]	**BRESP[1:0]**
bvalid_c	**BVALID**

a. See the *AMBA AXI Protocol v1.0 Specification* for a description of these signals.
b. The value of **C_ID_WIDTH** is set during configuration of the GIC.

A.2.4 Read address (AXI-AR) channel signals

Table A-8 shows the AXI read address signals for the Distributor.

Table A-8 AXI-AR signals for the Distributor

Signal	AMBA equivalent[a]
araddr_d[31:0]	**ARADDR[31:0]**
arburst_d[1:0]	**ARBURST[1:0]**
arcache_d[3:0][b]	**ARCACHE[3:0]**
arid_d[D_ID_WIDTH−1:0][c]	**ARID[3:0]**
arlen_d[3:0]	**ARLEN[3:0]**
arlock_d[1:0][b]	**ARLOCK[1:0]**
arprot_d[2:0]	**ARPROT[2:0]**
arready_d	**ARREADY**
arsize_d[2:0]	**ARSIZE[2:0]**
arvalid_d	**ARVALID**

a. See the *AMBA AXI Protocol v1.0 Specification* for a description of these signals.
b. The GIC ignores any information that it receives on these signals.
c. The value of **D_ID_WIDTH** is set during configuration of the GIC.

Table A-9 shows the AXI read address signals for a CPU Interface.

Table A-9 AXI-AR signals for a CPU Interface

Signal	AMBA equivalent[a]
araddr_c[31:0]	**ARADDR[31:0]**
arburst_c[1:0]	**ARBURST[1:0]**
arcache_c[3:0][b]	**ARCACHE[3:0]**
arid_c[C_ID_WIDTH−1:0][c]	**ARID[3:0]**
arlen_c[3:0]	**ARLEN[3:0]**
arlock_c[1:0][b]	**ARLOCK[1:0]**
arprot_c[2:0]	**ARPROT[2:0]**
arready_c	**ARREADY**
arsize_c[2:0]	**ARSIZE[2:0]**
arvalid_c	**ARVALID**

a. See the *AMBA AXI Protocol v1.0 Specification* for a description of these signals.
b. The GIC ignores any information that it receives on these signals.
c. The value of **C_ID_WIDTH** is set during configuration of the GIC.

A.2.5 Read data (AXI-R) channel signals

Table A-10 shows the AXI read data signals for the Distributor.

Table A-10 AXI-R signals for the Distributor

Signal	AMBA equivalent[a]
rdata_d[31:0]	**RDATA[31:0]**
rid_d[D_ID_WIDTH−1:0][b]	**RID[3:0]**
rlast_d	**RLAST**
rready_d	**RREADY**
rresp_d[1:0]	**RRESP[1:0]**
rvalid_d	**RVALID**

a. See the *AMBA AXI Protocol v1.0 Specification* for a description of these signals.
b. The value of **D_ID_WIDTH** is set during configuration of the GIC.

Table A-11 shows the AXI read data signals for a CPU Interface.

Table A-11 AXI-R signals for a CPU Interface

Signal	AMBA equivalent[a]
rdata_c[31:0]	**RDATA[31:0]**
rid_c[C_ID_WIDTH−1:0][b]	**RID[3:0]**
rlast_c	**RLAST**
rready_c	**RREADY**
rresp_c[1:0]	**RRESP[1:0]**
rvalid_c	**RVALID**

a. See the *AMBA AXI Protocol v1.0 Specification* for a description of these signals.
b. The value of **C_ID_WIDTH** is set during configuration of the GIC.

A.3 AHB-Lite signals

The GIC provides two AHB-Lite slave interfaces when you configure it to use AHB-Lite interfaces.

The following sections describe the AHB-Lite slave interfaces:

- *AHB-Lite slave interface for the Distributor*
- *AHB-Lite slave interface for the CPU Interface* on page A-9.

A.3.1 AHB-Lite slave interface for the Distributor

Table A-12 shows the AHB-Lite slave interface signals for the Distributor.

Table A-12 AHB-Lite slave interface signals for the Distributor

Signal	AMBA equivalent[a]
haddr_d[31:0]	**HADDR[31:0]**
hrdata_d[31:0]	**HRDATA[31:0]**
hready_d	**HREADY**
hreadyout_d	**HREADYOUT**
hresp_d[1:0]	**HRESP[1:0]**
hsel_d	**HSELx**
hsize_d[2:0]	**HSIZE[2:0]**
htrans_d[1:0]	**HTRANS[1:0]**
hwdata_d[31:0]	**HWDATA[31:0]**
hwrite_d	**HWRITE**

a. See the *AMBA 3 AHB-Lite Protocol v1.0 Specification* for a description of these signals.

A.3.2 AHB-Lite slave interface for the CPU Interface

Table A-13 shows the AHB-Lite slave interface signals for the CPU Interface.

Table A-13 AHB-Lite slave interface signals for the CPU Interface

Signal	AMBA equivalent[a]
haddr_c[31:0]	**HADDR[31:0]**
hrdata_c[31:0]	**HRDATA[31:0]**
hready_c	**HREADY**
hreadyout_c	**HREADYOUT**
hresp_c[1:0]	**HRESP[1:0]**
hsel_c	**HSELx**
hsize_c[2:0]	**HSIZE[2:0]**
htrans_c[1:0]	**HTRANS[1:0]**
hwdata_c[31:0]	**HWDATA[31:0]**
hwrite_c	**HWRITE**

a. See the *AMBA 3 AHB-Lite Protocol v1.0 Specification* for a description of these signals.

A.4 Interrupt signals

Table A-14 shows the interrupt signals.

Table A-14 Interrupt signals

Signal	Type	Source or destination	Description
nfiq_c<n>[a]	Output	Processor	FIQ interrupt for the processor that connects to CPU Interface <n>. The GIC provides this signal when it is configured to support the Security Extensions.
nirq_c<n>[a]	Output	Processor	IRQ interrupt for the processor that connects to CPU Interface <n>.
legacy_nfiq_c<n>[a]	Input	Peripheral or legacy interrupt controller	A legacy FIQ interrupt for CPU Interface <n>. This signal is a configuration option that is only available when the GIC is configured to support the Security Extensions.
legacy_nirq_c<n>[a]	Input	Peripheral or legacy interrupt controller	A legacy IRQ interrupt for CPU Interface <n>. You can specify if the GIC provides this signal when you configure the GIC.
ppi_c<n>[x:0][a b]	Input	Peripheral	Private peripheral interrupt inputs for GICs that are configured to support two or more processors. The GIC can provide up to 16 PPIs for each CPU Interface that it contains.
spi[x:0][c]	Input	Peripheral	Shared peripheral interrupt inputs. The GIC can provide from one to 988 SPIs.

a. Where <n> represents the number of a CPU Interface, and can vary from 0 to 7, depending on the configuration of the GIC.
b. Where *x* represents the number of PPIs minus one, that you configure the GIC to contain.
c. Where *x* represents the number of SPIs minus one, that you configure the GIC to contain.

A.5 Miscellaneous signals

The following sections describe the miscellaneous signals:

- *cfgsdisable*
- *Enable and match*.

A.5.1 cfgsdisable

Table A-15 shows the **cfgsdisable** signal.

Table A-15 cfgsdisable signal

Signal	Type	Source	Description
cfgsdisable	Input	External	When this signal is HIGH, it enhances the security of the GIC by preventing write accesses to security-critical configuration registers. See the *ARM Generic Interrupt Controller Architecture Specification*.

A.5.2 Enable and match

If the GIC is configured to contain two or more CPU Interfaces then it provides the following tie-off signals:

- *enable_d and match_d signals on the Distributor*
- *enable_c and match_c signals for a CPU Interface* on page A-12.

enable_d and match_d signals on the Distributor

Table A-16 shows the **enable_d<n>** and **match_d<n>** tie-off signals on the Distributor.

Table A-16 enable_d<n> and match_d<n> signals

Signal[a]	Type	Source	Description
enable_d<n>[D_ID_WIDTH–1:0]	Input	External	When you attempt to read a banked register in the Distributor then it performs a logical AND of **enable_d<n>** with **arid_d** and it compares the result with **match_d<n>**. If the comparison is successful then it grants access to the register that **araddr_d** addresses, otherwise it ignores the AXI read transaction. When you attempt to write to a banked register in the Distributor then it performs a logical AND of **enable_d<n>** with **awid_d** and it compares the result with **match_d<n>**. If the comparison is successful then it grants access to the register that **awaddr_d** addresses, otherwise it ignores the AXI write transaction. See *enable and match signals* on page 2-7 for more information.
match_d<n>[D_ID_WIDTH–1:0]	Input	External	These signals initialize the value of the AXI ID tag that must be used, to access the banked CPU Interface *n* registers. that are located in the Distributor. See *enable and match signals* on page 2-7 for more information.

a. Where:

<n>	Is a number, from 0 to 7, that identifies one of the CPU Interfaces.
D_ID_WIDTH	Is the width of the AXI ID tag bus on the Distributor and is set during configuration of the GIC.

enable_c and **match_c** signals for a CPU Interface

Table A-17 shows the **enable_c<n>** and **match_c<n>** tie-off signals that the GIC provides, for each of the CPU Interfaces that it contains.

Table A-17 enable_c<n> and match_c<n> signals

Signal[a]	Type	Source	Description
enable_c<n>[C_ID_WIDTH–1:0]	Input	External	When you attempt to read a register in a CPU Interface then the GIC performs a logical AND of **enable_c<n>** with **arid_c** and it compares the result with **match_c<n>**. If the comparison is successful then the GIC grants access to CPU Interface *n* and the register that **araddr_c** addresses, otherwise it ignores the AXI read transaction. When you attempt to write to a register in a CPU Interface then the GIC performs a logical AND of **enable_d<n>** with **awid_c** and it compares the result with **match_d<n>**. If the comparison is successful then the GIC grants access to CPU Interface *n* and the register that **awaddr_d** addresses, otherwise it ignores the AXI write transaction. See *enable and match signals* on page 2-7 for more information.
match_c<n>[C_ID_WIDTH–1:0]	Input	External	These signals initialize the system identification AXI ID tag for CPU Interface <n>. See *enable and match signals* on page 2-7 for more information.

a. Where:

<n>	Is a number, from 0 to 7, that identifies the CPU Interface.
C_ID_WIDTH	Is the width of the AXI ID tag bus for all the CPU Interfaces and is set during configuration of the GIC.

Appendix B
Interrupt Signaling

This appendix describes how the GIC signals interrupts to a processor. It contains the following sections:

B.1 Signaling interrupts when the GIC supports a single security state

If you configure a GIC to operate only in Secure state then the CPU Interface signals the secure interrupts to the processor using **nirq_c<n>**.

The CPU Interface signals an interrupt by setting **nirq_c<n>** LOW and the signal remains LOW until a processor acknowledges the interrupt by reading the Interrupt Acknowledge Register as Figure B-1 shows.

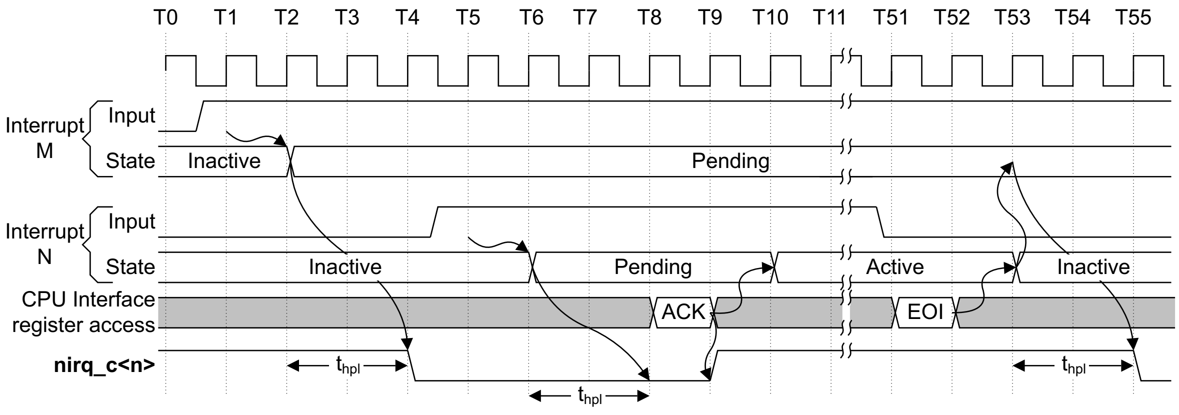


Figure B-1 Interrupt signaling using nirq_c

In Figure B-1, at time:

T1 The Distributor detects the assertion of interrupt M.

T2 The Distributor sets interrupt M to Pending.

T4 The CPU Interface asserts **nirq_c<n>**.

Note

- The assertion of **nirq_c<n>** occurs between one to 10 **gclk** cycles after interrupt M becomes Pending. This time duration, t_{hpl}, depends on the latency of the highest priority logic in the Distributor. t_{hpl} is set during configuration of the GIC.
- In Figure B-1, t_{hpl} = 2 clocks.
- During time t_{hpl}, if any interrupt becomes Pending then the Distributor must determine which interrupt has the highest priority. Therefore t_{hpl} now commences when that interrupt became Pending. For example, at T3 if an interrupt becomes Pending, then t_{hpl} starts at T3 and completes at T5.

T5 The Distributor detects the assertion of a higher priority interrupt, N, and interrupt M is pre-empted on the next clock cycle.

T6 The Distributor sets interrupt N to Pending. The CPU Interface updates the ACKINTID field in the Interrupt Acknowledge Register to contain the INTID value for interrupt N.

T8 t_{hpl} clocks after interrupt N becomes Pending then the CPU Interface asserts **nirq_c<n>**. The state of **nirq_c<n>** is unchanged because **nirq_c<n>** was asserted at T4.

T9 The processor reads the Interrupt Acknowledge Register and the CPU Interface deasserts **nirq_c<n>**.

T10	The Distributor sets interrupt N to Active and updates the Active Status Register.
T10 - T51	The processor services the interrupt. The peripheral deasserts interrupt N.
T52	The processor writes to the End of Interrupt Register with the INTID of interrupt N.
T53	The Distributor sets interrupt N to Inactive and updates the Active Status Register. The Distributor signals to the CPU Interface that interrupt M is now the highest priority Pending interrupt.
T55	When interrupt N becomes Inactive then t_{hpl} clocks after interrupt M becomes the highest priority Pending interrupt, the CPU Interface asserts **nirq_c<n>**.

B.2 Signaling interrupts when the GIC supports the Security Extensions

If you configure the GIC to support the Security Extensions then by using the control<n> Register, in the CPU Interface, you can program a CPU Interface to either:

- signal non-secure and secure interrupts using **nirq_c<n>**
- signal non-secure using **nirq_c<n>** and signal secure interrupts using **nfiq_c<n>**.

If you program a CPU Interface to signal interrupts using **nirq_c<n>**, irrespective of the security state of an interrupt, then the CPU Interface signaling is as *Signaling interrupts when the GIC supports a single security state* on page B-2 describes.

If you program a CPU Interface to use **nirq_c<n>** and **nfiq_c<n>** then the CPU Interface signals an interrupt by setting the appropriate signal LOW, depending on the security state of the interrupt. **nfiq_c<n>** remains LOW until a processor acknowledges the interrupt by reading the Interrupt Acknowledge Register. **nirq_c<n>** remains LOW until either:

- a processor acknowledges the interrupt by reading the Interrupt Acknowledge Register
- the CPU Interface asserts **nfiq_c<n>** as Figure B-2 shows.

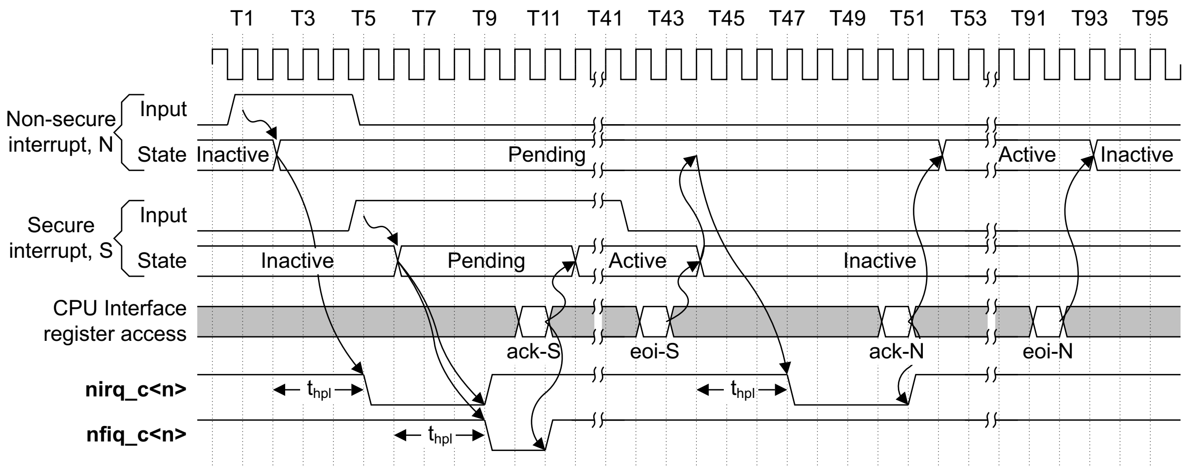


Figure B-2 Interrupt signaling using nirq_c and nfiq_c

In Figure B-2, at time:

T1 The Distributor detects the assertion of a non-secure interrupt, N.

T2 The Distributor sets interrupt N to Pending.

T5 The CPU Interface asserts **nirq_c<n>**.

——— **Note** ———

- The assertion of **nirq_c<n>** occurs between one to 10 **gclk** cycles after interrupt N becomes Pending. This time duration, t_{hpl}, depends on the latency of the highest priority logic in the Distributor. t_{hpl} is set during configuration of the GIC.
- In Figure B-2, t_{hpl} = 3 clocks.
- During time t_{hpl}, if any interrupt becomes Pending then the Distributor must determine which interrupt has the highest priority. Therefore t_{hpl} now commences when that interrupt became Pending. For example, at T3 if an interrupt becomes Pending, then t_{hpl} starts at T3 and completes at T6.

T5 The Distributor detects the assertion of a secure interrupt, S, and interrupt N is pre-empted on the next clock cycle.

T6 The Distributor sets interrupt S to Pending. The CPU Interface updates the ACKINTID field in the Interrupt Acknowledge Register to contain the INTID value for interrupt S.

T9 t_{hpl} clocks after interrupt S becomes Pending, the CPU Interface:

- asserts **nfiq_c<n>**
- deasserts **nirq_c<n>**.

T11 The processor reads the Interrupt Acknowledge Register and the CPU Interface deasserts **nfiq_c<n>**.

T12 The Distributor sets interrupt S to Active and updates the Active Status Register.

T11 - T42 The processor services the secure interrupt. The peripheral deasserts interrupt S.

T43 The processor writes to the End of Interrupt Register with the INTID of interrupt S.

T44 The Distributor sets interrupt S to Inactive and updates the Active Status Register.

The Distributor signals to the CPU Interface that interrupt N is now the highest priority Pending interrupt.

T47 When interrupt S becomes Inactive then t_{hpl} clocks after interrupt N becomes the highest priority Pending interrupt, the CPU Interface asserts **nirq_c<n>**.

T51 The processor reads the Interrupt Acknowledge Register and the CPU Interface deasserts **nirq_c<n>**.

T52 The Distributor sets interrupt N to Active and updates the Active Status Register.

T51 - T91 The processor services the non-secure interrupt.

T92 The processor writes to the End of Interrupt Register with the INTID of interrupt N.

T93 The Distributor sets interrupt N to Inactive and updates the Active Status Register.

Appendix C
Revisions

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

Table C-1 Differences between issue A and issue B

Change	Location	Affects
Updated the short name for the Distributor Control Register	Table 3-1 on page 3-5	r0p0
Updated the short name for the Distributor Implementer Identification Register	*Distributor Implementer Identification Register (ICDIIDR)* on page 3-8	
Updated the value of the architecture field	Table 3-9 on page 3-24	
Updated the short name for the Priority Mask Register	Table 3-18 on page 3-30	
Updated the short name for the CPU Interface Implementer Identification Register	*CPU Interface Implementer Identification Register (ICCIIDR)* on page 3-31	
Updated the bit widths of the **rresp_d** and **rresp_c** signals	• Table A-10 on page A-6 • Table A-11 on page A-7	

Glossary

This glossary describes some of the terms used in technical documents from ARM.

Advanced eXtensible Interface (AXI)

A bus protocol that supports separate address/control and data phases, unaligned data transfers using byte strobes, burst-based transactions with only start address issued, separate read and write data channels to enable low-cost DMA, ability to issue multiple outstanding addresses, out-of-order transaction completion, and easy addition of register stages to provide timing closure.

The AXI protocol also includes optional extensions to cover signaling for low-power operation.

AXI is targeted at high performance, high clock frequency system designs and includes a number of features that make it very suitable for high speed sub-micron interconnect.

Advanced High-performance Bus (AHB)

A bus protocol with a fixed pipeline between address/control and data phases. It only supports a subset of the functionality provided by the AMBA AXI protocol. The full AMBA AHB protocol specification includes a number of features that are not commonly required for master and slave IP developments and ARM Limited recommends only a subset of the protocol is usually used. This subset is defined as the AMBA AHB-Lite protocol.

See also Advanced Microcontroller Bus Architecture and AHB-Lite.

Advanced Microcontroller Bus Architecture (AMBA)

A family of protocol specifications that describe a strategy for the interconnect. AMBA is the ARM open standard for on-chip buses. It is an on-chip bus specification that describes a strategy for the interconnection and management of functional blocks that make up a *System-on-Chip* (SoC). It aids in the development of embedded processors with one or more CPUs or signal processors and multiple peripherals. AMBA complements a reusable design methodology by defining a common backbone for SoC modules.

AHB *See* Advanced High-performance Bus.

AHB-Lite	A subset of the full AMBA AHB protocol specification. It provides all of the basic functions required by the majority of AMBA AHB slave and master designs, particularly when used with a multi-layer AMBA interconnect. In most cases, the extra facilities provided by a full AMBA AHB interface are implemented more efficiently by using an AMBA AXI protocol interface.
Aligned	A data item stored at an address that is divisible by the number of bytes that defines the data size is said to be aligned. Aligned words and halfwords have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore stipulate addresses that are divisible by four and two respectively.
AMBA	*See* Advanced Microcontroller Bus Architecture.

Application Specific Integrated Circuit (ASIC)

An integrated circuit that has been designed to perform a specific application function. It can be custom-built or mass-produced.

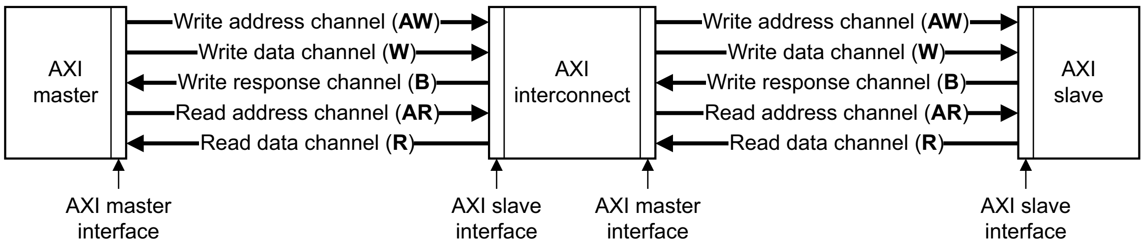
ASIC *See* Application Specific Integrated Circuit.

AXI *See* Advanced eXtensible Interface.

AXI channel order and interfaces

The block diagram shows:

- the order in which AXI channel signals are described
- the master and slave interface conventions for AXI components.



AXI terminology The following AXI terms are general. They apply to both masters and slaves:

Active read transaction

A transaction for which the read address has transferred, but the last read data has not yet transferred.

Active transfer

A transfer for which the **xVALID**[1] handshake has asserted, but for which **xREADY** has not yet asserted.

Active write transaction

A transaction for which the write address or leading write data has transferred, but the write response has not yet transferred.

Completed transfer

A transfer for which the **xVALID**/**xREADY** handshake is complete.

1. The letter **x** in the signal name denotes an AXI channel as follows:

AW	Write address channel.
W	Write data channel.
B	Write response channel.
AR	Read address channel.
R	Read data channel.

Payload	The non-handshake signals in a transfer.
Transaction	An entire burst of transfers, comprising an address, one or more data transfers and a response transfer (writes only).
Transmit	An initiator driving the payload and asserting the relevant **xVALID** signal.
Transfer	A single exchange of information. That is, with one **xVALID**/**xREADY** handshake.

The following AXI terms are master interface attributes. To obtain optimum performance, they must be specified for all components with an AXI master interface:

Combined issuing capability

The maximum number of active transactions that a master interface can generate. It is specified for master interfaces that use combined storage for active write and read transactions. If not specified then it is assumed to be equal to the sum of the write and read issuing capabilities.

Read ID capability

The maximum number of different **ARID** values that a master interface can generate for all active read transactions at any one time.

Read ID width

The number of bits in the **ARID** bus.

Read issuing capability

The maximum number of active read transactions that a master interface can generate.

Write ID capability

The maximum number of different **AWID** values that a master interface can generate for all active write transactions at any one time.

Write ID width

The number of bits in the **AWID** and **WID** buses.

Write interleave capability

The number of active write transactions for which the master interface is capable of transmitting data. This is counted from the earliest transaction.

Write issuing capability

The maximum number of active write transactions that a master interface can generate.

The following AXI terms are slave interface attributes. To obtain optimum performance, they must be specified for all components with an AXI slave interface:

Combined acceptance capability

The maximum number of active transactions that a slave interface can accept. It is specified for slave interfaces that use combined storage for active write and read transactions. If not specified then it is assumed to be equal to the sum of the write and read acceptance capabilities.

Read acceptance capability

The maximum number of active read transactions that a slave interface can accept.

Read data reordering depth

The number of active read transactions for which a slave interface can transmit data. This is counted from the earliest transaction.

Write acceptance capability

The maximum number of active write transactions that a slave interface can accept.

Write interleave depth

The number of active write transactions for which the slave interface can receive data. This is counted from the earliest transaction.

Big-endian

Byte ordering scheme in which bytes of decreasing significance in a data word are stored at increasing addresses in memory.

See also Little-endian and Endianness.

Boundary scan chain

A boundary scan chain is made up of serially-connected devices that implement boundary scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain connected between **TDI** and **TDO**, through which test data is shifted. Processors can contain several shift registers to enable you to access selected parts of the device.

CPSR

See Current Program Status Register

Current Program Status Register (CPSR)

The register that holds the current operating processor status.

Endianness

Byte ordering. The scheme that determines the order that successive bytes of a data word are stored in memory. An aspect of the system's memory mapping.

See also Little-endian and Big-endian

Implementation-defined

The behavior is not architecturally defined, but is defined and documented by individual implementations.

Implementation-specific

The behavior is not architecturally defined, and does not have to be documented by individual implementations. Used when there are a number of implementation options available and the option chosen does not affect software compatibility.

Interrupt handler

A program that control of the processor is passed to when an interrupt occurs.

Interrupt vector

One of a number of fixed addresses in low memory, or in high memory if high vectors are configured, that contains the first instruction of the corresponding interrupt handler.

Little-endian

Byte ordering scheme in which bytes of increasing significance in a data word are stored at increasing addresses in memory.

See also Big-endian and Endianness.

Microprocessor

See Processor.

Processor

A processor is the circuitry in a computer system required to process data using the computer instructions. It is an abbreviation of microprocessor. A clock source, power supplies, and main memory are also required to create a minimum complete working computer system.

Reserved	A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as 0 and read as 0.
SBZ	*See* Should Be Zero.
Should Be Zero (SBZ)	Write as 0, or all 0s for bit fields, by software. Writing as 1 produces Unpredictable results.
Unaligned	A data item stored at an address that is not divisible by the number of bytes that defines the data size is said to be unaligned. For example, a word stored at an address that is not divisible by four.
Unpredictable	For reads, the data returned when reading from this location is unpredictable. It can have any value. For writes, writing to this location causes unpredictable behavior, or an unpredictable change in device configuration. Unpredictable instructions must not halt or hang the processor, or any part of the system.
Word	A 32-bit data item.