Cortex™-A9 MPCore

Revision: r0p1

Technical Reference Manual

ARM®

ARM DDI 0407B

Cortex-A9 MPCore
Technical Reference Manual

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
04 April 2008	A	Non-Confidential	First release for r0p0
08 July 2008	B	Non-Confidential Restricted Access	First release for r0p1

Proprietary Notice

Confidentiality Status

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

Restricted Access is an ARM internal classification.

Product Status

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

Web Address

`http://www.arm.com`

Contents
Cortex-A9 MPCore Technical Reference Manual

List of Tables
Cortex-A9 MPCore Technical Reference Manual

List of Figures
Cortex-A9 MPCore Technical Reference Manual

Preface

This preface introduces the *Cortex-A9 MPCore Technical Reference Manual*. It contains the following sections:

About this manual

This book is for the *Cortex-A9 MPCore*. It provides information that enables designers to integrate the processor into a target system.

Note

The Cortex-A9 MPCore consists of between one and four Cortex-A9 processors and a *Snoop Control Unit* (SCU).

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 verification engineers who are designing a *System-on-Chip* (SoC) device that uses the Cortex-A9 MPCore. The manual describes the external functionality of the Cortex-A9 MPCore.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this for a high-level view of the Cortex-A9 MPCore processor and a description of its features.

Chapter 2 *Cortex-A9 MPCore Private Memory Region*

Read this for a description of the private memory region of the Cortex-A9 MPCore processor.

Chapter 3 *Snoop Control Unit*

Read this for a description of the Snoop Control Unit of the Cortex-A9 MPCore processor.

Chapter 4 *Interrupt Controller*

Read this for a description of the Cortex-A9 MPCore Interrupt Controller.

Note

The *PrimeCell Generic Interrupt Controller (PL390)* and the Cortex A9 Interrupt Controller share the same programmers model. There are implementation-specific differences.

Chapter 5 ***Timer and Watchdog Registers***

Read this for a description of the Cortex-A9 MPCore timer and watchdog registers.

Chapter 6 ***Level 2 Memory Interface***

Read this for a description of the Cortex-A9 MPCore Level 2 Memory Interface.

Chapter 7 ***Clocking, Resets, and Power Management***

Read this for a description of the Cortex-A9 MPCore clocking, resets, and power management.

Chapter 8 ***Cortex-A9 MPCore Debug and Trace***

Read this for a description of the debug and trace considerations in Cortex-A9 MPCore designs.

Appendix A ***Signal Descriptions***

Read this for a description of the Cortex-A9 MPCore input and output signals.

Appendix B ***Revisions***

Read this for a description of 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 xiv
- *Timing diagrams* on page xiv
- *Signals* on page xv.

Typographical

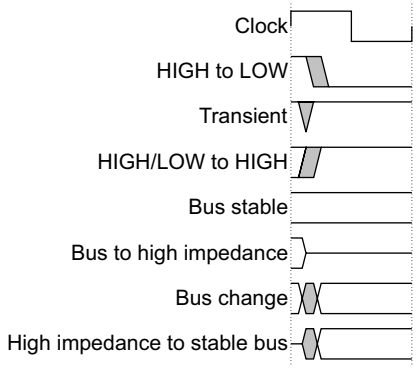
The typographical conventions are:

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.
`mono`space	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
< and >	Angle brackets enclose replaceable terms for assembler syntax where they appear in code or code fragments. They appear in normal font in running text. For example: • `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>` • The Opcode_2 value selects which register is accessed.

Timing diagrams

The figure named *Key to timing diagram conventions* on page xv 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 and LOW for active-LOW signals.

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

Prefix A Denotes global *Advanced eXtensible Interface* (AXI) signals:

Prefix AR Denotes AXI read address channel signals.

Prefix AW Denotes AXI write address channel signals.

Prefix B Denotes AXI write response channel signals.

Prefix C Denotes AXI low-power interface signals.

Prefix H Denotes *Advanced High-performance Bus* (AHB) signals.

Prefix P Denotes *Advanced Peripheral Bus* (APB) signals.

Prefix R Denotes AXI read data channel signals.

Prefix W Denotes AXI write data channel signals.

Additional reading

This section lists publications by ARM and by third parties.

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

ARM publications

This book contains information that is specific to the Cortex-A9 MPCore. See the following documents for other relevant information:

- *Cortex-A9 Technical Reference Manual* (ARM DDI 0338)
- *Cortex-A9 Floating-Point Unit Technical Reference Manual* (ARM DDI 0408)
- *Cortex-A9 NEON Media Processing Engine Technical Reference Manual* (ARM DDI 0409)
- *Cortex-A9 MBIST TRM* (ARM DDI 0414)
- *Cortex-A9 Configuration and Sign-Off Guide* (ARM DII 0146)
- *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition* (ARM DDI 0406)
- *AMBA AXI Protocol v1.0 Specification* (ARM IHI 0022)
- *CoreSight™ PTM™-A9 TRM* (ARM DDI 0401)
- *CoreSight PTM-A9 IM* (ARM DII 0162)
- *CoreSight Program Flow Trace Architecture Specification* (ARM IHI 0035)
- *CoreSight Technology System Design Guide* (ARM DGI 0012)
- *PrimeCell Level 2 Cache Controller (PL310) Technical Reference Manual* (ARM DDI 0246)
- *RealView® Compilation Tools Developer Guide* (ARM DUI 0203)
- *RealView ICE and RealView Trace User Guide* (ARM DUI 0155)
- *Intelligent Energy Controller Technical Overview* (ARM DTO 0005).

Other publications

This section lists relevant documents published by third parties:

- JEDEC Solid State Technology Association, *JEP106, Standard Manufacturer's Identification Cod*e, obtainable at `http://www.jedec.org`.

Feedback

ARM Limited welcomes feedback both on the Cortex-A9 MPCore, 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 if appropriate.

Feedback on this manual

If you have any comments on this manual, send an e-mail to `errata@arm.com`. Give:

- the title
- the number
- the relevant page number(s) to which your comments apply
- a concise explanation of your comments.

ARM Limited also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter describes the Cortex-A9 MPCore processor. It describes the major functional blocks. It contains:

1.1 About the Cortex-A9 MPCore processor

The Cortex-A9 MPCore processor consists of:

- from one to four Cortex-A9 processors in a cluster and a *Snoop Control Unit* (SCU) that can be used to ensure coherency within the cluster.
- an integrated Timer and Watchdog unit per Cortex-A9 processor
- an integrated Interrupt Controller that is an implementation of the Generic Interrupt Controller architecture.
- an optional second master port with programmable address filtering capability
- an optional *Accelerator Coherency Port* (ACP) suitable for coherent memory transfers

It is possible to implement only one Cortex-A9 processor in a Cortex-A9 MPCore processor design. In this configuration, an SCU is still provided. The ACP, and an additional master port, are also available in this configuration.

Figure 1-1 on page 1-3 shows an example multiprocessor configuration.

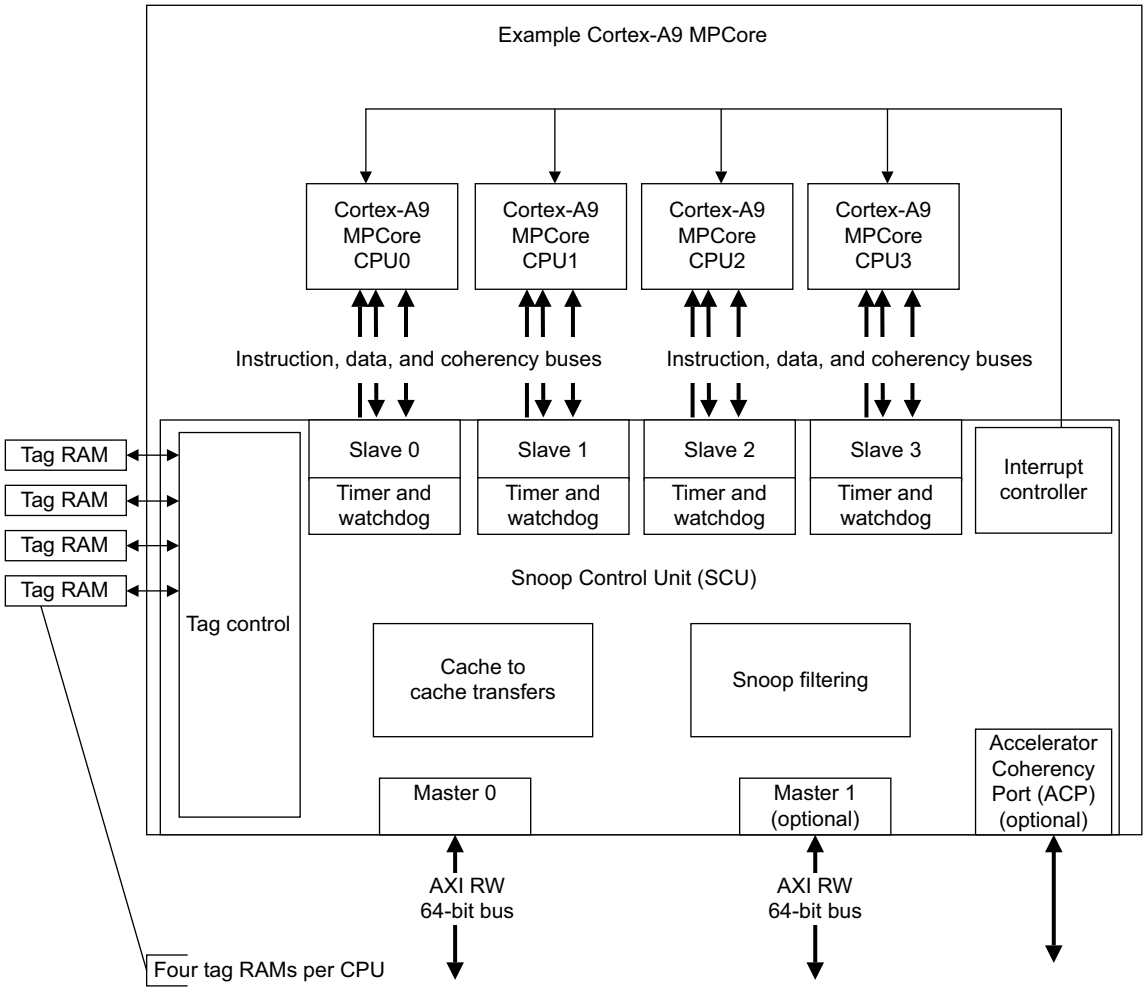


Figure 1-1 Example multiprocessor configuration

1.2 Coherency

Memory coherency in a Cortex-A9 MPCore is maintained following a weakly ordered memory consistency model.

Cache coherency among multiple L1 caches of the Cortex-A9 processors is enforced when the Cortex-A9 processors are operating in *Symmetric Multi-Processing* (SMP) mode. This mode is controlled by the SMP bit, bit 6, of the Auxiliary Control Register.

To be kept coherent, the memory must be marked as Normal memory, with the following attributes:

- write-back
- Write Allocate
- Shared.

Note

When the Shared attribute is applied to a memory region that is not write-back Normal memory, data held in this region is treated as non-cacheable.

1.3 Configurable options

Table 1-1 shows the Cortex-A9 MPCore processor configurable options.

Table 1-1 Configurable options for the Cortex-A9 MPCore processor

Feature	Options
Cortex-A9 processors	One to four
Instruction cache size per Cortex-A9 processor	16KB, 32KB, or 64KB
Data cache size per Cortex-A9 processor	16KB, 32KB, or 64KB
Master ports	One or two
Accelerator Coherency Port	One, included or not
Shared Peripheral Interrupts (SPIs)	0-224, in steps of 32
Media Processing Engine with NEON technology per Cortex-A9 processor[a]	Included or not
FPU per Cortex-A9 processor[b]	Included or not
Jazelle DBX per Cortex-A9 processor	Included or not
Program Trace Macrocell (PTM) interface per Cortex-A9 processor	Included or not
Power off and dormant mode wrappers	Included or not
Support for parity error detection[c]	Included or not

a. Includes support for floating-point operations. If this option is implemented then the FPU option cannot also be implemented.
b. If this option is implemented then the Media Processing Engine with NEON technology option cannot also be implemented.
c. The Cortex-A9 TRM describes the parity error scheme. See *Parity error signals* on page A-27 for a description of the signals.

The MBIST solution you choose must be configured to match the chosen Cortex-A9 MPCore cache sizes. In addition, you must determine the form of the MBIST solution for the RAM blocks in the Cortex-A9 MPCore design when the processor is implemented.

For details, see the Cortex-A9 *MBIST Controller TRM*.

1.4 Accelerator Coherency Port

The *Accelerator Coherency Port* (ACP) is an optional AXI 64-bit slave port that can be connected to a DMA engine or a non-cached peripheral.

This AMBA 3 AXI compatible slave interface on the SCU provides an interconnect point for a range of system masters that for overall system performance, power consumption or reasons of software simplification are better interfaced directly with the Cortex-A9 MPCore processor. See *Accelerator Coherency Port* on page 6-7.

1.5 Product revisions

This section summarizes the differences in functionality between the different releases of this processor:

- *Differences in functionality between r0p0 and r0p1*.

1.5.1 Differences in functionality between r0p0 and r0p1

There is no change in the described functionality between r0p0 and r0p1.

The only differences between the two revisions are:

- r0p1 includes fixes for all known engineering errata relating to r0p0
- r0p1 includes an upgrade of the micro TLB entries from 8 to 32 entries, on both the Instruction and Data side.

Neither of these changes affect the functionality described in this document.

Chapter 2
Cortex-A9 MPCore Private Memory Region

This chapter describes the Cortex-A9 MPCore remappable private memory region. It contains the following section:

- *About the Cortex-A9 MPCore memory region* on page 2-2.

2.1 About the Cortex-A9 MPCore memory region

Most Cortex-A9 processor control operations are through CP15 instructions. These operations are not applicable to Cortex-A9 MPCore processor global control. Cortex-A9 MPCore global control and peripherals must be accessed through memory-mapped transfers.

All registers accessible by all Cortex-A9 processors within the Cortex-A9 MPCore are grouped into two contiguous 4KB pages accessed through a dedicated internal bus. The base address of these pages is defined by the pins **PERIPHBASE[31:13]**. See *Configuration signals* on page A-5 for more information on **PERIPHBASE[31:13]**.

Memory regions used for these registers must be marked as Device or Strongly-ordered in the translation tables.

All these registers can be accessed with single load/store instructions.

Accesses can be word or byte accesses, except in the case of timer and watchdog registers. Byte accesses or doubleword accesses to timer and watchdog registers are not permitted. Load or store multiple accesses cause an abort, to the requesting Cortex-A9 processor and the Fault Status Register shows this as a SLVERR.

Doubleword or halfword accesses cause an abort to the requesting Cortex-A9 processor and the Fault Status Register shows this as a SLVERR.

The *Accelerator Coherency Port* (ACP) cannot access any of the registers in this memory region.

Table 2-1 shows register addresses for the Cortex-A9 MPCore processor relative to this base address.

Table 2-1 Cortex-A9 MPCore memory region

Offset from PERIPHBASE[31:13]	Peripheral	Description
0x0000 - 0x00FC	SCU registers	Chapter 3 *Snoop Control Unit*
0x0100 - 0x01FF	Cortex-A9 processor interrupt interfaces	Chapter 4 *Interrupt Controller*
0x0200 - 0x02FF	Reserved	-
0x0300 - 0x03FF		
0x0400 - 0x04FF		
0x0500 - 0x05FF		

Table 2-1 Cortex-A9 MPCore memory region (continued)

Offset from PERIPHBASE[31:13]	Peripheral	Description
0x0600 - 0x06FF	CPU timer and watchdog	Chapter 5 *Timer and Watchdog Registers*
0x0700 - 0x07FF	Reserved	Any access to this region causes a DECERR abort exception
0x0800 - 0x08FF		
0x0900 - 0x09FF		
0x0A00 - 0x0AFF		
0x0B00 - 0x0FFF		
0x1000 - 0x1FFF	Interrupt Distributor	*About the Interrupt Distributor* on page 4-7

Chapter 3
Snoop Control Unit

This chapter describes the *Snoop Control Unit* (SCU). It contains the following sections:

- *About the SCU* on page 3-2
- *SCU registers* on page 3-3.

3.1 About the SCU

The SCU connects one to four Cortex-A9 processors to the memory system through the AXI interfaces.

The SCU functions are to:

- maintain data cache coherency between the Cortex-A9 processors
- initiate L2 AXI memory accesses
- arbitrate between Cortex-A9 processors requesting L2 accesses
- manage ACP accesses.

Note

The A9 SCU does not support hardware management of coherency of the instruction cache.

3.1.1 Address filtering

The SCU has the option to have one or two external master ports to the underlying memory system. These master ports are 64-bit AXI read and write ports. In the two master port configuration, the SCU can be given an address range that redirects all memory transactions within this range to the second master port. All other memory transactions are routed to the first master port.

The SCU register bank provides the filtering mode enable bits and the address range selection registers. See *Filtering Start Address Register* on page 3-10, *Filtering End Address Register* on page 3-10 and *SCU Control Register* on page 3-4.

Exclusive accesses always go to port M0 if filtering is off. If filtering is on, exclusive accesses go to port M0 or port M1, depending on the address. If the exclusive access is in the filtering range, it goes to M1, otherwise, it goes to M0.

3.1.2 TrustZone extensions

The SCU implements support for the ARM Architecture security extensions. See *SCU Access Control Register (SAC)* on page 3-11 and *SCU Secure Access Control Register (SSAC)* on page 3-12.

3.1.3 SCU event monitoring

The individual CPU event monitors can be configured to gather statistics on the operation of the SCU. See the *Cortex-A9 TRM* for a description of event monitoring.

3.2 SCU registers

Table 3-1 shows the SCU registers. Addresses are relative to the base address of the region for the SCU memory map, which is **PERIPHBASE[31:13]**. All SCU registers are byte accessible and are reset by **nSCURESET**.

Table 3-1 SCU registers summary

Offset from PERIPHBASE [31:13]	Name	Security state		Reset value	Banked	Page
		S	NS			
0x00	SCU Control Register	RW[a]	RW[b]	Implementation defined	N	page 3-4
0x04	SCU Configuration Register	RO	RO	Implementation defined	N	page 3-4
0x08	SCU CPU Power Status Register	RW[a]	RW[b]	-	N	page 3-7
0x0C	Invalidate All In Non-secure State Register	NA	WO[c]	-	Y	page 3-8
0x0C	Invalidate All In Secure State Register	WO[a]	NA	-	Y	page 3-9
0x40	Filtering Start Address Register	RW[a]	RW[b]	Defined by **FILTERSTART** input	N	page 3-10
0x44	Filtering End Address Register	RW[a]	RW[b]	Defined by **FILTEREND** input	N	page 3-10
0x50	*SCU Access Control* (SAC) Register	RW[a]	RW[b]	b1111	N	page 3-11
0x54	*SCU Secure Access Control* (SSAC) Register	RW[a]	NA	0x0	N	page 3-12

a. This register is writable in Secure state if the relevant bit in the SAC register is set.
b. This register is writable in Non-secure state if the relevant bits in the SAC and SSAC registers are set.
c. This register is writable in Non-secure state if the relevant bit in the SAC register is set.

3.2.1 SCU Control Register

The purpose of the SCU Control Register is to:

- enable the SCU
- enable address filtering
- enable parity support.

Figure 3-1 shows the format for this register.

Figure 3-1 SCU Control Register format

Table 3-2 shows the SCU Control Register bit functions.

Table 3-2 SCU Control Register bit functions

Bits	Name	Description
[31:3]	-	Reserved
[2]	Parity ON	1 = Parity on. 0 = Parity off. This is the default setting. This bit is always zero if support for parity is not implemented.
[1]	Address filtering enable	1 = Addressing filtering on. 0 = Addressing filtering off. The default value is the value of **FILTEREN** sampled on exit from reset. This bit is always zero if the SCU is implemented in the single master port configuration.
[0]	SCU enable	1 = SCU enable. 0 = SCU disable. This is the default setting.

3.2.2 SCU Configuration Register

The purpose of the SCU Configuration Register is to:

- read tag RAM sizes for the Cortex-A9 processors that are present
- determine the Cortex-A9 processors that are taking part in coherency

- read the number of Cortex-A9 processors present.

The SCU Configuration Register is read-only. Figure 3-2 shows the format for this register.

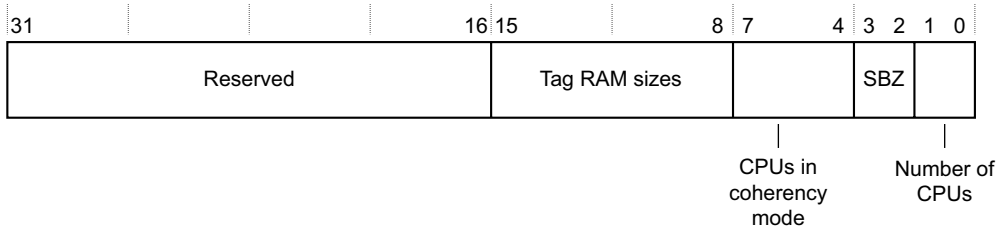


Figure 3-2 SCU Configuration Register format

Table 3-3 shows the SCU Configuration Register bit assignments.

Table 3-3 SCU Configuration Register functions

Bits	Name	Description
[31:16]	Reserved	*Should-Be-Zero* (SBZ).
[15:8]	Tag RAM sizes	Bits [15:14] indicate Cortex-A9 processor CPU3 tag RAM size if present. Bits [13:12] indicate Cortex-A9 processor CPU2 tag RAM size if present. Bits [11:10] indicate Cortex-A9 processor CPU1 tag RAM size if present. Bits [9:8] indicate Cortex-A9 processor CPU0 tag RAM size. The encoding is as follows: b11 = 64KB cache, 256 indexes per tag RAM b10 = reserved b01 = 32KB cache, 128 indexes per tag RAM b00 = 16KB cache, 64 indexes per tag RAM.

Table 3-3 SCU Configuration Register functions (continued)

Bits	Name	Description
[7:4]	CPUs SMP	Defines the Cortex-A9 processors that are in *Symmetric Multi-processing* (SMP) or *Asymmetric Multi-processing* (AMP) mode. 0 = this Cortex-A9 processor is in AMP mode not taking part in coherency or not present. 1 = this Cortex-A9 processor is in SMP mode taking part in coherency. Bit 7 is for CPU3 Bit 6 is for CPU2 Bit 5 is for CPU1 Bit 4 is for CPU0.
[3:2]	Reserved	SBZ
[1:0]	CPU number	Number of CPUs present in the Cortex-A9 MPCore processor b11 = four Cortex-A9 processors, CPU0, CPU1, CPU2, and CPU3 b10 = three Cortex-A9 processors, CPU0, CPU1, and CPU2 b01 = two Cortex-A9 processors, CPU0 and CPU1 b00 = one Cortex-A9 processor, CPU0.

Enabling coherency mode

By default, each Cortex-A9 processor is not in coherent mode. That is, the SMP bit, bit [6] of the CP15 Auxiliary Control Register, is set to zero. Also, the SCU is not enabled. That is, the SCU enable bit, bit[0] of the SCU Control Register, is set to zero.

To prevent coherent data corruption the sequence to bring the processor into coherency mode is:

1. If the SCU is disabled, enable it by writing a 1 to the SCU enable bit of the SCU Control Register.

   ARM recommends that you invalidate the SCU tag RAMs before enabling the SCU. Invalidate the SCU tag RAMs with the appropriate command in the SCU Invalidate All Register. Not invalidating the SCU tag RAMs incurs a performance penalty.

   Note

   This behavior is not guaranteed for future revisions of the product.
2. Set the SMP bit to one.
3. Disable interrupts.
4. Clean and invalidate the data cache.
5. Enable interrupts.

Similarly, the sequence to bring the processor out of coherency mode is:

1. Disable interrupts.
2. Clean and invalidate the data cache.
3. Set the SMP bit to zero
4. Data Synchronization Barrier.
5. Enable interrupts.

Note

In AMP mode, shared write-back write-allocate regions are treated as non-cacheable, as with other Shared regions.

3.2.3 SCU CPU Power Status Register

The purpose of the SCU CPU Power Status Register is to specify the state of the Cortex-A9 processors with reference to power modes.

Figure 3-3 shows the format for the SCU CPU Power Status Register.

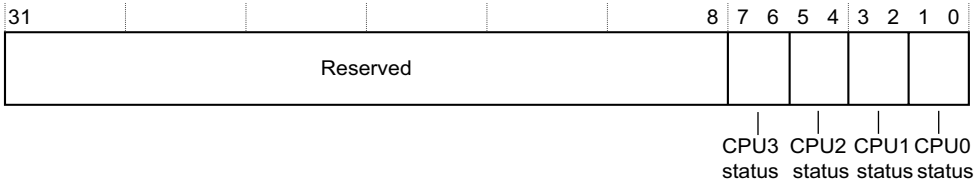


Figure 3-3 SCU CPU Power Status Register

Table 3-4 shows the SCU CPU Power Status Register bit assignments.

Table 3-4 SCU CPU Power Status Register bit assignments

Bits	Name	Description
[31:8]	Reserved	SBZ
[7:6]	CPU3 status	Power status of the Cortex-A9 processor: b00: Normal mode. b01: RESERVED. b10: the Cortex-A9 processor is about to enter (or is in) dormant mode. No coherency request is sent to the Cortex-A9 processor. b11: the Cortex-A9 processor is about to enter (or is in) powered-off mode, or is nonpresent. No coherency request is sent to the Cortex-A9 processor.
[5:4]	CPU2 status	
[3:2]	CPU1 status	
[1:0]	CPU0 status	

Dormant mode and powered-off mode are controlled by an external power controller. SCU CPU Status Register bits indicate to the external power controller the power domains that can be powered down.

Before entering any other power mode than Normal, the Cortex-A9 processor must set its status field to signal to the SCU the mode it is about to enter so that the SCU can determine if it still can send coherency requests to the Cortex-A9 processor. The Cortex-A9 processor then executes a WFI entry instruction. When in WFI state, the **PWRCTLOn** bus is enabled and signals to the power controller what it must do with power domains.

The SCU CPU Power Status Register bits can also be read by a Cortex-A9 processor exiting low-power mode to determine its state before executing its reset set-up.

Cortex-A9 processors status fields take **PWRCTLIn** values at reset, except for nonpresent Cortex-A9 processors. For nonpresent Cortex-A9 processors writing to this field has no effect.

3.2.4 SCU Invalidate All Register in Non-secure State

The purpose of the SCU Invalidate All Register in Non-secure state is to invalidate the SCU tag RAMs on a per Cortex-A9 processor and per way basis. This operation invalidates only Non-secure lines in the selected ways.

Figure 3-4 shows the format of this register.

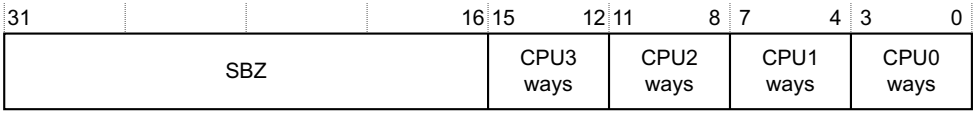


Figure 3-4 SCU Invalidate All Register in Non-secure state format

Table 3-5 shows the SCU Invalidate All Register in Non-secure state bit assignments.s

Table 3-5 SCU Invalidate All Register in non-secure state bit assignment

Bits	Name	Description
[31:16]	-	-
[15:12]	CPU3 ways	Indicates the ways that must be invalidated for Cortex-A9 MPCore CPU3. Writing to these bits has no effect if the Cortex-A9 MPCore processor has fewer than four Cortex-A9 processors.

Table 3-5 SCU Invalidate All Register in non-secure state bit assignment (continued)

Bits	Name	Description
[11:8]	CPU2 ways	Indicates the ways that must be invalidated for Cortex-A9 MPCore CPU2. Writing to these bits has no effect if the Cortex-A9 MPCore processor has fewer than three Cortex-A9 processors.
[7:4]	CPU1 ways	Indicates the ways that must be invalidated for Cortex-A9 MPCore CPU1. Writing to these bits has no effect if the Cortex-A9 MPCore processor has fewer than two Cortex-A9 processors.
[3:0]	CPU0 ways	Indicates the ways that must be invalidated for Cortex-A9 MPCore CPU0.

3.2.5 SCU Invalidate All Register in Secure State

The purpose of the SCU Invalidate All Register in Secure State is to invalidate the SCU tag RAMs on a per Cortex-A9 processor and per way basis. This operation invalidates all lines in the selected ways.

Figure 3-5 shows the format of this register.

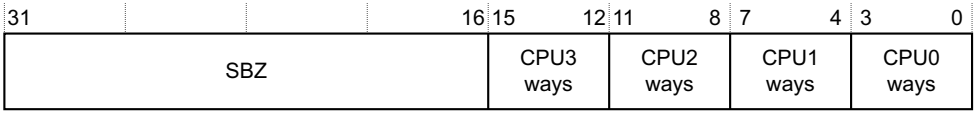


Figure 3-5 SCU Invalidate All Register in Secure state format

Table 3-6 shows the SCU Invalidate All Register in Secure state bit assignments.

Table 3-6 SCU Invalidate All Register in Secure state bit assignment

Bits	Name	Description
[31:16]	-	-
[15:12]	CPU3 ways	Indicates the ways that must be invalidated for Cortex-A9 MPCore CPU3. Writing to these bits has no effect if the Cortex-A9 MPCore processor has fewer than four processors.
[11:8]	CPU2 ways	Indicates the ways that must be invalidated for Cortex-A9 MPCore CPU2. Writing to these bits has no effect if the Cortex-A9 MPCore processor has fewer than three processors.
[7:4]	CPU1 ways	Indicates the ways that must be invalidated for Cortex-A9 MPCore CPU1. Writing to these bits has no effect if the Cortex-A9 MPCore processor has fewer than two processors.
[3:0]	CPU0 ways	Indicates the ways that must be invalidated for Cortex-A9 MPCore CPU0.

3.2.6 Filtering Start Address Register

The purpose of the Filtering start address register is to provide the start address for use with master port 1 in a two-master port configuration.

Figure 3-6 shows the format of this register.

Figure 3-6 Filtering Start Address Register formats

Table 3-7 shows the bit assignment for the Filtering Start Address Register.

Table 3-7 Filtering Start Address Register bit assignment

Bits	Name	Description
[31:20]	Filtering start address	Start address for use with master port 1 in a two-master port configuration when address filtering is enabled. The default value is the value of **FILTERSTART** sampled on exit from reset.
[19:0]	-	SBZ

See *Configuration signals* on page A-5 for a description of the **FILTERSTART** signal.

3.2.7 Filtering End Address Register

The purpose of the Filtering End Address Register is to provide the end address for use with master port 1 in a two-master port configuration. This is an inclusive address so that the topmost megabyte of address space of memory can be included in the filtering address range.

Figure 3-7 shows the format of this register.

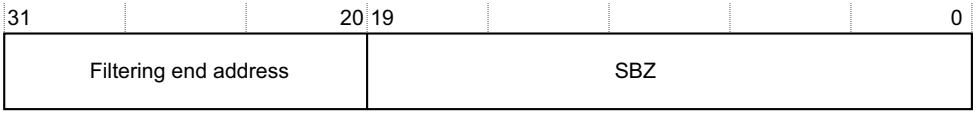


Figure 3-7 Filtering End Address Register format

Table 3-8 shows the bit assignment for the Filtering End Address Register.

Table 3-8 Filtering End Address Register bit assignment

Bits	Name	Description
[31:20]	Filtering end address	End address for use with master port 1 in a two-master port configuration, when address filtering is enabled. The default value is the value of **FILTEREND** sampled on exit from reset.
[19:0]	-	SBZ

See *Configuration signals* on page A-5 for a description of the **FILTEREND** signal.

3.2.8 SCU Access Control Register (SAC)

The SAC register controls access to the following registers on a per Cortex-A9 processor basis:

- *SCU Control Register* on page 3-4
- *SCU CPU Power Status Register* on page 3-7
- *SCU Invalidate All Register in Non-secure State* on page 3-8
- *SCU Invalidate All Register in Secure State* on page 3-9
- *Filtering Start Address Register* on page 3-10
- *Filtering End Address Register* on page 3-10
- *SCU Access Control Register (SAC)*
- *SCU Secure Access Control Register (SSAC)* on page 3-12.

Figure 3-8 shows the format for this register.

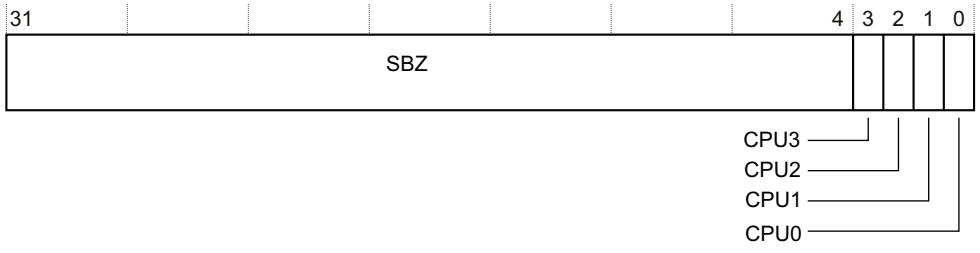


Figure 3-8 SCU Access Control Register bit assignment

Table 3-9 shows the bit assignment for the SCU Access Control Register.

Table 3-9 SCU Access Control Register bit assignment

Bits	Name	Description
[31:4]	SBZ	-
[3]	CPU3	0 = CPU3 cannot access the components[a] 1 = CPU3 can access the components This is the default.
[2]	CPU2	0 = CPU2 cannot access the components. 1 = CPU2 can access the components.This is the default.
[1]	CPU1	0 = CPU1 cannot access the components. 1 = CPU1 can access the components. This is the default.
[0]	CPU0	0 = CPU0 cannot access the components. 1 = CPU0 can access the components. This is the default.

a. The accessible components are the SAC Register, the SCU Control Register, the SCU CPU Status Register, the SCU Invalidate All Register and the filtering registers.

3.2.9 SCU Secure Access Control Register (SSAC)

The purpose of the SCU Secure Access Control Register (SSAC) is to control whether SCU registers can be accessed by the respective Cortex-A9 processors, in Non-secure state. It controls access to the following registers:

- *SCU Control Register* on page 3-4
- *SCU CPU Power Status Register* on page 3-7
- *Filtering Start Address Register* on page 3-10
- *Filtering End Address Register* on page 3-10
- *SCU Access Control Register (SAC)* on page 3-11.

The SSAC also controls access to the timer and watchdog blocks of the respective Cortex-A9 processors.

Figure 3-9 on page 3-13 shows the format of this register.

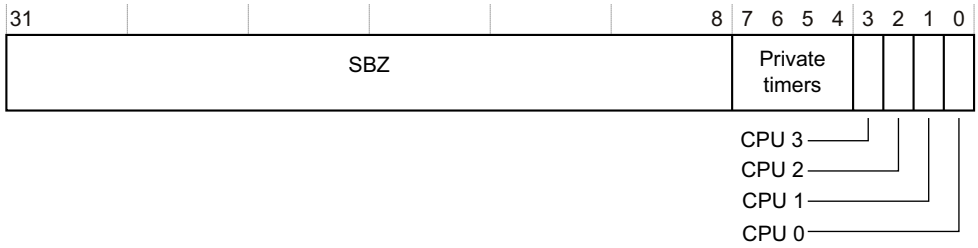


Figure 3-9 SCU Secure Access Control Register bit assignment

Table 3-10 shows the bit assignment for the SCU Secure Access Control Register.

Table 3-10 SCU Secure Access Control Register bit assignment

Bits	Name	Description
[31:8]	SBZ	-
[7]	Timer for CPU3	Private timer for CPU3: 0 = Secure accesses only. Non-secure reads return 0. This is the default value. 1 = Secure accesses and Non-secure accesses.
[6]	Timer for CPU2	Private timer for CPU2: 0 = Secure accesses only. Non-secure reads return 0. This is the default value. 1 = Secure accesses and Non-secure accesses.
[5]	Timer for CPU1	Private timer for CPU1: 0 = Secure accesses only. Non-secure reads return 0. This is the default value. 1 = Secure accesses and Non-secure accesses.
[4]	Timer for CPU0	Private timer for CPU0: 0 = Secure accesses only. Non-secure reads return 0. This is the default value. 1 = Secure accesses and Non-secure accesses.
[3]	CPU3	0 = CPU3 cannot write the components[a] 1 = CPU3 can access the components[a].
[2]	CPU2	0 = CPU2 cannot write the components[a]. 1 = CPU2 can access the components[a].
[1]	CPU1	0 = CPU1 cannot write the components[a]. 1 = CPU1 can access the components[a].
[0]	CPU0	0 = CPU0 cannot write the components[a]. 1 = CPU0 can access the components[a].

a. The accessible components are the SAC Register, the SCU Control Register, the SCU CPU Status Register, the SCU Invalidate All Register and the filtering registers.

Chapter 4
Interrupt Controller

This chapter describes the Interrupt Controller. The Interrupt Controller collates interrupts from a number of sources and arbitrates between them. The chapter contains the following sections:

Note

The *PrimeCell Generic Interrupt Controller GIC (PL390)* and the Cortex A9 Interrupt Controller share a programmers model. There are implementation-specific differences.

4.1 About the Interrupt Controller

The Interrupt Controller collates interrupts from a large number of sources. It provides:

- masking of interrupts
- prioritization of the interrupts
- distribution of the interrupts to the target Cortex-A9 processors
- tracking the status of interrupts
- generation of interrupts by software
- support for Security Extensions.

The Interrupt Controller is a single functional unit that is located in a Cortex-A9 multiprocessor design.

The Interrupt Controller is memory-mapped. The Cortex-A9 processors access it by using a private interface through the SCU. You can access the Interrupt Controller using **PERIPHBASE[31:13]**. See Chapter 2 *Cortex-A9 MPCore Private Memory Region*.

The Interrupt Controller consists of:

Interrupt distributor

The interrupt distributor handles interrupt detection, interrupt prioritization, and distribution of interrupts to individual CPUs.

Cortex-A9 processor interfaces

There is one Cortex-A9 processor interface per Cortex-A9 processor. The Cortex-A9 processor interfaces handle

- interrupt acknowledgement
- interrupt masking
- interrupt completion acknowledgement.

Timer and watchdog interfaces

These interfaces generate interrupts for the Interrupt Controller.

4.1.1 Interrupt Controller Clock frequency

The clock period is configured, during integration, as a multiple of the SCU clock period. This multiple, N, must be greater than or equal to two. As a consequence, the minimum pulse width of signals driving external interrupt lines is N Cortex-A9 processor clock cycles. See *Clocking* on page 7-2 for a description of **PERIPHCLK** and **PERIPHCLKEN**.

The timers and watchdogs use the same user clock as the interrupt controller.

4.2 Terminology

This manual uses the following interrupt terminology for states:

Active state

A Cortex-A9 processor is servicing the interrupt.

Active and pending state

A Cortex-A9 processor is servicing the interrupt and the Interrupt Controller has a pending interrupt from the same source.

Pending state

The Interrupt Controller has received the interrupt but the target Cortex-A9 processor is not servicing the interrupt.

Inactive state

An interrupt that is not active, active and pending, nor pending.

This manual uses the following interrupt terminology for interrupts:

Preempted interrupt

While the Cortex-A9 processor is servicing an interrupt, it receives a higher priority interrupt and starts servicing this interrupt. The Cortex-A9 processor suspends servicing the initial interrupt. In this state, the initial interrupt has been preempted.

Software Generated Interrupt **(SGI)**

Generated by writing to the sgi_trigger Register. A maximum of 16 SGIs can be generated for each Cortex-A9 processor interface. See *Software Generated Interrupt Register* on page 4-50.

Private Peripheral Interrupt **(PPI)**

An interrupt generated by a peripheral that is specific to a single Cortex-A9 processor. There are 4 PPIs for each Cortex-A9 processor interface. See *Interrupt Distributor interrupt sources* on page 4-9.

Shared Peripheral Interrupt **(SPI)**

An interrupt generated by a peripheral that the Interrupt Controller can route to any, or all, Cortex-A9 processor interfaces.

The Interrupt Controller supports a maximum of 224 SPIs. The permitted values are 0, 32, 64, 96, 128, 160, 192, or 224.

Lockable Shared Peripheral Interrupts (LSPI)

There are 31 LSPIs. You can configure and then lock these interrupts against further change using **CFGSDISABLE**. The LSPIs are present only if the SPIs are present. See *SPI Status Registers* on page 4-48.

4.3 TrustZone support

The Interrupt Controller permits all implemented interrupts to be individually defined as secure or non-secure. See *Interrupt security registers* on page 4-23.

You can program Secure interrupts to use either the IRQ or FIQ interrupt mechanism of a Cortex-A9 processor through the FIQen bit in the ICPICR Register.See *Processor Interface Control Register* on page 4-60. Non-secure interrupts are always signalled using the IRQ mechanism of a Cortex-A9 processor. See Table 4-1 on page 4-12.

Note

A Cortex-A9 processor might stall an active interrupt if it receives a higher priority interrupt. See *Terminology* on page 4-3 for a definition of Preempted interrupt.

Note

A Non-secure access to a register of a Secure interrupt behaves as RAZ/WI.

4.3.1 Priority formats

The software view of priority fields depends on the status of the access request to the priority field (NS-prot), and the security status (NS-int) of the interrupt that the priority field refers to.

The priority space is partitioned to ensure that secure interrupts can always be given a priority higher than any Non-secure interrupt. The Non-secure domain observes a smaller available range of priority levels than the range available to the secure domain as Figure 4-1 on page 4-6 shows.

In Figure 4-1 on page 4-6, Priority format A shows the format this implementation uses for Secure accesses. Priority format B shows the format this implementation uses for Non-secure accesses. Bit D is the MSB of the Non-secure interrupt priority view. The LSB is always zero. Priority format C shows the non-secure interrupt priority internal format as viewed by Secure accesses. The MSB is usually one and it is automatically set for Non-secure writes.

Note

Priority zero is the highest priority. The lowest priority is priority `0x1F`.

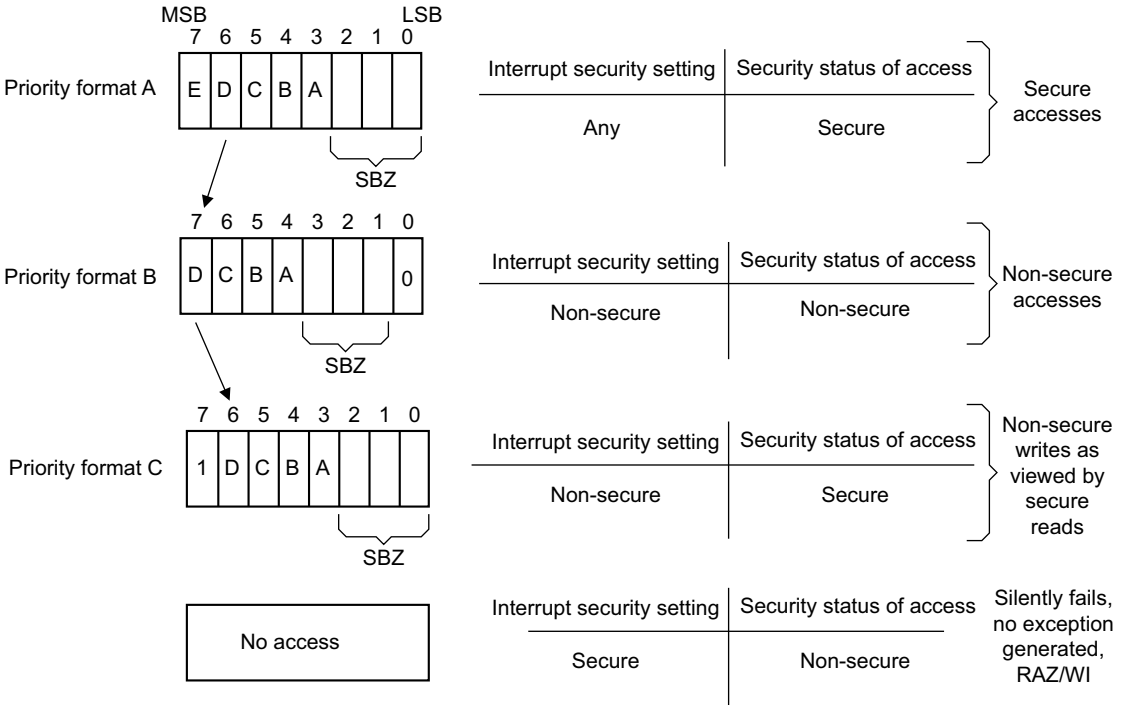


Figure 4-1 Secure and Non-secure interrupt priority formats

4.4 About the Interrupt Distributor

The interrupt distributor holds the list of pending interrupts for each Cortex-A9 processor, and then selects the highest priority interrupt before issuing it to the Cortex-A9 processor interface. Interrupts of equal priority are resolved by selecting the lowest ID.

The interrupt distributor consists of a register-based list of interrupts, their priorities and activation requirements, Cortex-A9 processor targets, and their pending and active status.

The prioritization logic is physically duplicated to enable the selection of the highest priority interrupt for each Cortex-A9 processor.

The interrupt distributor holds the central list of interrupts, processors and activation information, and is responsible for triggering software interrupts to processors.

The Cortex-A9 processor interface acknowledges interrupts and changes interrupt priority masks.

Hardware ensures that an interrupt targeted at several Cortex-A9 processors can only be taken by one Cortex-A9 processor at a time.

The interrupt distributor transmits to the Cortex-A9 processor interfaces their highest pending interrupt. It receives back the information that the interrupt has been acknowledged, and can then change the status of the corresponding interrupt. The Cortex-A9 processor interface also transmits *End of Interrupt Information* (EOI), enabling the interrupt distributor to update the status of this interrupt from active to inactive.

Figure 4-2 shows the interfaces of the interrupt distributor.

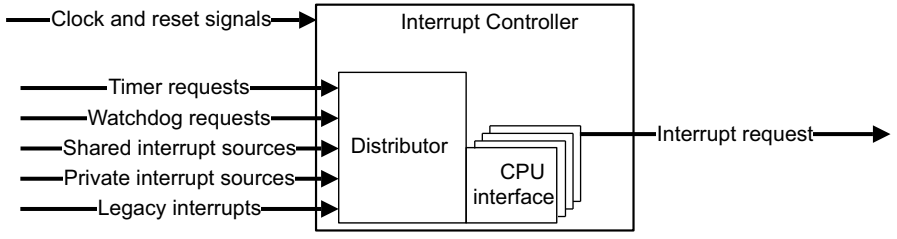


Figure 4-2 Interrupt Controller interfaces

4.4.1 Behavior of the Interrupt Distributor

When the interrupt distributor detects an interrupt assertion, it sets the status of the interrupt for the targeted Cortex-A9 processors to pending. Level-triggered interrupts cannot be marked as pending if they are active for at least one Cortex-A9 processor.

For each Cortex-A9 processor the prioritization and selection block searches for the pending interrupt with the highest priority. This interrupt is then sent with its priority to the Cortex-A9 processor interface.

When multiple pending interrupts have the same priority, the selected interrupt is the one with lowest ID. If there are multiple pending software-generated interrupts with the same ID, the lowest Cortex-A9 processor source is selected.

The Cortex-A9 processor interface returns information to the distributor when the Cortex-A9 processor acknowledges (pending to active transition) or clears an interrupt (active to inactive transition). With the given interrupt ID, the interrupt distributor updates the status of this interrupt according to the information sent by the Cortex-A9 processor interface.

When an interrupt is triggered by the Software Interrupt Register or the Set-pending Register, the status of that interrupt for the targeted Cortex-A9 processor or Cortex-A9 processors is set to pending. This interrupt then has the same behavior as a hardware interrupt. The distributor does not differentiate between software and hardware triggered interrupts.

4.5 Interrupt Distributor interrupt sources

The interrupt distributor centralizes all interrupt sources before dispatching the highest priority ones to each individual Cortex-A9 processor.

All interrupt sources are identified by a unique ID. All interrupt sources have their own configurable priority and list of targeted Cortex-A9 processors, that is, a list of Cortex-A9 processors that the interrupt is sent to when triggered by the interrupt distributor.

Interrupt sources are of the following types:

Software Generated Interrupts (SGI)

Each Cortex-A9 processor has private interrupts, ID0-ID15, that can only be triggered by software. These interrupts are aliased so that there is no requirement for a requesting Cortex-A9 processor to determine its own CPU ID when it deals with SGIs. The priority of an SGI depends on the value set by the receiving Cortex-A9 processor in the banked SGI priority registers, not the priority set by the sending Cortex-A9 processor.

A legacy nFIQ pin, PPI(0)

In legacy FIQ mode the legacy **nFIQ** pin, on a per Cortex-A9 processor basis, bypasses the interrupt distributor logic and directly drives interrupt requests into the Cortex-A9 processor.

When a Cortex-A9 processor uses the Interrupt Controller, rather than the legacy pin in the legacy mode, by enabling its own Cortex-A9 processor interface, the legacy **nFIQ** pin is treated like other interrupt lines and uses ID28.

Private timer, PPI(1)

Each Cortex-A9 processor has its own private timers that can generate interrupts, using ID29.

Watchdog timers, PPI(2)

Each Cortex-A9 processor has its own watchdog timers that can generate interrupts, using ID30.

A legacy nIRQ pin, PPI(3)

In legacy IRQ mode the legacy **nIRQ** pin, on a per Cortex-A9 processor basis, bypasses the interrupt distributor logic and directly drives interrupt requests into the Cortex-A9 processor.

When a Cortex-A9 processor uses the Interrupt Controller (rather than the legacy pin in the legacy mode) by enabling its own Cortex-A9 processor interface, the legacy **nIRQ** pin is treated like other interrupt lines and uses ID31.

Shared Peripheral Interrupts (SPI)

SPIs are triggered by events generated on associated interrupt input lines. The Interrupt Controller can support up to 224 interrupt input lines. The interrupt input lines can be configured to be edge sensitive (posedge) or level sensitive (high level). SPIs start at ID32.

4.6 Cortex-A9 processor interfaces

The Cortex-A9 processor interfaces are slaves to the Cortex-A9 processors. The Cortex-A9 processor interfaces handle interrupt priority masking and interrupt preemption.

A pending interrupt is only accepted if its priority is higher than the priority mask and is also higher than the priority of the highest priority active interrupt active on that Cortex-A9 processor. If a pending interrupt is accepted, the effect is that an interrupt request is made to the Cortex-A9 processor for interrupt exception entry.

If the Cortex-A9 processor then reads its Interrupt Acknowledge Register, the Cortex-A9 processor interface records the priority of this interrupt and marks it as active in the interrupt distributor for that Cortex-A9 processor.

If the interrupt is cleared before the Cortex-A9 processor reads its Interrupt Acknowledge Register, for example because of a priority mask change or a write to the Interrupt Pending Clear Register, the Cortex-A9 processor gets the interrupt ID value 1023, indicating a spurious interrupt. If an interrupt is sent by several processors, only the first one gets this interrupt ID and other processors read the spurious ID, or another pending interrupt ID.

The interrupt active to inactive transition is triggered by an Cortex-A9 processor writing the completed Interrupt ID in its End of Interrupt Register.

4.6.1 Interrupt output source selection

There are two legacy interrupt inputs, **nFIQ[n]** and **nIRQ[n]**, for each Cortex-A9 processor. When you use the legacy mode, the Interrupt Controller disables the corresponding Cortex-A9 processor interface and it routes the legacy interrupt inputs to the Cortex-A9 processor generating IRQ and FIQ exceptions respectively. Otherwise, these pins are used as **PPI(0)** and **PPI(3)**.

Table 4-1 shows the bits in the ICPICR Register that enable you to select the signals that drive the interrupt outputs of a Cortex-A9 processor interface.

Table 4-1 Interrupt output source selection

ICPICR Register			Interrupt output signals	
Bit [3], FIQEn	**Bit [1], EnableNS**	**Bit [0], EnableS**	**FIQ exception generated by**	**IRQ exception generated by**
0	0	0	**nFIQ[n]**	**nIRQ[n]**
0	0	1	**nFIQ[n]**	Secure interrupts
0	1	0	**nFIQ[n]**	Non-secure interrupts
0	1	1	**nFIQ[n]**	Secure and Non-secure interrupts
1	0	0	**nFIQ[n]**	**nIRQ[n]**
1	0	1	Secure interrupts	**nIRQ[n]**
1	1	0	**nFIQ[n]**	Non-secure interrupts
1	1	1	Secure interrupts	Non-secure interrupts

4.6.2 Using CFGSDISABLE

The Interrupt Controller provides the facility to prevent write accesses to critical configuration registers when you assert **CFGSDISABLE**. This signal controls the read and write behavior for the secure control registers in the distributor and Cortex-A9 processor interfaces, and the *Lockable Shared Peripheral Interrupts* (LSPIs) in the Interrupt Controller.

If you use **CFGSDISABLE**, ARM recommends that you assert **CFGSDISABLE** during the system boot process, after the software has configured the registers. Ideally, the system must only deassert **CFGSDISABLE** if a hard reset occurs.

When **CFGSDISABLE** is HIGH, the Interrupt Controller prevents write accesses to the following registers in the:

Distributor

The enable_s Register. See *Distributor Enable Registers* on page 4-18.

Secure interrupts defined by LSPI field in the ic_type_reg:

- Interrupt Security Registers. See *Interrupt security registers* on page 4-23.

- Enable Set Registers. See *Enable set registers* on page 4-26.
- Enable Clear Registers. See *Enable clear registers* on page 4-29.
- Pending Set Registers. See*Pending set registers* on page 4-32.
- Pending Clear Registers. See *Pending clear registers* on page 4-34.
- Priority Level Registers. See *Priority level registers* on page 4-38.
- SPI Target Registers. See *SPI Target registers* on page 4-41.
- Interrupt Configuration Register. See *Interrupt Configuration Registers* on page 4-44.

Cortex-A9 processor interface

The ICPICR Register, except for the EnableNS bit. See *Processor Interface Control Register* on page 4-60.

Note

When **CFGSDISABLE** is HIGH the Interrupt Controller only permits write access to the EnableNS bit. All other bits are read-only.

After you assert **CFGSDISABLE**, it changes the register bits to read-only and therefore the behavior of these secure interrupts cannot change, even in the presence of rogue code executing in the secure domain.

4.7 Interrupt distributor programmers model

The interrupt distributor contains the following register maps:

- *Distributor register map*
- *Cortex-A9 processor interface register map* on page 4-16.

4.7.1 Distributor register map

The distributor has 4KB of memory allocated to it, as Figure 4-3 on page 4-15 shows. See Table 2-1 on page 2-2 for the offset of this page from PERIPHBASE[31:13].

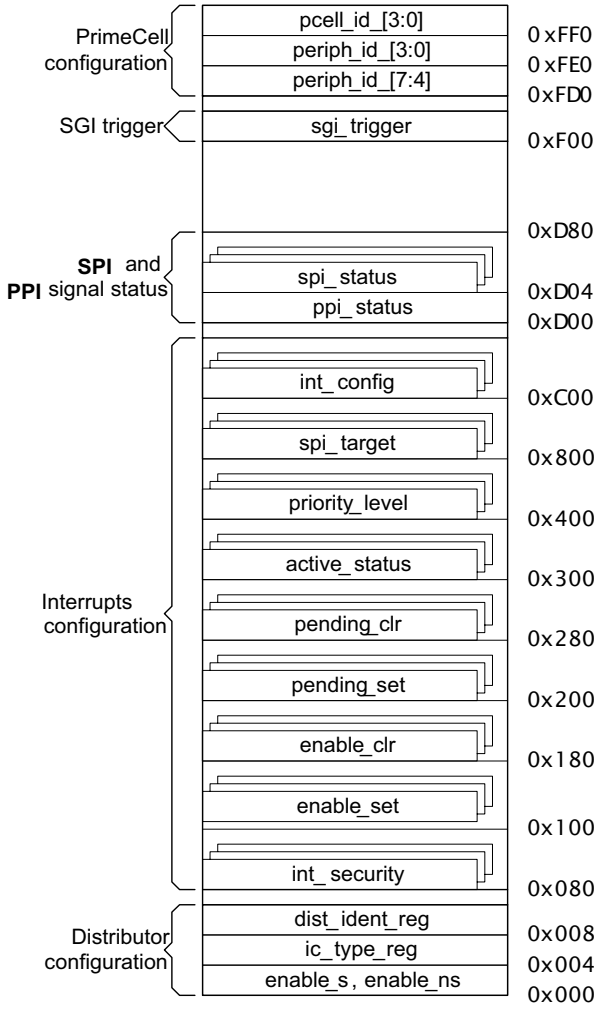


Figure 4-3 Distributor register map

In Figure 4-3, the register map address range contains the following regions:

Distributor configuration

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

Interrupts configuration

These registers hold the operating parameters for each interrupt.

PPI and SPI status

These registers provide the status of the **PPI** and **SPI** input signals.

SGI trigger Use this register to generate a Software Generated Interrupt.

4.7.2 Cortex-A9 processor interface register map

Each Cortex-A9 processor interface has 256 bytes of memory allocated to it, as Figure 4-4 shows.

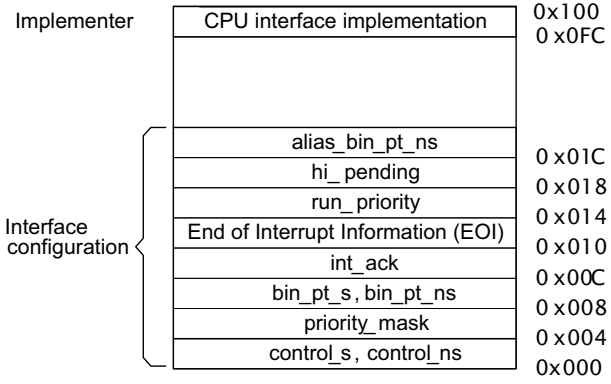


Figure 4-4 Cortex-A9 processor interface register map

In Figure 4-4, the register map address range contains the following regions:

Interface configuration

Use these registers to control the operating state of the Cortex-A9 processor interface.

Implementer

These registers enable the identification of the processor interface by software.

4.7.3 Distributor register descriptions

This section describes the registers that the distributor provides. Table 4-2 shows the distributor registers. See Table 2-1 on page 2-2 for the offset of this page from **PERIPHBASE[31:13]**.

Table 4-2 Distributor register summary

Base	Name	Type	Reset	Width	Description
0x000	enable_s or enable_ns	RW	0x00000000	32	*enable_s Register and enable_ns Register bit assignments* on page 4-19
0x004	ICDICTR	RO	Configuration dependent	32	*Interrupt Controller Type Register* on page 4-20
0x008	ICDDIR	RO	0x0000043B	32	*Distributor Implementer Identification Register* on page 4-22
0x00C - 0x07C	-	-	-	-	Reserved
0x080 - 0x09C	ICDISR	RW[a]	0x00000000	32	*Interrupt security registers* on page 4-23
0x100 - 0x11C	ICDISER	RW	0x00000000[b]	32	*Enable set registers* on page 4-26
0x180 - 0x19C	ICDICER	RW	0x00000000[b]	32	*Enable clear registers* on page 4-29
0x200 - 0x21C	IDISPR	RW	0x00000000	32	*Pending set registers* on page 4-32
0x280 - 0x29C	ICDICPR	RW	0x00000000	32	*Pending clear registers* on page 4-34
0x300 - 0x31C	ICDABR	RO	0x00000000	32	*Active status registers* on page 4-37
0x380 - 0x3FC	-	-	-	-	Reserved
0x400 - 0x4FC	ICDIPR	RW	0x00000000	32	*Priority level registers* on page 4-38
0x7FC	-	-	-	-	Reserved
0x800 - 0x8FC	ICDIPTR	RW[c]	0x0000000	32	*SPI Target registers* on page 4-41
0xBFC	-	-	-	-	Reserved
0xC00 - 0xC3C	ICDICR	RW	Configuration dependent	32	*Interrupt Configuration Registers* on page 4-44
0xD00	ppi_status	-	0x00000000	32	*PPI Status Register* on page 4-47
0xD04 -0xD1C	spi_status	RO	0x00000000	32	*SPI Status Registers* on page 4-48
0xD80 -0xEFC	-	-	-		Reserved

Table 4-2 Distributor register summary (continued)

Base	Name	Type	Reset	Width	Description
0xF00	ICDSGIR	WO	-	32	*Software Generated Interrupt Register* on page 4-50
0xF04 - 0xFCC	-	-	-	-	Reserved
0xFE0 - 0xFEC	periph_id_[4:0]	RO	Configuration dependent	8	*Peripheral Identification Registers* on page 4-52
0xFF0 - 0xFFC	component_id_[3:0]	RO	-	8	*PrimeCell Identification Registers* on page 4-56

a. You must access this register in Secure state.
b. The reset value for the registers that contain the SGI and PPI interrupts is implementation-dependent.
c. The registers that control the SGI and PPI interrupts are read-only and the reset value is configuration-dependent.

All registers not described in Table 4-2 on page 4-17 are Reserved and read as zero. Writing to these registers has no effect.

The ICDIPR and ICDIPTR registers are byte and word accessible. All other registers in Table 4-2 on page 4-17 are word accessible.

4.7.4 Distributor Enable Registers

The enable_s and enable_ns Register characteristics are:

Purpose The enable_s and enable-ns Registers control if the distributor responds to external stimulus changes that occur on **SPI** and **PPI**.

Usage constraints This register is banked. The register you access depends on the type of access:

- **Non-secure access** distributor provides access to the enable_ns Register.
- **Secure access** distributor provides access to the enable_s Register.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-2 on page 4-17.

Figure 4-5 on page 4-19 shows the enable_ns Register bit assignments and the enable_s Register bit assignments.

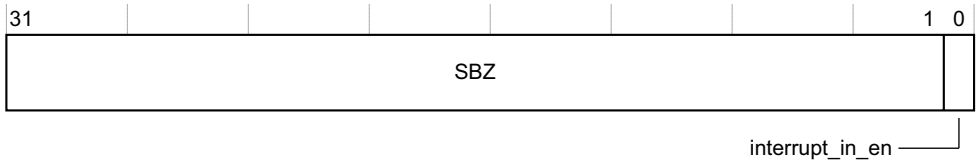


Figure 4-5 enable_s Register and enable_ns Register bit assignments

Table 4-3 shows the enable_ns Register bit assignments.

Table 4-3 enable_ns Register bit assignments

Bits	Name	Description
[31:1]	-	SBZ.
[0]	interrupt_in_en	Controls if the distributor responds to changes in the status of its interrupt inputs, **SPI** or **PPI**: 0 = disables all Non-secure interrupts control bits in the distributor from changing state because of any external stimulus change that occurs on the corresponding **SPI** or **PPI** signals. **Note** The distributor still responds to Non-secure interrupts for: • SGIs that are generated by using the sgi_trigger Register • PPIs and SPIS that are generated by using the pending_set Register, or cleared by using the pending_clr Register. 1 = enables the distributor to update register locations for Non-secure interrupts.

Table 4-4 shows the enable_s register bit assignments.

Table 4-4 enable_s Register bit assignments

Bits	Name	Description
[31:1]	-	SBZ.
[0]	interrupt_in_en	Controls if the distributor responds to changes in the status of its interrupt inputs, **SPI** or **PPI**: 0 = disables all Secure interrupt control bits in the distributor from changing state because of any external stimulus change that occurs on the corresponding **SPI** or **PPI** signals. **Note** The distributor still responds to Secure interrupts for: • SGIs that are generated by using the sgi_trigger Register • PPIs and SPIs that are generated by using the pending_set Register, or cleared by using the pending_clr Register. 1 = enables the distributor to update register locations for Secure interrupts.

Note

If you disable the distributor when it contains pending interrupts then you might get spurious interrupts after you re-enable the distributor. This occurs when the Cortex-A9 processor interface processes a pending request and immediately after this you disable the distributor and therefore the distributor cannot register the change in state of the interrupt.

4.7.5 Interrupt Controller Type Register

The ic_type Register characteristics are:

Purpose provides information about the configuration of the Interrupt Controller.

Usage constraints There are no usage constraints.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in *Distributor register map* on page 4-15.

Figure 4-6 on page 4-21 shows the ic_type Register bit assignments.

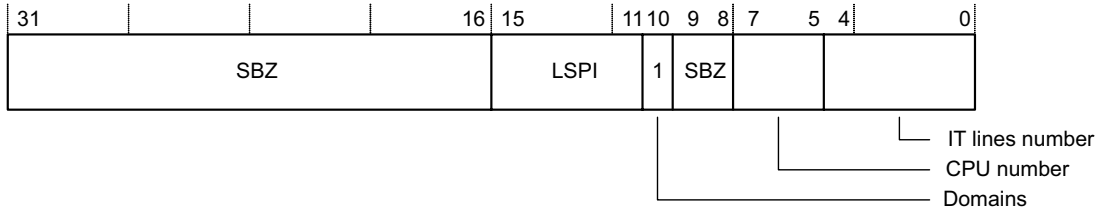


Figure 4-6 Interrupt Controller Type Register

Table 4-5 shows the ic_type bit assignments.

Table 4-5 ic_type Register bit assignments

Bits	Name	Description
[31:16]	-	Reserved
[15:11]	LSPI	Returns the number of *Lockable Shared Peripheral Interrupts* (LSPIs) that the controller contains. The encoding is: b11111 = 31 LSPIs, on **INTID[30:0]** mapped on IDs 62-32. When **CFGSDISABLE** is HIGH then the interrupt controller prevents writes to any register locations that control the operating state of an LSPI. See *Using CFGSDISABLE* on page 4-12.
[10]	Domains	Returns the number of security domains that the controller contains: 1 = the controller contains two security domains. This bit always returns the value one.

Table 4-5 ic_type Register bit assignments (continued)

Bits	Name	Description
[9:8]	-	SBZ
[7:5]	CPU number	The encoding is: b000 the Cortex-A9 design contains one Cortex-A9 processor. b001 the Cortex-A9 design contains two Cortex-A9 processors. b010 the Cortex-A9 design contains three Cortex-A9 processors. b011 the Cortex-A9 design contains four Cortex-A9 processors. b1xx: Reserved for future extensions.
[4:0]	SPI lines number	The encoding is: b00000 = the distributor provides 32 interrupts[a], no external interrupt lines. b00001 = the distributor provides 64 interrupts, 32 external interrupt lines. b00010 = the distributor provides 96 interrupts, 64 external interrupt lines. b00011 = the distributor provide 128 interrupts, 96 external interrupt lines. b00100 = the distributor provides 160 interrupts, 128 external interrupt lines. b00101 = the distributor provides 192 interrupts, 160 external interrupt lines. b00110 = the distributor provides 224 interrupts, 192 external interrupt lines. b00111 = the distributor provides 256 interrupts, 224 external interrupt lines. All other values are Reserved for future extension.

a. The distributor always uses interrupts of IDs 0 to 31 to control any SGIs and PPIs that the Interrupt Controller might contain.

4.7.6 Distributor Implementer Identification Register

The dist_ident Register characteristics are:

Purpose provides information about the implementer and the revision of the controller.

Usage constraints There are no usage constraints.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-2 on page 4-17.

Figure 4-7 on page 4-23 shows the dist_ident register bit assignments.

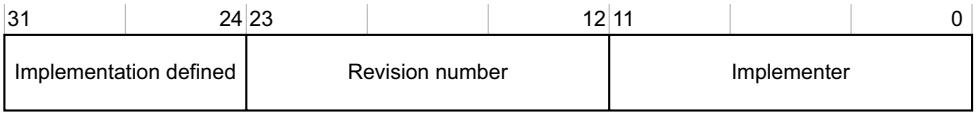


Figure 4-7 Distributor Implementer Identification Register bit assignments

Table 4-6 shows the dist_ident Register bit assignments.

Table 4-6 Distributor Implementer Identification Register bit assignment

Bits	Name	Description
[31:24]	Implementation version	For this implementation the value is `0x01`.
[23:12]	Revision number	Returns the revision number of the controller.This is r0p0, value `0x000`.
[11:0]	Implementer	Implementer number This is `0x43B`.

4.7.7 Interrupt security registers

The int_ns Register characteristics are:

Purpose controls the security level of an interrupt, to be either secure or non-secure.

Usage constraints You can only access this register with secure read or secure write accesses

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-2 on page 4-17.

Figure 4-8 on page 4-24 shows the int_ns Register bit assignments.

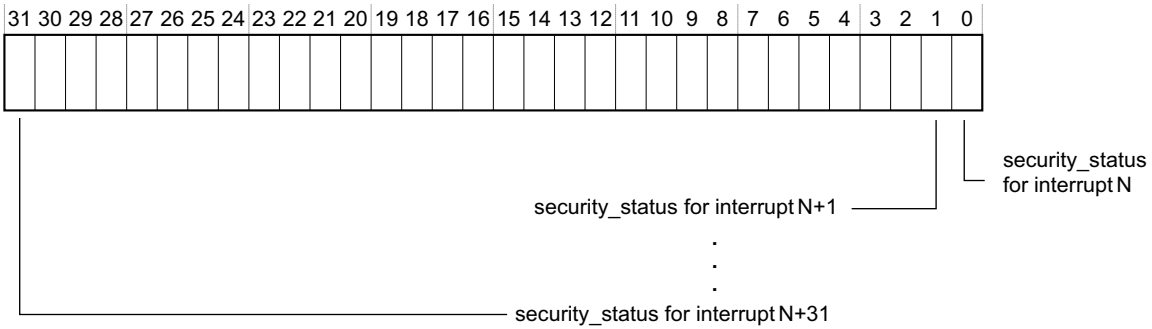


Figure 4-8 int_ns Register bit assignments

Table 4-7 shows the int_ns register bit assignments

Table 4-7 int ns Register bit assignment

Bits	Name	Description
[31:0]	security_status	Set the appropriate bit to change the security level for the corresponding interrupt to secure. Use as: **Bit [N]** = 0 Interrupt N is secure. **Bit [N]** = 1 Interrupt N is non-secure. **Note** You can only access this register using secure reads and secure writes. If you perform a non-secure read, the controller returns `0x00000000`. The controller ignores Non-secure writes.

If you dynamically change the security level of an interrupt then you must take care to flush the controller of all status information for that interrupt, prior to changing its security level.

Interrupts 0 to 31 fields are banked for each Cortex-A9 processor, meaning that Cortex-A9 processors can read and write different values from these registers.

For locations where interrupt IDs are not implemented then the controller:

- ignores writes to these IDs
- returns 0 when it reads from these IDs.

Figure 4-9 on page 4-25 shows the address map that the controller provides, to enable you to configure the security levels for each interrupt type.

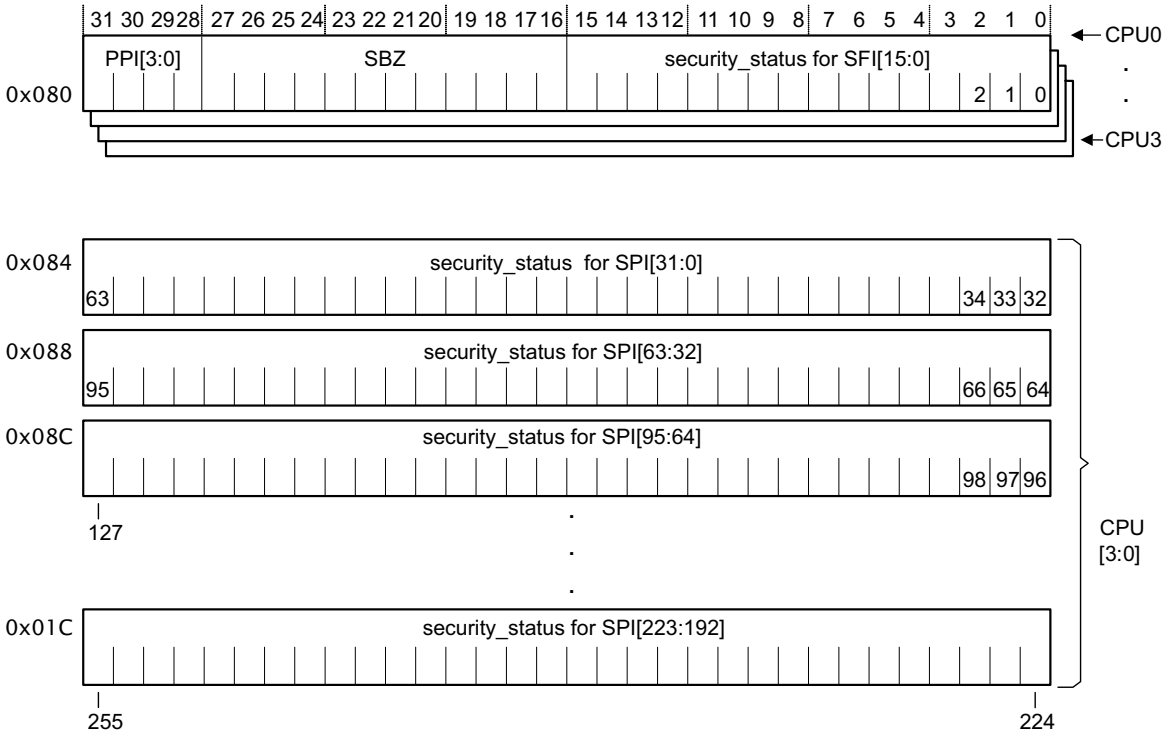


Figure 4-9 Interrupt Security Registers address map

In Figure 4-9:

- The distributor provides a register at address offset `0x080`. This register contains the values for the PPIs and STIs for the corresponding Cortex-A9 processor interface.
- The distributor provides up to 8 registers. If you configure the Interrupt Controller to use fewer than 224 SPIs, then it reduces the number of registers accordingly. For locations where interrupts are not implemented then the distributor:
  - — ignores writes to these corresponding bits
  - — returns 0 when it reads from these bits.

Each Cortex-A9 processor accesses its own register for the PPIs and SGIs.

4.7.8 Enable set registers

The enable_set Register characteristics are:

Purpose Controls an enable bit for each interrupt in the interrupt distributor.

Usage constraints There are no usage constraints.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-2 on page 4-17.

Figure 4-10 shows the enable_set Register bit assignments.

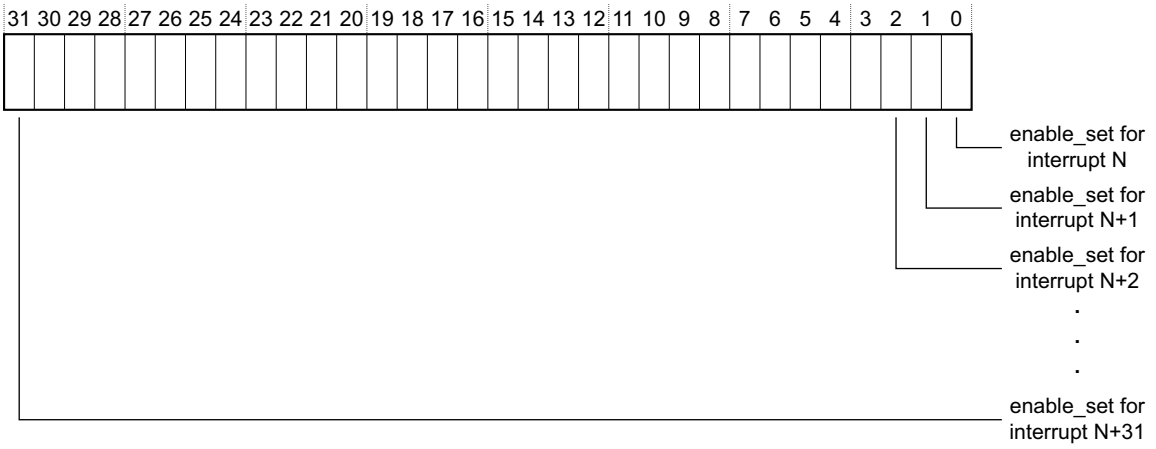


Figure 4-10 enable_set registers bit assignment

Table 4-8 shows the enable_set Register bit assignments.

Table 4-8 enable_set Register bit assignments

Bit	Name	Description
[31:0]	enable_set	Set the appropriate bit to enable the corresponding interrupt. Read as: **Bit [*N*] = 0** Interrupt is disabled. The distributor does not forward this interrupt to any Cortex-A9 processor interfaces, when it becomes pending. **Bit [*N*] = 1** Interrupt is enabled. When this interrupt is pending the distributor forwards it to the appropriate Cortex-A9 processor interface. Writes: **Bit [*N*] = 0** No effect. Use the enable_clr Register to set bit [*N*] to 0. **Bit [*N*] = 1** Enables interrupt. **Note** The interrupt that *N* refers to depends on its bit position and the base address offset of the enable_set Register as Figure 4-11 on page 4-28 shows.

Figure 4-11 on page 4-28 shows the address map that the distributor provides for the SGI, PPI, and SPI interrupts.

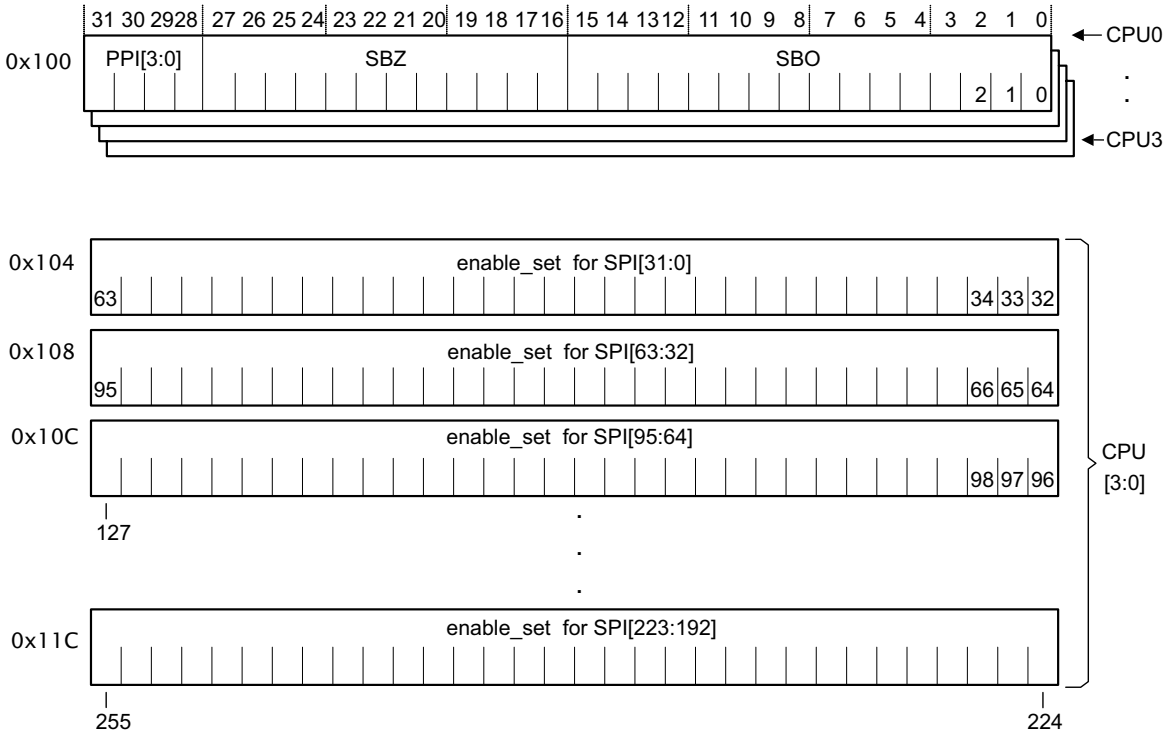


Figure 4-11 Enable set registers address map

In Figure 4-11:

- The distributor does not provide registers for interrupts < 16 because SGIs are always enabled.
- The distributor provides a register at address offset `0x100`. This register contains the values for the PPIs and SGIs for the corresponding Cortex-A9 processor interface.
- The distributor provides up to 8 registers. If you configure the Interrupt Controller to use fewer than 224 SPIs, then it reduces the number of registers accordingly. For locations where interrupts are not implemented then the distributor:
  - — ignores writes to the corresponding bits
  - — returns 0 when it reads from these bits.

4.7.9 Enable clear registers

The enable_clr Register characteristics are:

Purpose controls the disabling of an interrupt.

Usage constraints There are no usage constraints.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-2 on page 4-17.

Figure 4-12 shows the enable_clr Register bit assignments.

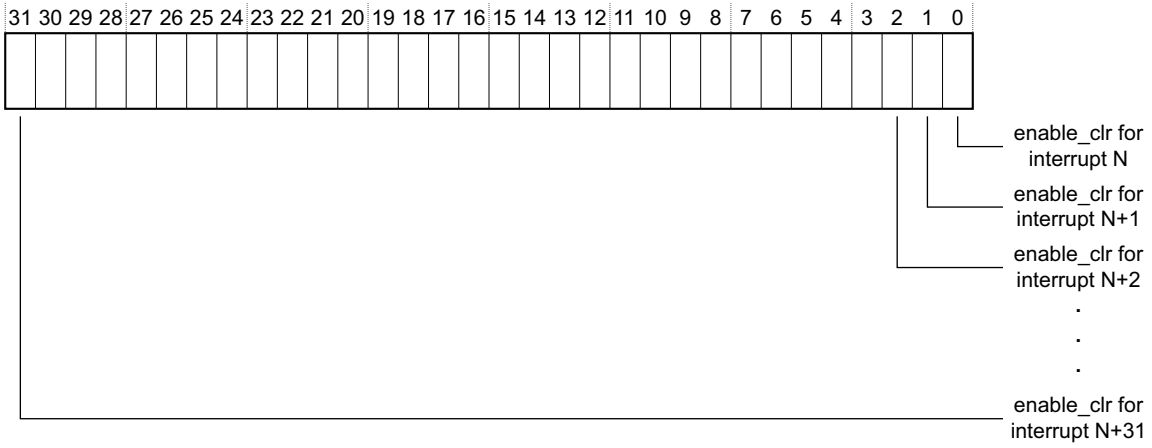


Figure 4-12 enable_clr Registers bit assignment

Table 4-9 shows the register bit assignments.

Table 4-9 enable_clr Register bit assignments

Bits	Name	Description
[31:0]	enable_clr	Set the appropriate bit to disable the corresponding interrupt. Reads: **Bit [*N*] = 0** Interrupt is disabled. The distributor does not forward this interrupt to any Cortex-A9 processor interfaces, when it becomes pending. **Bit [*N*] = 1** Interrupt is enabled. When this interrupt is pending the distributor forwards it to the appropriate Cortex-A9 processor interface. Writes: **Bit [*N*] = 0** No effect. Use the enable_set Register to set bit [*N*] to 1. **Bit [*N*] = 1** Disables interrupt. **Note** The interrupt that *N* refers to depends on its bit position and the base address offset of the enable_clr Register as Figure 4-13 on page 4-31 shows. You cannot disable SGIs.

Figure 4-13 on page 4-31 shows the address map that the distributor provides for the SGI, PPI, and SPI interrupts.

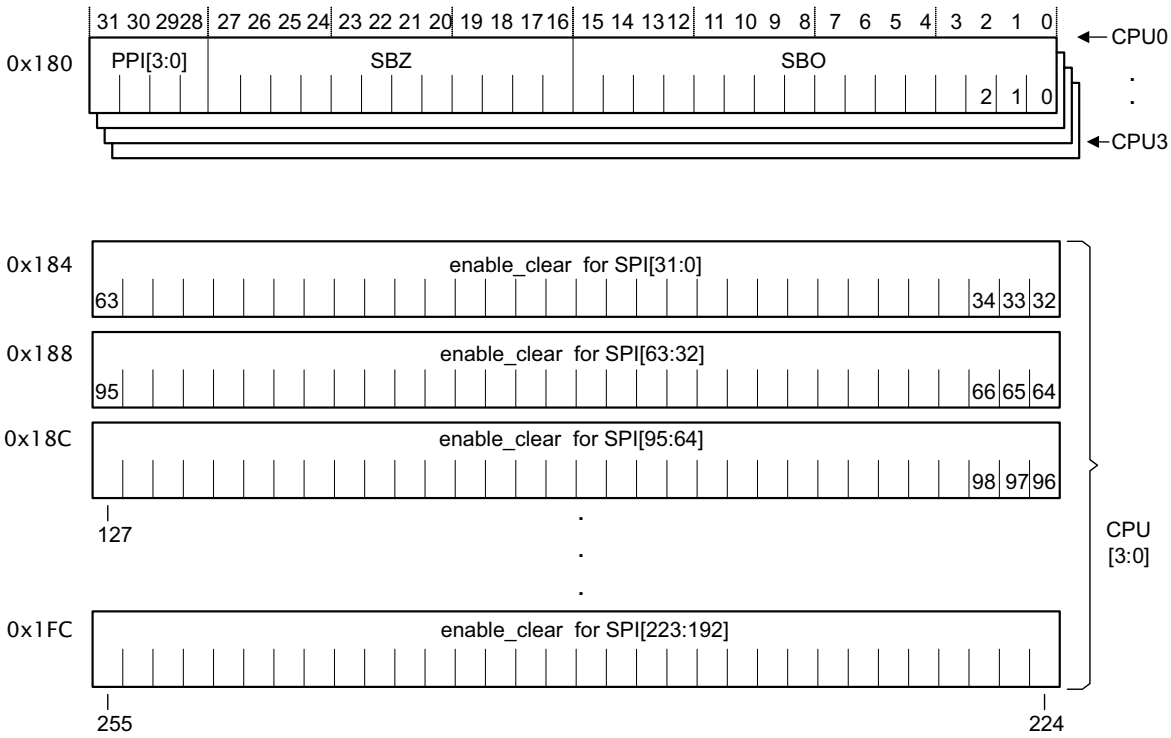


Figure 4-13 Enable clear registers address map

The distributor does not provide configuration bits for interrupts < 16 because STIs are always enabled.

The distributor provides a register at address offset `0x180`. This register contains the values for the PPIs and STIs for the corresponding Cortex-A9 processor interface.

The distributor provides up to 8 registers. If you configure the Interrupt Controller to use fewer than 224 SPIs, then it reduces the number of registers accordingly. For locations where interrupts are not implemented then the distributor:

- ignores writes to the corresponding bits
- returns 0 when it reads from these bits.

4.7.10 Pending set registers

The pending_set Register characteristics are:

Purpose Enables you to set or read the status of interrupts:

- writes enable you to set an interrupt to the pending, or active and pending state.
- reads provide the status of the interrupts that are pending

Usage constraints There are no usage constraints.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-2 on page 4-17.

Figure 4-14 shows the pending_set Register bit assignments.

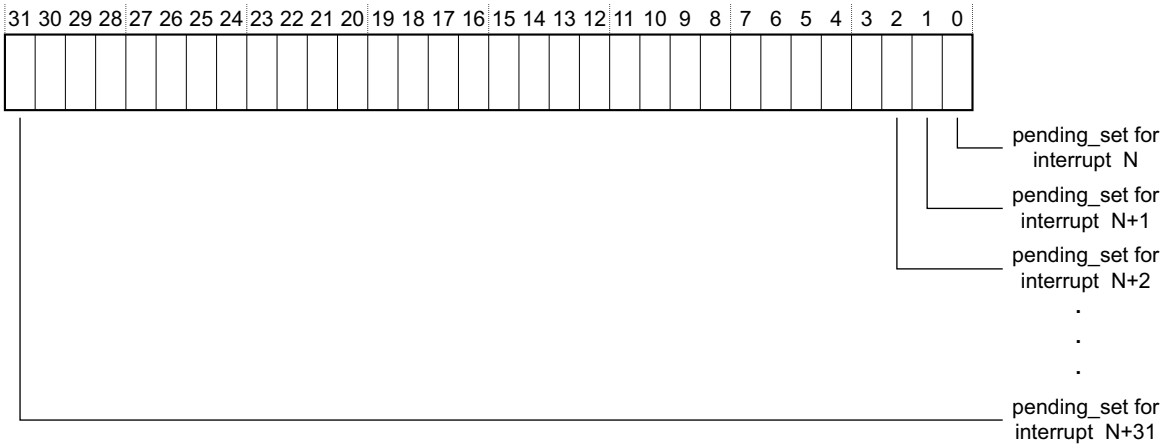


Figure 4-14 pending_set registers bit assignments

Figure 4-15 on page 4-33 shows the pending_set Register address map.

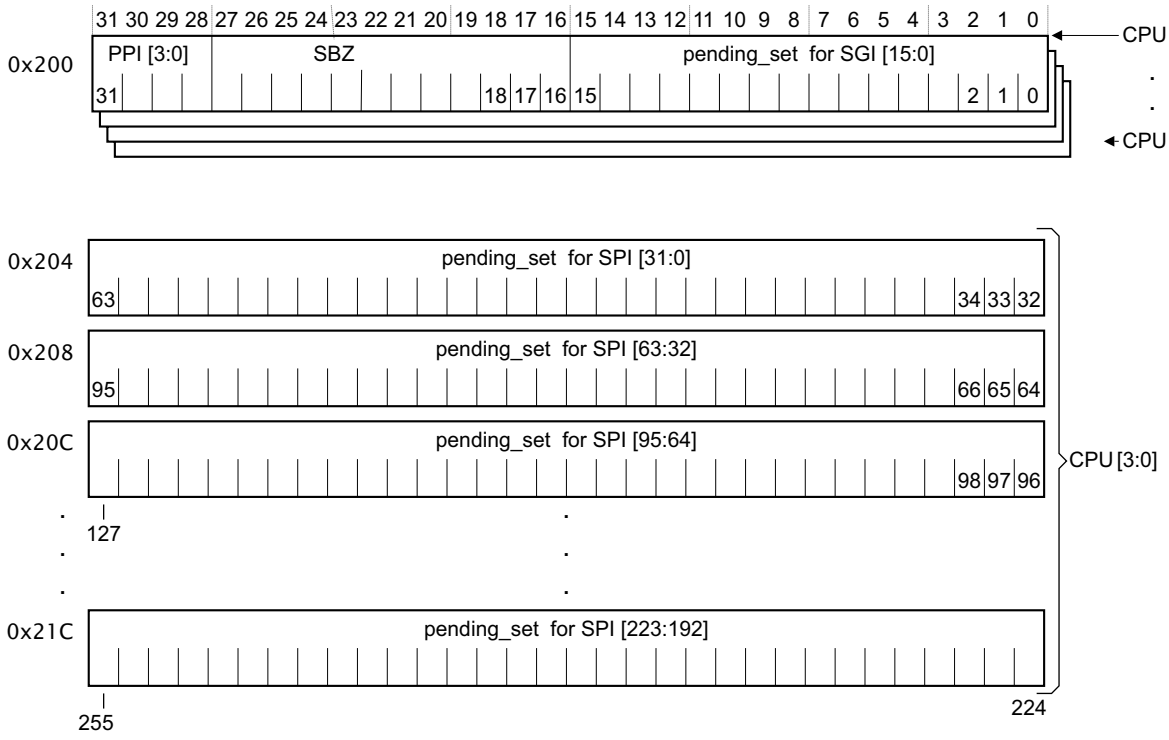


Figure 4-15 Pending set registers address map

In Figure 4-15:

- The values for the SGIs are read-only. However, the distributor updates these bits using the information that the sti_trigger Register contains. See *Software Generated Interrupt Register* on page 4-50.
- The distributor provides a register at address offset `0x200`. This register contains the values for the PPIs and SGIs for the corresponding Cortex-A9 processor interface.
- The distributor provides up to 8 registers. If you configure the Interrupt Controller to use fewer than 224 SPIs then it reduces the number of registers accordingly. For locations where interrupts are not implemented then the distributor:
  - — ignores writes to the corresponding bits
  - — returns 0 when it reads from these bits.

4.7.11 Pending clear registers

The pending_clr Register characteristics are:

Purpose Enables you to clear or read the status of interrupts:

- writes enable you to clear an interrupt to the pending, or active and pending state.
- reads provide the status of the interrupts that are pending

Usage constraints There are no usage constraints.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-2 on page 4-17.

Figure 4-16 shows the pending_clr Register bit assignments.

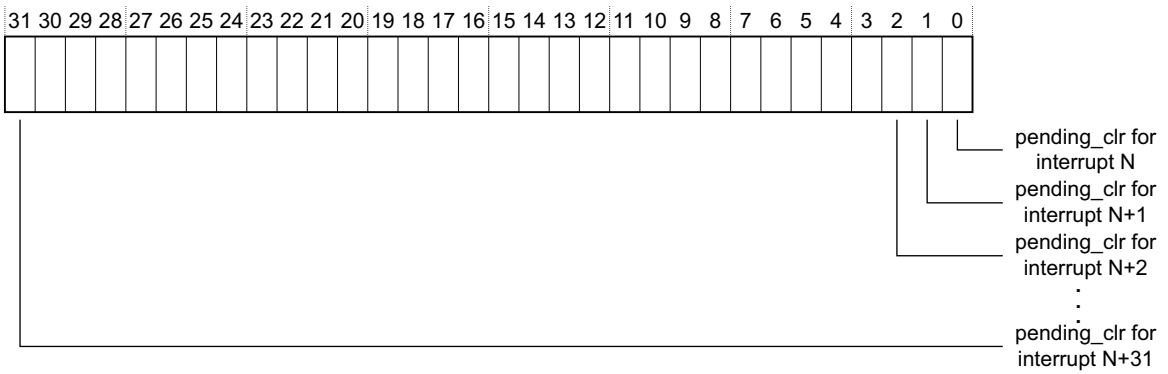


Figure 4-16 pending_clr Register bit assignments

Table 4-10 shows the register bit assignments.

Table 4-10 pending_clr Register bit assignments

Bits	Name	Description
[31:0]	pending_clr	Set the appropriate bit to clear the pending state of the corresponding interrupt. Read as: **Bit [*N*] = 0** Interrupt is not pending, but is either inactive or active. **Bit [*N*] = 1** Interrupt is either pending or active and pending. Writes: **Bit [*N*] = 0** No effect. Use the pending_set Register to set bit [*N*] to 1. **Bit [*N*] = 1** Clears interrupt from the pending state. **Note** The interrupt that *N* refers to depends on its bit position and the base address offset of the pending_clr Register as Figure 4-17 on page 4-36 shows. The distributor is unable to clear an interrupt from the pending state when this interrupt is configured as level-sensitive and is still asserted by a peripheral. The distributor ignores writes for SGI [15:0].

Figure 4-17 on page 4-36 shows the address map that the distributor provides for the SGI, PPI, and SPI interrupts.

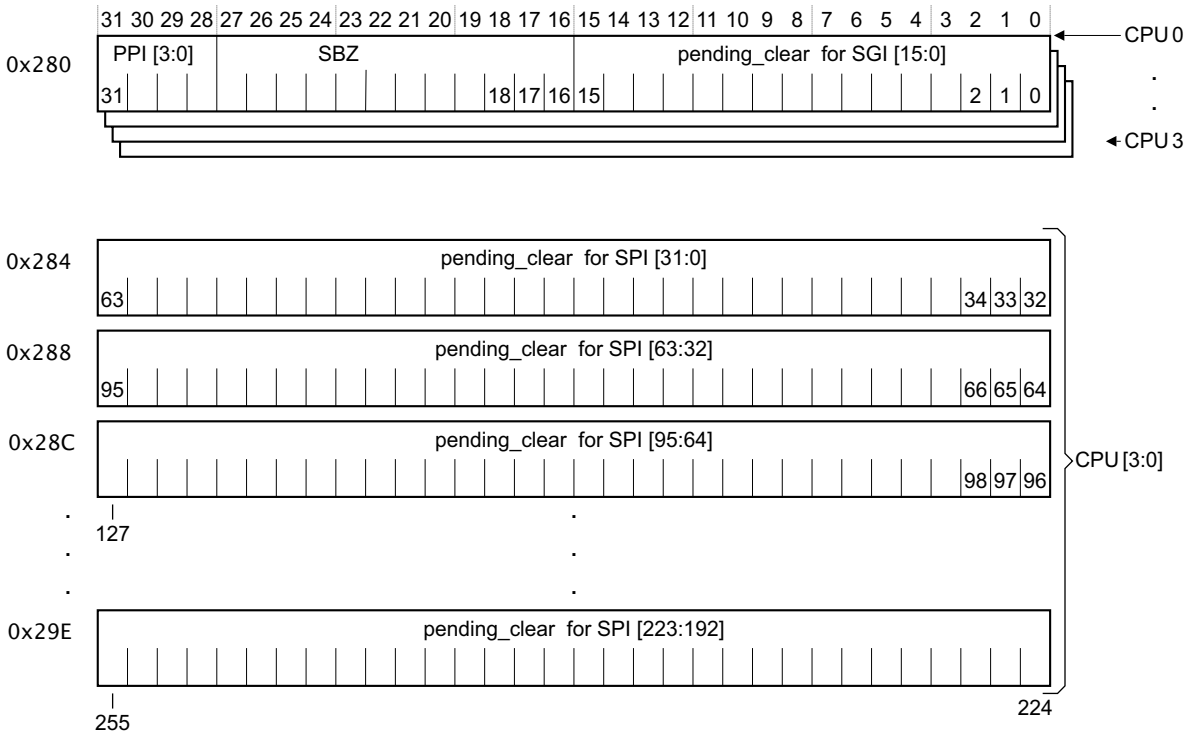


Figure 4-17 Pending clear registers address map

In Figure 4-17:

- The values for the SGIs are read-only. The distributor updates these bits using the information that the sti_trigger Register contains.
- The distributor provides a register at address offset `0x280`. This register contains the values for the PPIs and SGIs for the corresponding Cortex-A9 processor interface.
- The distributor provides up to 8 registers. If you configure the Interrupt Controller to use fewer than 224 SPIs then it reduces the number of registers accordingly. For locations where interrupts are not implemented then the distributor:
  - — ignores writes to the corresponding bits
  - — returns 0 when it reads from these bits.

4.7.12 Active status registers

The active_status Register characteristics are:

Purpose Provides the active status of interrupts.

Usage constraints There are no usage constraints.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-2 on page 4-17.

Figure 4-18 shows the active_status Register bit assignments.

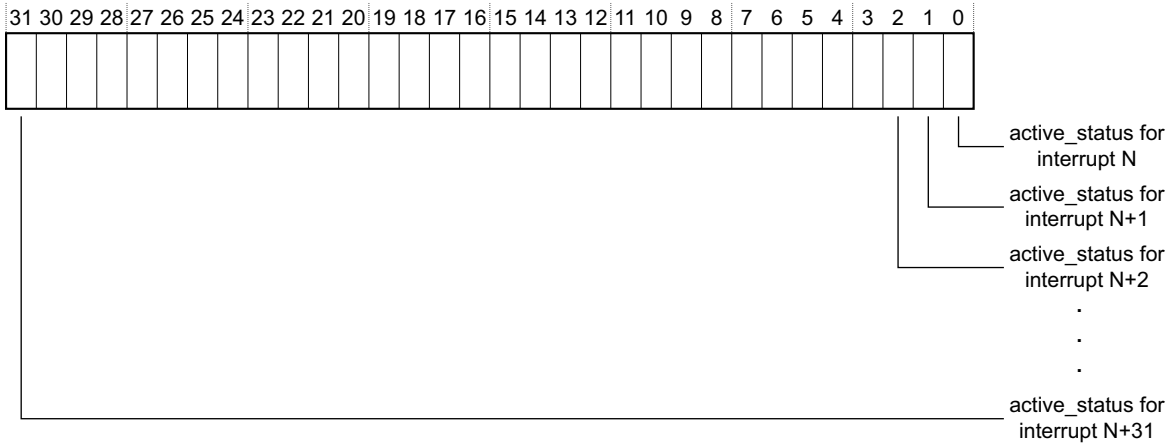


Figure 4-18 active_status register bit assignments

Table 4-11 shows the active_status bit assignments.

Table 4-11 active_status Register bit assignments

Bits	Name	Description
[31:0]	active_status	Returns the active status of the corresponding interrupt. Read as: **Bit [*N*] = 0** Interrupt is not active and is therefore either inactive or pending. **Bit [*N*] = 1** Interrupt is either active or active and pending. **Note** The interrupt that *N* refers to depends on its bit position and the base address offset of the active_status Register as Figure 4-19 on page 4-38 shows.

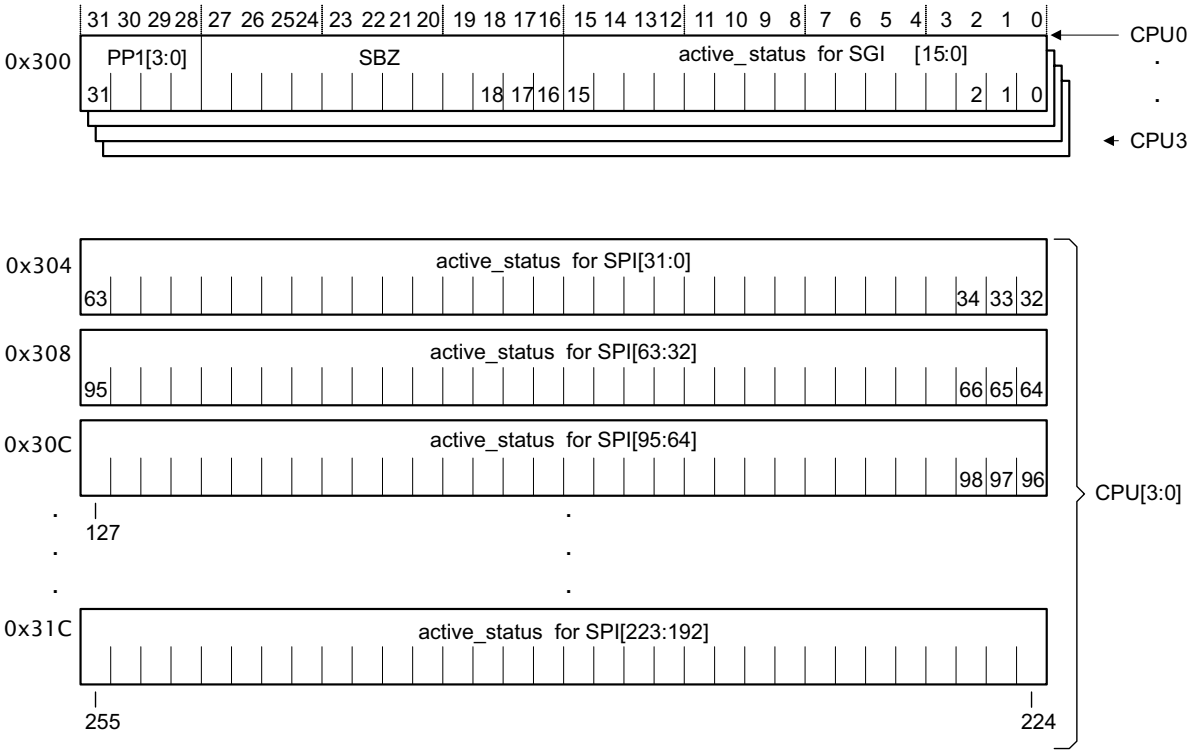


Figure 4-19 Active status registers address map

In Figure 4-19:

- The distributor provides a register at address offset `0x300`. This register contains the values for the PPIs and SGIs for the corresponding Cortex-A9 processor interface.
- The distributor provides up to 8 registers. If you configure the Interrupt Controller to use fewer than 224 SPIs then it reduces the number of registers accordingly. For locations where interrupts are not implemented then the distributor returns 0.

4.7.13 Priority level registers

The priority_level Register characteristics are:

Purpose Enables you to set or read the priority level of an interrupt:

- writes enable you to set the priority level of an interrupt
- reads a register provide the priority level of an interrupt.

Usage constraints The distributor supports byte accesses to these 32-bit registers to enable you to update the priority level for a single interrupt.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-2 on page 4-17.

Figure 4-20 shows the priority_level Register bit assignments. See also *Priority formats* on page 4-5.

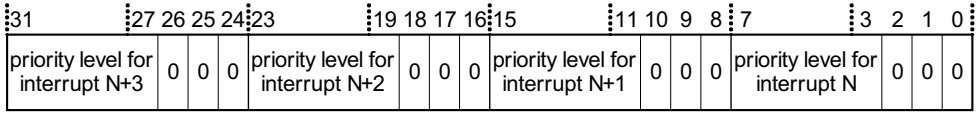


Figure 4-20 Interrupt priority registers format for 32 priority levels

Table 4-12 shows the priority_level Register bit assignments.

Table 4-12 Interrupt bit priority registers bit assignments

Bits	Name	Description
[31:27]	priority level for interrupt n+3	Set the priority level of an interrupt to a priority level that your interrupt controller provides. The priority levels are: `0x00` = highest priority level `0x01` = second highest priority level `0x02` = third highest priority level . . . `0x1F` = lowest priority level.
[26:24]	SBZ	
[23:19]	priority level for interrupt n+2	
[18:16]	SBZ	
[15:11]	priority level for interrupt n+1	
[10:8]	SBZ	
[7:3]	priority level for interrupt n	
[2:0]	SBZ	-

Figure 4-21 on page 4-40 shows the address map that the controller provides for the SGI, PPI, and SPI interrupts.

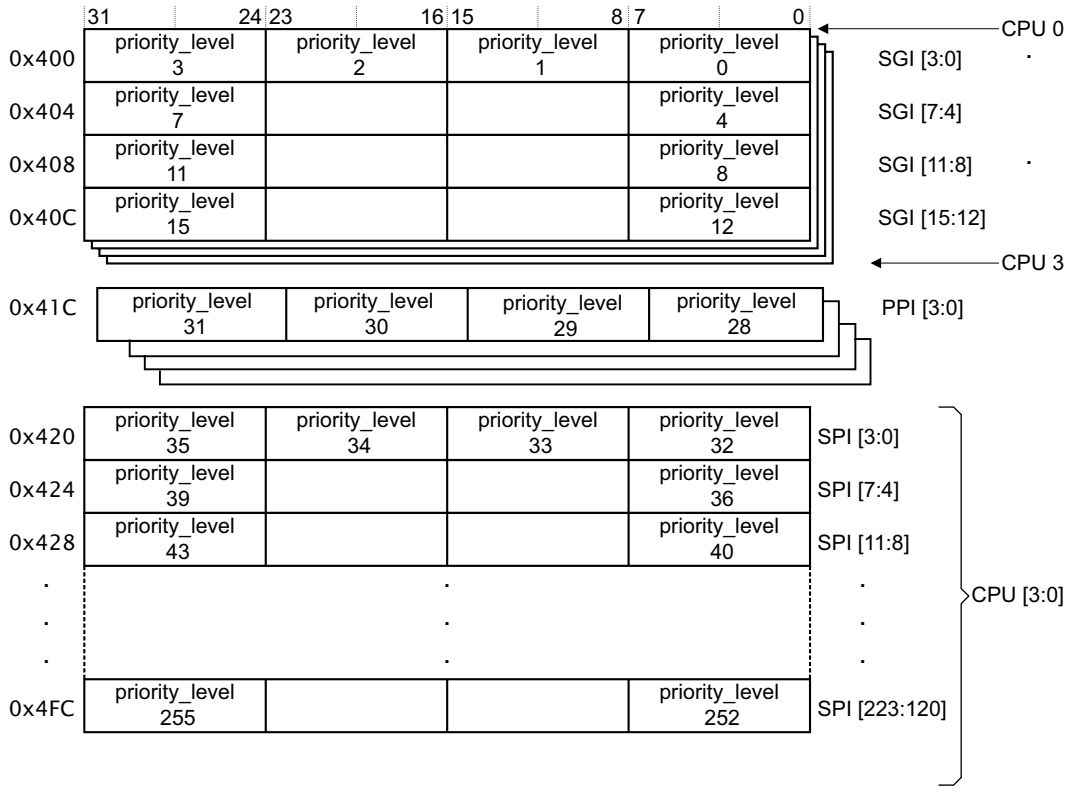


Figure 4-21 priority-level Registers address map

In Figure 4-21:

- The distributor provides registers at address offset `0x400-0x41C` that contain the values for the PPIs and STIs for the corresponding Cortex-A9 processor interface.
- The distributor provides up to 64 registers. If you configure the interrupt controller to use fewer than 224 SPIs then it reduces the number of registers accordingly. For locations where interrupts are not implemented then the distributor:
  - — ignores writes to the corresponding bits
  - — returns 0 when it reads from these bits.

4.7.14 SPI Target registers

The spi_target Register characteristics are:

Purpose Enables you to program how many of the four Cortex-A9 processor interfaces are sent a pending interrupt.

Usage constraints The distributor does not provide the spi_target Registers for single-processor systems because all interrupts can only have one destination. Under these circumstances, the distributor returns `0x00000000` and writes are ignored.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-2 on page 4-17.

Figure 4-22 shows the spi_target Register bit assignments.

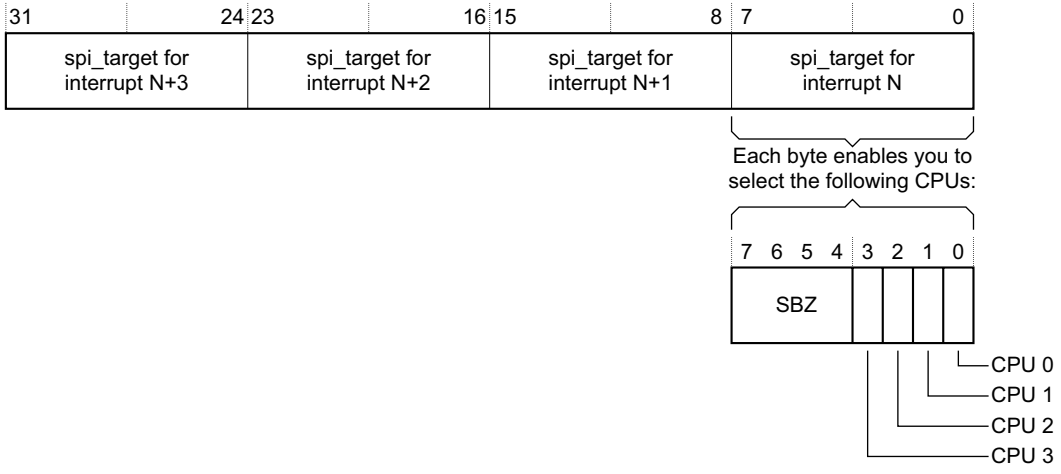


Figure 4-22 spi_target Register bit assignments

The distributor supports byte accesses to these 32-bit registers, to enable you to update the target Cortex-A9 processor list for a single SPI. Table 4-13 shows the register bit assignments.

Table 4-13 spi_target Register bit assignments

Bits	Name	Description
[31:24]	spi_target for interrupt N+3	Configures which of the four Cortex-A9 processors that the interrupt distributor routes an SPI to. The options are: **Bit [24]=1, Bit [16]=1, Bit [8]=1, Bit [0]=1** the distributor routes interrupt to CPU0. **Bit [25]=1, Bit [17]=1, Bit [9]=1, Bit [1]=1** the distributor routes interrupt to CPU1. **Bit [26]=1, Bit [18]=1, Bit [10]=1, Bit [2]=1** the distributor routes interrupt to CPU2. **Bit [27]=1, Bit [19]=1, Bit [11]=1, Bit [3]=1** the distributor routes interrupt to CPU3. **Note** The interrupt that *N* refers to, depends on its bit position and the base address offset of the spi_target Register as Figure 4-23 on page 4-43 shows.
[23:16]	spi_target for interrupt N+2	
[15:8]	spi_target for interrupt N+1	
[7:0]	spi_target for interrupt N	

If you change the target Cortex-A9 processors for an SPI that is pending then the distributor immediately uses the new information and changes the target Cortex-A9 processors for that SPI.

If you change the target Cortex-A9 processors for an SPI that is active and pending then the:

- distributor immediately reassigns the pending interrupt, using the new target Cortex-A9 processor information
- SPI remains active on the original target Cortex-A9 processors.

Figure 4-23 on page 4-43 shows the address map that the distributor provides for the SGI, PPI, and SPI interrupts.

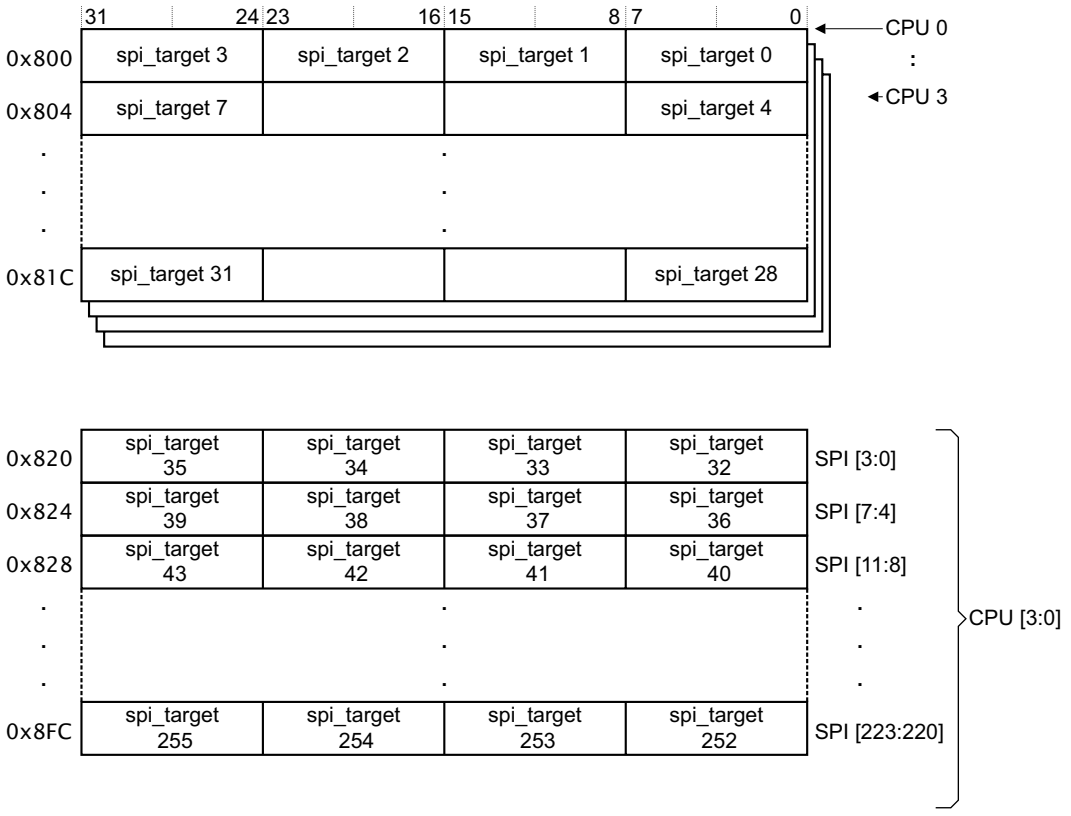


Figure 4-23 spi_target Register address map

In Figure 4-23:

- The distributor does not provide registers for interrupts < 32. The distributor returns the number of the Cortex-A9 processor that performs the read, otherwise the distributor returns `0x00`.
- For systems that support fewer than four Cortex-A9 processors, the distributor:
  - — ignores writes to non-implemented bits
  - — returns 0 when it reads from these non-implemented bits.
- The distributor provides up to 24 registers. If you configure the Interrupt Controller to use fewer than 224 SPIs then it reduces the number of registers accordingly. For locations where interrupts are not implemented then the distributor:
  - — ignores writes to the corresponding bits
  - — returns 0 when it reads from these bits.

For systems that support only one Cortex-A9 processor, all these Cortex-A9 processor spi_target Registers read as zero, and writes are ignored.

4.7.15 Interrupt Configuration Registers

The int_config Register characteristics are:

Purpose

- Programs an SPI to be either:
  - — active HIGH level-sensitive
  - — rising-edge sensitive.
- Reads the interrupt configuration of PPIs and SGIs.

Usage constraints ARM recommends that you only change the interrupt configuration when an SPI is disabled. Changing the configuration when an SPI is enabled might cause unpredictable behavior.

You must use the enable_clr Register to disable an SPI. See *Enable clear registers* on page 4-29. This register is byte accessible.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-2 on page 4-17.

Figure 4-24 shows the int_config Register bit assignments.

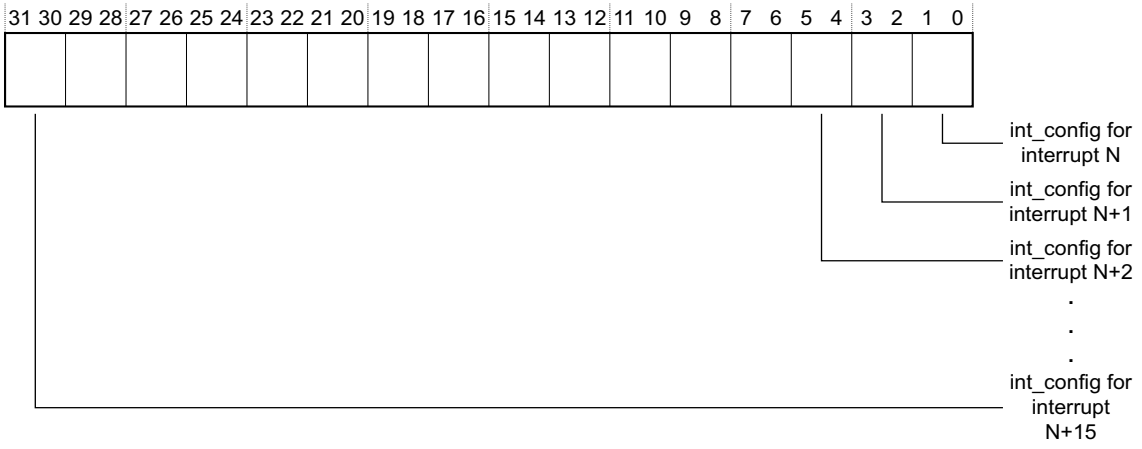


Figure 4-24 int_config Register bit assignments

Table 4-14 shows the int_config Register bit assignments.

Table 4-14 int_config Register bit assignments

Bits	Name	Description
[31:0]	int_config for interrupt N to interrupt N+15	Each bit-pair describes the interrupt configuration for an interrupt. The options for each pair depend on the interrupt type as follows: **SGI** The bits are read-only and a bit-pair always reads as b10. Therefore each interrupt is active HIGH level sensitive. **PPI** The bits are read-only **PPI[0] and [3]:b01** interrupt is active HIGH level sensitive. **PPI[1] and [2]:b11** interrupt is rising-edge sensitive. See Table 4-15 on page 4-48. **SPI** The LSB bit of a bit-pair is read-only and is always b1. You can program the MSB bit of the bit-pair to alter the triggering sensitivity as follows: **b01** interrupt is active HIGH level sensitive **b11** interrupt is rising-edge sensitive. **Note** The interrupt that *N* refers to, depends on its bit position and the base address offset of the int_config Register as Figure 4-25 on page 4-46 shows.

Figure 4-25 on page 4-46 shows the address map that the distributor provides for the SGI, PPI, and SPI interrupts.

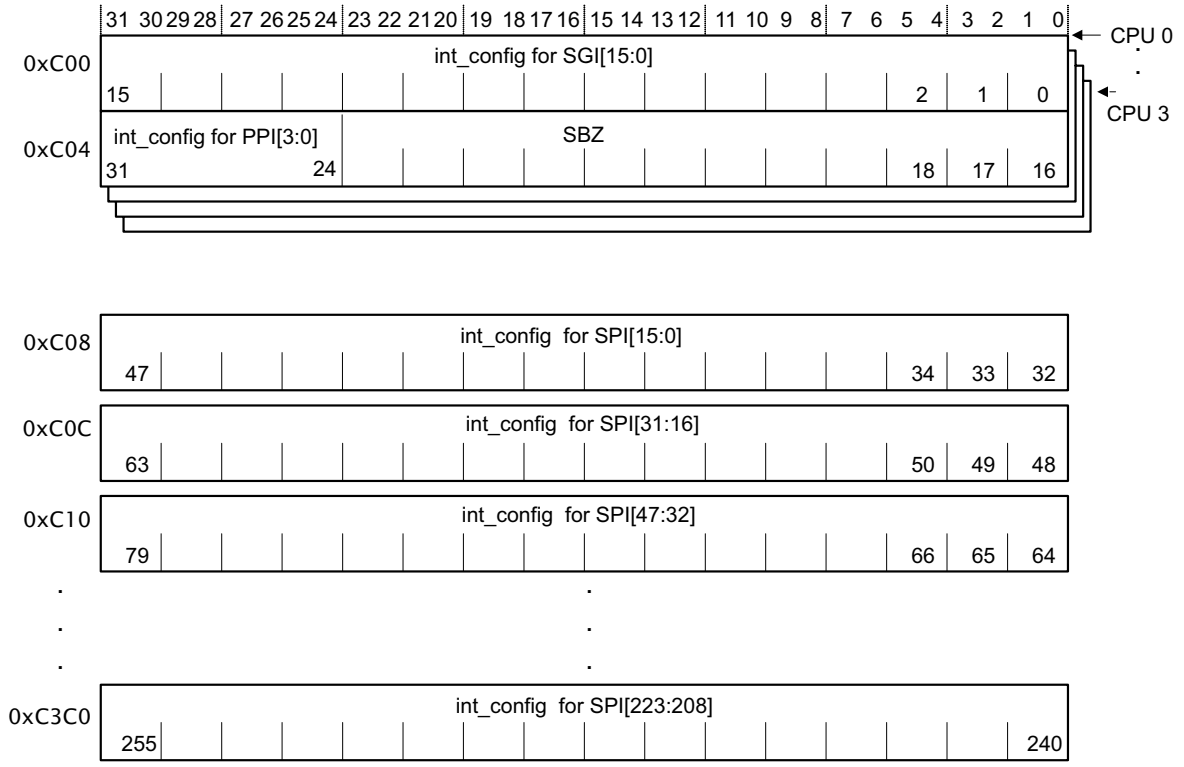


Figure 4-25 int_config Register address map

In Figure 4-25:

- The distributor returns b10, for each bit-pair, when you read the value for interrupts < 16.
- You cannot update the interrupt configuration for PPIs.
- The distributor provides a register at address offset `0xC04`. This register contains the values for the PPIs for the corresponding Cortex-A9 processor interface.
- The distributor provides up to 16 registers. If you configure the Interrupt Controller to use fewer than 224 SPIs then it reduces the number of registers accordingly. For locations where interrupts are not implemented then the distributor:
  - ignores writes to the corresponding bits
  - returns 0 when it reads from these bits.

4.7.16 PPI Status Register

The ppi_status Register characteristics are:

Purpose Enables a Cortex-A9 processor to access the status of the private inputs on the distributor:

- PPI(3) is for **nIRQ<n>**
- PPI(2) is for watchdog interrupts
- PPI(1) is for timer interrupts
- PPI(0) is for **nFIQ<n>**.

Usage constraints A Cortex-A9 processor can only read the status of its own **PPI** and therefore cannot read the status of **PPI** for other Cortex-A9 processors.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-2 on page 4-17.

Figure 4-26 shows the ppi_status Register bit assignments.

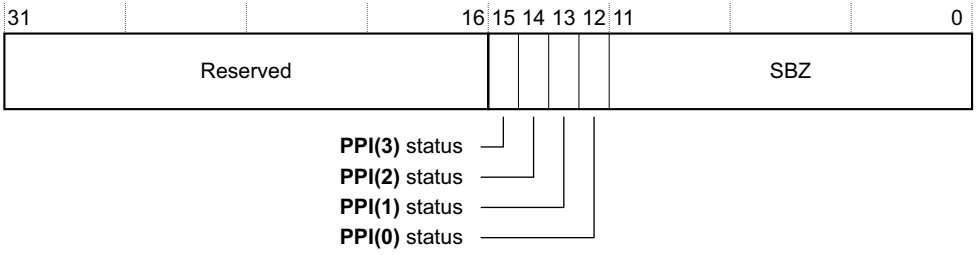


Figure 4-26 ppi_status Register bit assignments

Table 4-15 lists the register bit assignments.

Table 4-15 ppi_status Register bit assignments

Bits	Name	Description
[31:16]	-	Reserved
[15:12]	ppi_status	Returns the status of the **PPI(3:0)** inputs on the distributor: PPI(3) and PPI(0) are active LOW, so if the interrupt line is set to 0, then the corresponding bit is 1. For PPI(2) and PPI(1) if the interrupt line is active then the corresponding bit is 1. **Note** These bits return the actual status of the **PPI(3:0)** signals. The pending_set and pending_clr Registers also can provide the **PPI(3:0)** status but because you can write to these registers then they might not contain the actual status of the **PPI(3:0)** signals.
[11:0]	-	SBZ

4.7.17 SPI Status Registers

The spi_status Register characteristics are:

Purpose Enables a Cortex-A9 processor to access the status of the **INT[31:0]** inputs on the distributor.

Usage constraints There are no usage constraints.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-2 on page 4-17.

Figure 4-27 on page 4-49 shows the spi_status Register bit assignments.

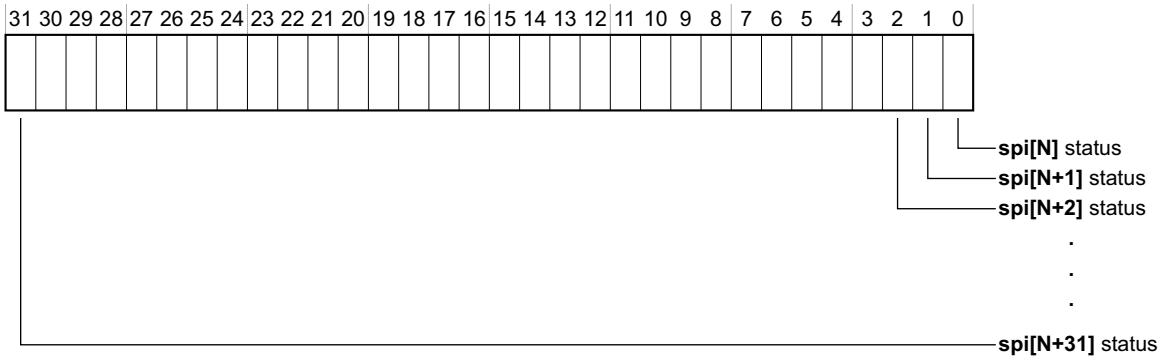


Figure 4-27 spi_status Register bit assignments

Table 4-16 shows the spi_status Register bit assignments.

Table 4-16 spi_status Register bit assignments

Bits	Name	Description
[31:0]	spi_status	Returns the status of the **INT[31:0]** inputs on the distributor: **Bit [*X*] = 0** **INT[X]** is LOW **Bit [*X*] = 1** **INT[X]** is HIGH. **Note** The **INT** that *X* refers to depends on its bit position and the base address offset of the spi_status Register as Figure 4-28 on page 4-50 shows. These bits return the actual status of the **INT** signals. The pending_set and pending_clr Registers can also provide the **INT** status but because you can write to these registers then they might not contain the actual status of the **INT** signals.

Figure 4-28 on page 4-50 shows the address map that the distributor provides for the SPIs.

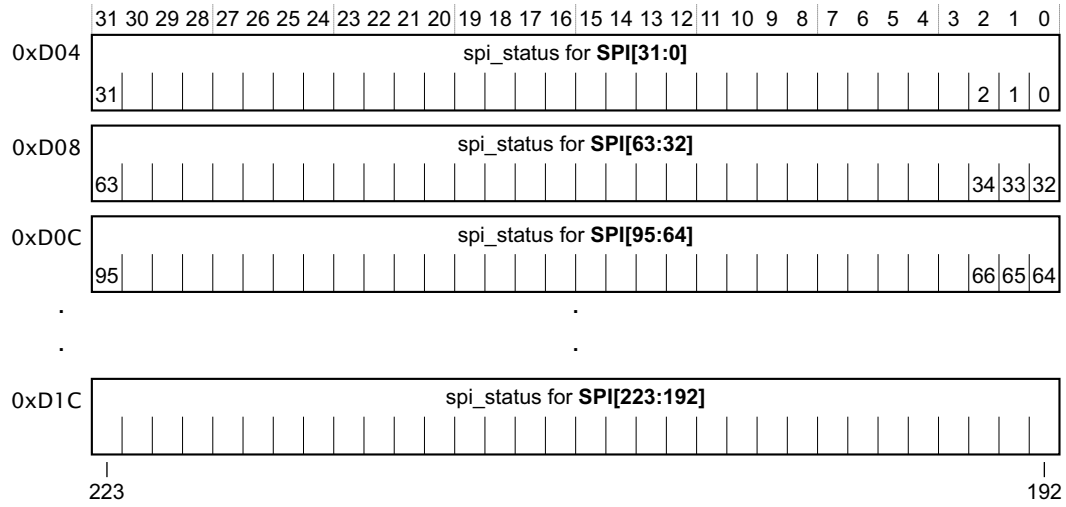


Figure 4-28 spi_status Register address map

In Figure 4-28 the values for the SPIs are read-only. The distributor provides a register at address offset `0xD04`. This register contains the values for the SPIs for the corresponding Cortex-A9 processor interface. The distributor provides up to 7 registers. If you configure the Interrupt Controller to use fewer than 224 SPIs then it reduces the number of registers accordingly. For locations where interrupts are not implemented then the distributor:

- ignores writes to the corresponding bits
- returns 0 when it reads from these bits.

4.7.18 Software Generated Interrupt Register

The sgi_trigger Register characteristics are:

Purpose Controls the issue of software interrupts.

Usage constraints There are no usage constraints.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-2 on page 4-17.

Figure 4-29 on page 4-51 shows the sgi_trigger Register bit assignments.

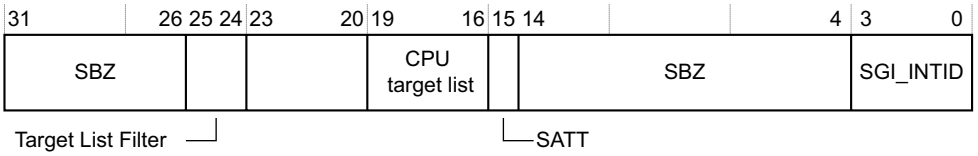


Figure 4-29 sgi_trigger Register bit assignments

Table 4-17 shows the sgi_trigger Register bit assignments.

Table 4-17 sgi_trigger Register bit assignments

Bits	Name	Description
[31:26]	-	SBZ.
[25:24]	Target List Filter	Controls the filtering that the distributor performs on software interrupts as follows: b00 = the distributor sends the interrupt to Cortex-A9 processors that the CPU Target List field specifies[a] b01 = the distributor sends the interrupt to all Cortex-A9 processors except the Cortex-A9 processor that requested the interrupt b10 = the distributor sends the interrupt to the Cortex-A9 processor that requested the interrupt b11 = reserved.
[23:20]	-	SBZ
[19:16]	CPU Target List	Selects which Cortex-A9 processors the distributor signals a software interrupt to. Each bit selects the following Cortex-A9 processor interface: **Bit [19] = 1** the distributor signals a pending interrupt to Cortex-A9 processor interface 3. **Bit [18] = 1** the distributor signals a pending interrupt to Cortex-A9 processor interface 2. **Bit [17] = 1** the distributor signals a pending interrupt to Cortex-A9 processor interface 1. **Bit [16] = 1** the distributor signals a pending interrupt to Cortex-A9 processor interface 0. You can set one or more bits, to signal an SGI to as many Cortex-A9 processors as you require.

Table 4-17 sgi_trigger Register bit assignments (continued)

Bits	Name	Description
[15]	SATT	You can only access the *Security ATTribute* (SATT) bit using a secure write. Use as: 0 = the distributor uses a secure interrupt for the interrupt that STI_INTID specifies 1 = the distributor uses a non-secure interrupt for the interrupt that STI_INTID specifies. This value must match the security setting for the interrupt in the receiving processor interface.
[14:4]	-	SBZ.
[3:0]	SGI_INTID	Set this to the ID of the software interrupt that the distributor signals to the Cortex-A9 processors, that the CPU Target List field and Target Filter List field define. Write as: b0000 = the distributor signals SGI0 to the selected Cortex-A9 processors b0001 = the distributor signals SGI1 to the selected Cortex-A9 processors b0010 = the distributor signals SGI2 to the selected Cortex-A9 processors b0011 = the distributor signals SGI3 to the selected Cortex-A9 processors . . . b1111 = the distributor signals SGI15 to the selected Cortex-A9 processors.

a. If both the Target List Filter and the CPU Target List are `0x00`, then the distributor does not permit the interrupt that SGI_INTID specifies to become pending on any Cortex-A9 processor.

4.7.19 Peripheral Identification Registers

The periph_id_[4:0] Registers provide information about the configuration of the peripheral. Table 4-2 on page 4-17 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*
- *periph_id_[7:4] register group* on page 4-55

periph_id_[3:0] register group

Figure 4-30 on page 4-53 shows the periph_id_[3:0] register group bit assignments.

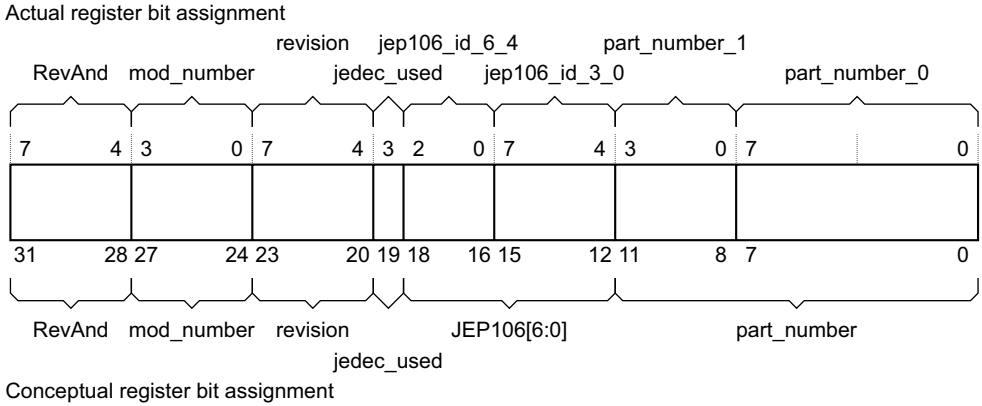


Figure 4-30 periph_id_[3:0] Register bit assignments

Table 4-18 shows the periph_id_[3:0] register group bit assignments.

Table 4-18 periph_id_[3:0] Register bit assignments

Bits	Name	Description
[31:28]	RevAnd	These bits read as 0x0.
[27:24]	mod_number	Identifies data that is relevant to the ARM partner.
[23:20]	revision	Identifies the major revision number, r*n*, of the peripheral. The revision number starts from 0 and is revision-dependent.
[19]	jedec_used	Identifies if the IC 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* on page 4-54
- *Peripheral Identification Register 1* on page 4-54
- *Peripheral Identification Register 2* on page 4-54
- *Peripheral Identification Register 4* on page 4-56.

Peripheral Identification Register 0

The periph_id_0 Register is hard-coded and the fields in the register control the reset value. Table 4-19 shows the register bit assignments.

Table 4-19 periph_id_0 Register bit assignments

Bits	Name	Description
[31:8]	-	Reserved, read undefined
[7:0]	part_number_0	These bits read back as `0x90`

Peripheral Identification Register 1

The periph_id_1 Register is hard-coded and the fields in the register control the reset value. Table 4-20 shows the register bit assignments.

Table 4-20 periph_id_1 Register bit assignments

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

Peripheral Identification Register 2

The periph_id_2 Register is hard-coded and the fields in the register control the reset value. Table 4-21 shows the register bit assignments.

Table 4-21 periph_id_2 Register bit assignments

Bits	Name	Description
[31:8]	-	Reserved, read undefined.
[7:4]	Architecture number	The architecture number of the IC. For revision r0p1, these bits read back as `0x1`.
[3]	jedec_used	This indicates that the IC uses a manufacturer's identity code that was allocated by JEDEC according to JEP106. This bit always reads back as `0x1`.
[2:0]	jep106_id_6_4	JEP106 identity code [6:4]. See the *JEP106, Standard Manufacturer's Identification Code*. These bits read back as b011 because ARM is the designer of this peripheral.

Peripheral Identification Register 3

The periph_id_3 Register is hard-coded and the fields in the register control the reset value. Table 4-22 shows the register bit assignments.

Table 4-22 periph_id_3 Register bit assignments

Bits	Name	Description
[31:8]	-	Undefined.
[7:4]	RevAnd	These bits read as 0x0.
[3:0]	mod_number	The customer can update this field if they modify the RTL of the Interrupt Controller. ARM set this to `0x0`.

periph_id_[7:4] register group

Figure 4-31 shows the periph_id_[7:4] register group bit assignments.

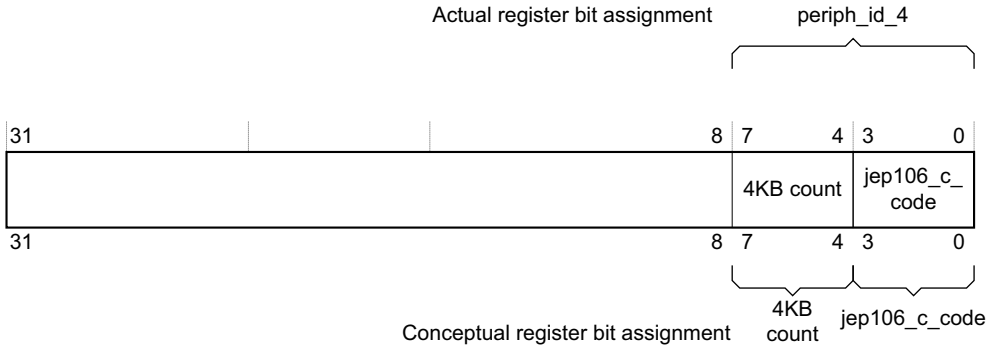


Figure 4-31 periph_id_[7:4] Register bit assignments

Table 4-23 shows the periph_id_[7:4] register group bit assignments.

Table 4-23 periph_id_[7:4] Register bit assignments

Bits	Name	Description
[31:8]	-	-
[7:4]	4KB count	Identifies the address space that the registers occupy. See Table 4-24 on page 4-56.
[3:0]	jep106_c_code	Identifies the JEP106 continuation code. See Table 4-24 on page 4-56.

The following subsection describe the periph_id_[7:4] register:

- *Peripheral Identification Register 4*.

Peripheral Identification Register 4

The periph_id_4 Register is hard-coded and the fields in the register control the reset value. Table 4-24 shows the register bit assignments.

Table 4-24 periph_id_4 Register bit assignments

Bits	Name	Description
[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 *JEP106, Standard Manufacturer's Identification Code*. These bits read back as `0x4`.

4.7.20 PrimeCell Identification Registers

The component_id_[3:0] Registers are four eight-bit wide registers, that can conceptually be treated as a single register that holds a 32-bit PrimeCell ID value. You can use the register for automatic BIOS configuration. The component_id Register is set to `0xB105F00D`. Table 4-2 on page 4-17 lists the address base offset, reset value, and access type for these registers.

Table 4-25 lists the register bit assignments.

Table 4-25 component_id Register bit assignments

component_id_0-3 register

Bits	Reset value	Register	Bits	Description
-	-	component_id_3	[31:8]	Read undefined
[31:24]	`0xB1`	component_id_3	[7:0]	These bits read back as `0xB1`
-	-	component_id_2	[31:8]	Read undefined
[23:16]	`0x05`	component_id_2	[7:0]	These bits read back as `0x05`
-	-	component_id_1	[31:8]	Read undefined

Table 4-25 component_id Register bit assignments (continued)

component_id_0-3 register

Bits	Reset value	Register	Bits	Description
[15:8]	0xF0	component_id_1	[7:0]	These bits read back as 0xF0
-	-	component_id_0	[31:8]	Read undefined
[7:0]	0x0D	component_id_0	[7:0]	These bits read back as 0x0D

Figure 4-32 shows the register bit assignments.

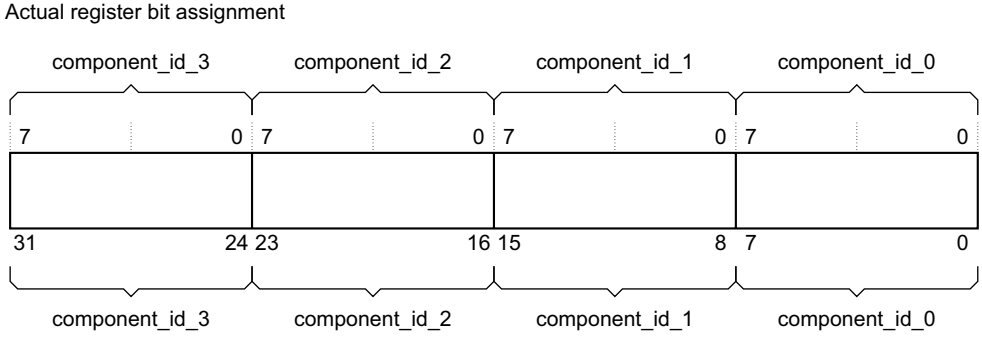


Figure 4-32 PrimeCell ID Register bit assignments

The following subsections describe the component_id Registers:

- *PrimeCell Identification Register 0* on page 4-58
- *PrimeCell Identification Register 1* on page 4-58
- *PrimeCell Identification Register 2* on page 4-58
- *PrimeCell Identification Register 3* on page 4-59.

PrimeCell Identification Register 0

The component_id_0 Register is hard-coded and the fields in the register control the reset value. Table 4-26 lists the register bit assignments.

Table 4-26 component_id_0 Register bit assignments

Bits	Name	Description
[31:8]	-	Reserved, read undefined
[7:0]	component_id_0	These bits read back as `0x0D`

PrimeCell Identification Register 1

The component_id_1 Register is hard-coded and the fields in the register control the reset value. Table 4-27 lists the register bit assignments.

Table 4-27 component_id_1 Register bit assignments

Bits	Name	Description
[31:8]	-	Reserved, read undefined
[7:0]	component_id_1	These bits read back as `0xF0`

PrimeCell Identification Register 2

The component_id_2 Register is hard-coded and the fields in the register control the reset value. Table 4-28 lists the register bit assignments.

Table 4-28 component_id_2 Register bit assignments

Bits	Name	Description
[31:8]	-	Reserved, read undefined
[7:0]	component_id_2	These bits read back as `0x5`

PrimeCell Identification Register 3

The component_id_3 Register is hard-coded and the fields in the register control the reset value. Table 4-29 lists the register bit assignments.

Table 4-29 component_id_3 Register bit assignments

Bits	Name	Description
[31:8]	-	Reserved, read undefined
[7:0]	component_id_3	These bits read back as `0xB1`

4.8 Cortex-A9 processor interface register descriptions

This section describes the registers that each Cortex-A9 processor interface provides. Table 4-30 shows the Cortex-A9 processor interface registers.

Table 4-30 Cortex-A9 processor interface register summary

Base	Name	Type	Reset	Width	Description
0x000	ICPICR	RW	0x00000000	32	*Processor Interface Control Register*
0x004	ICCIPMR	RW	0x00000000	32	*Priority Mask Register* on page 4-63
0x008	ICCBPR	RW	0x2 0x3	32	*Binary Point Register* on page 4-64
0x00C	ICCIAR	RO	0x000003FF	32	*Interrupt Acknowledge Register* on page 4-68
0x010	ICCEOIR	WO	-	32	*End Of Interrupt Register* on page 4-69
0x014	ICCRPR	RO	0x000000FF	32	*Running Priority Register* on page 4-71
0x018	ICCHPIR	RO	0x000003FF	32	*Highest Pending Interrupt Register* on page 4-72
0x01C[a]	ICCABPR	RW	0x3	32	*Aliased Non-secure Binary Point Register* on page 4-73
0xFC	ICPIIR	RO	0x3901043B	32	*Processor Interface Implementer Identification Register* on page 4-74

a. This address location is only accessible when the Cortex-A9 processor performs a Secure access.

4.8.1 Processor Interface Control Register

The ICPICR Register characteristics are:

Purpose Controls the operating state of a Cortex-A9 processor interface.

Usage constraints This register is banked. The register you access depends on the type of access:

Non-secure access

Cortex-A9 processor interface provides access to the control_ns Register.

Secure access

Cortex-A9 processor interface provides access to the ICPICR Register.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-30 on page 4-60.

Figure 4-33 shows the ICPICR Register bit assignments.

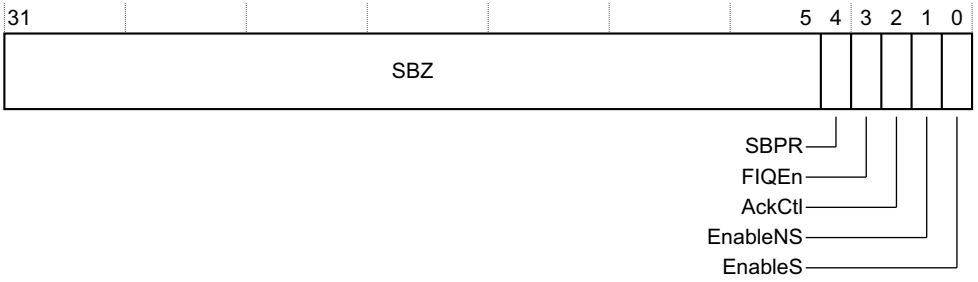


Figure 4-33 ICPICR Register bit assignments

Figure 4-34 shows the control_ns Register bit assignments.

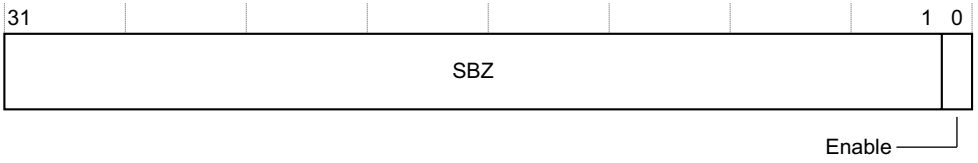


Figure 4-34 control_ns Register bit assignments

Table 4-31 shows the ICPICR Register bit assignments when you perform a secure access.

Table 4-31 ICPICR Register bit assignments

Bits	Name	Description
[31:5]	-	SBZ.
[4]	SBPR	Controls which Binary Pointer Register the Cortex-A9 processor interface uses when it performs a preemptive calculation. The options are: 0 = secure interrupts use the bin_pt_s Register and non-secure interrupts use the bin_pt_ns Register 1 = secure read and writes access the secure binary point register directly. Non-secure writes are ignored, and non-secure reads return the value in the secure binary point register plus 1, with the addition saturating at a value of 7. See *Priority formats* on page 4-5 and *Binary Point Register* on page 4-64.
[3]	FIQEn	Enables the Cortex-A9 processor interface to send secure interrupts using the **nFIQ** signal. 0 = the Cortex-A9 processor interface uses **nIRQ** when signaling secure or non-secure interrupts. 1 = the Cortex-A9 processor interface uses **nFIQ** when signaling secure interrupts, and **nIRQ** when signaling non-secure interrupts.
[2]	AckCtl[a]	When a Cortex-A9 processor performs a secure read of the ICCIAR Register and the highest priority interrupt is non-secure, this bit controls the acknowledge response as follows: 0 = The Cortex-A9 processor interface returns a value of 1022 and the interrupt remains pending. 1 = The Cortex-A9 processor interface returns the ID of the non-secure interrupt and acknowledges the interrupt. The interrupt changes state to active, or active and pending.
		When a Cortex-A9 processor performs a secure write to the EOI Register to signal the completion of a non-secure interrupt, this bit controls if the Interrupt Controller clears the interrupt as follows: 0 = the Interrupt Controller ignores the write and the interrupt remains active, or active and pending 1 = the Interrupt Controller changes the interrupt status to inactive, or pending.
[1]	EnableNS[b]	Non-secure enable for the Cortex-A9 processor interface: 0 = disables the Cortex-A9 processor interface from sending non-secure interrupts to the Cortex-A9 processor 1 = enables the Cortex-A9 processor interface to send non-secure interrupts to the Cortex-A9 processor.
[0]	EnableS	Secure enable for the Cortex-A9 processor interface: 0 = disables the Cortex-A9 processor interface from sending secure interrupts to the Cortex-A9 processor 1 = enables the Cortex-A9 processor interface to send secure interrupts to the Cortex-A9 processor.

a. This bit modifies the behavior of some secure accesses to the ICCIAR Register and the EOI Register.
b. This is an alias of the Enable bit in the control_ns Register.

Note

If you disable a Cortex-A9 processor interface when it contains pending interrupts then you might get spurious interrupts after you re-enable the Cortex-A9 processor interface.

Table 4-32 shows the control_ns Register bit assignments when you perform a non-secure access.

Table 4-32 control_ns Register bit assignments

Bits	Name	Description
[31:1]	-	SBZ.
[0]	Enable	Enable for the Cortex-A9 processor interface: 0 = disables the Cortex-A9 processor interface from sending Non-secure interrupts to the Cortex-A9 processor 1 = enables the Cortex-A9 processor interface to send Non-secure interrupts to the Cortex-A9 processor.

Legacy interrupt support

You can configure the ICPICR Register so that the Interrupt Controller supports the **nIRQ** and **nFIQ** signals of a Cortex-A9 processor, with interrupts passing through the Cortex-A9 processor interface with no interrupt processing. Table 4-1 on page 4-12 shows the pass-through behavior.

4.8.2 Priority Mask Register

The ICCIPMR Register characteristics are:

Purpose Enables the Cortex-A9 processor to set a priority level mask.

Usage constraints

- The Cortex-A9 processor interface cannot send interrupts to the Cortex-A9 processor that have a priority level that is lower than, or equal to, the priority level that the mask contains.
- For Non-secure writes the Cortex-A9 processor interface ignores the write, if prior to this command the mask contained a value in the upper half of the priority level range.

- Non-secure reads the Cortex-A9 processor interface returns `0x00`, if prior to this command the mask contained a value in the upper half of the priority level range.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-30 on page 4-60.

Figure 4-35 shows the ICCIPMR Register bit assignments.

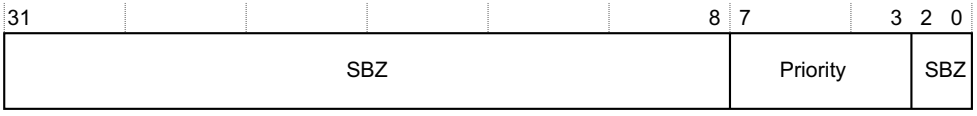


Figure 4-35 ICCIPMR Register bit assignments

Table 4-33 shows the ICCIPMR Register bit assignments.

Table 4-33 ICCIPMR Register bit assignments

Bits	Name	Description
[31:8]	-	SBZ.
[7:3]	Priority	Configures the priority mask level for the Cortex-A9 processor. The Cortex-A9 processor interface does not send interrupts to the Cortex-A9 processor if the priority level of the interrupt is lower than, or equal to, the priority level that this field contains. For example: **Priority mask =** `0x20` Interrupts with a priority level lower than or equal to `0x20` are not sent to the Cortex-A9 processor.
[2:0]	-	SBZ.

4.8.3 Binary Point Register

The ICCIBPR Register characteristics are:

Purpose Modifies the preemption behavior of the Cortex-A9 processor interface. It enables the Cortex-A9 processor to either:

- prevent the Cortex-A9 processor interface from preempting interrupts on the Cortex-A9 processor
- only enable the Cortex-A9 processor interface to issue interrupts that have a higher priority, after the binary point mask value has been applied.

Usage constraints This register is banked. The register you access depends on the type of access:

- **Non-secure access** Cortex-A9 processor interface provides access to the ICCIBPR_ns Register.
- **Secure access** Cortex-A9 processor interface provides access to the ICCIBPR_s Register.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-30 on page 4-60.

Figure 4-36 shows the register bit assignments for both binary point registers.

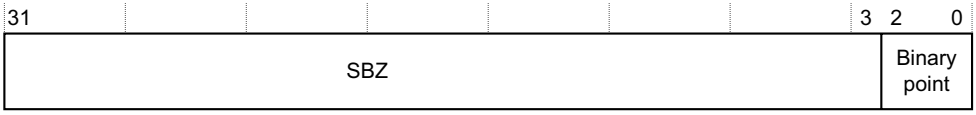


Figure 4-36 ICCIBPR_s Register and ICCIBPR_ns Register bit assignments

Table 4-34 shows the ICCIBPR Register bit assignments.

Table 4-34 ICCIBPR_s Register and ICCIBPR_ns Register bit assignments

Bits	Name	Description
[31:3]	-	SBZ.
[2:0]	Binary point	Configures the value of the binary point mask. The mask value also depends on the security level of the interrupt as follows: **Secure interrupts** b000 = mask is `0xF8` b001 = mask is `0xF8` b010 = mask is `0xF8` b011 = mask is `0xF0` b100 = mask is `0xE0` b101 = mask is `0xC0` b110 = mask is `0x80` b111 = mask is `0x00`. This prevents the Cortex-A9 processor interface from preempting interrupts. The reset value for the secure binary point mask is `0x2`. **Non-secure interrupts** b000 = mask is `0xF0` b001 = mask is `0xF0` b010 = mask is `0xF0` b011 = mask is `0xF0` b100 = mask is `0xF0` b101 = mask is `0xE0` b110 = mask is `0xC0` b111 = mask is `0x80`. This prevents the Cortex-A9 processor interface from preempting interrupts that have a priority level that occupy the lower half of the priority level range. Normally non-secure interrupts occupy the lower half of the priority level range. The reset value for the secure binary point mask is `0x3`. See *Priority formats* on page 4-5.

Note

The Cortex-A9 processor interface provides the ICCABPR Register, an alias of the bin_pt_ns Register. This enables a Cortex-A9 processor operating permanently in the secure domain, to access the bin_pt_ns Register. See *Aliased Non-secure Binary Point Register* on page 4-73.

When the Cortex-A9 processor interface receives an interrupt with a higher priority than the ICCRPR Register contains, then it performs the following calculations:

1. Calculate the masked priority of the interrupt:

   IntMask = (Priority-level of interrupt) AND (binary point mask)

2. Calculate the masked priority of the running interrupt:

   RunIntMask = (Priority-level of interrupt in the ICCRPR Register) AND (binary point mask)

If IntMask < RunIntMask then the interrupt satisfies the priority-level requirement and the Cortex-A9 processor interface signals the interrupt to the Cortex-A9 processor.

Note

After this calculation, if two or more interrupts satisfy the priority-level requirement then the Cortex-A9 processor interface signals the interrupt that has the lowest priority number.

If several interrupts share the lowest priority number then the Cortex-A9 processor interface issues the interrupt that has the lowest interrupt.

The reset value for the secure register is `0x2` and the value for the non-secure register is `0x3`.

4.8.4 Interrupt Acknowledge Register

The ICCIAR Register characteristics are:

Purpose Enables the Cortex-A9 processor to determine the identifier of the current pending interrupt.

Usage constraints
- When the ACK_INTID field contains an ID from zero to 255 then after the Cortex-A9 processor reads this register the Interrupt Controller changes the interrupt status to either active, or active and pending.
- When you program an SPI to target more than one Cortex-A9 processor, hardware guarantees that only one CPU receives this interrupt.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-30 on page 4-60.

Figure 4-37 shows the ICCIAR Register bit assignments.

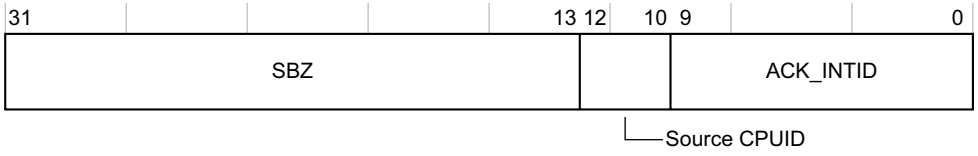


Figure 4-37 int_ack Register bit assignments

Table 4-35 shows the ICCIAR Register bit assignments.

Table 4-35 int_ack Register bit assignments

Bits	Name	Description
[31:13]	-	SBZ.
[12:10]	Source Processor ID	Returns the Processor ID of the Cortex-A9 processor that requested the software interrupt[a]: b000 = CPU0 generated the SGI that the ACK_INTID field contains b001 = CPU1 generated the SGI that the ACK_INTID field contains b010 = CPU2 generated the SGI that the ACK_INTID field contains b011 = CPU3 generated the SGI that the ACK_INTID field contains
[9:0]	ACK_INTID	Returns the ID of the interrupt that requires servicing by the Cortex-A9 processor: 15 - 0 = SGI [15:0] 27 - 16 = Reserved 31 - 28 = PPI [3:0] 255 - 32 = SPI [223:0] 256 - 1021 - = reserved 1022 = the highest priority interrupt that requires servicing is non-secure[b] 1023 = no outstanding interrupts.

a. For PPIs, SPIs, and single-processor implementations this field is always b000.
b. This response only occurs when the Cortex-A9 processor performs a secure read and the AckCtl bit is 0 in the ICPICR Register. See Table 4-31 on page 4-62.

4.8.5 End Of Interrupt Register

The ICCEOIR Register characteristics are:

Purpose Enables the Cortex-A9 processor to signal to the Cortex-A9 processor interface when it completes the interrupt service routine.

Usage constraints When the Cortex-A9 processor writes an INTID of 0-255 in the ICCEOIR_INTID field then the Interrupt Controller changes the interrupt status to either inactive, or pending, under one of the following conditions:

- The Cortex-A9 processor performs a secure write to the register, to signal the completion of a secure interrupt.
- The Cortex-A9 processor performs a non-secure write to the register, to signal the completion of a non-secure interrupt.

- The Cortex-A9 processor performs a secure write to the register, to signal the completion of a non-secure interrupt and the AckCtl bit is 1 in the ICPICR Register. See Table 4-31 on page 4-62.

Note

The Cortex-A9 processor interface ignores writes to the register under the following conditions:

- The Cortex-A9 processor provides a combination of CPUID and INTID that does not correspond to an interrupt that is active, or active and pending, in the distributor.
- The Cortex-A9 processor performs a secure write to the register, to signal the completion of a non-secure interrupt and the AckCtl bit is 0 in the ICPICR Register. See Table 4-31 on page 4-62.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-30 on page 4-60.

Figure 4-38 shows the ICCEOIR Register bit assignments.

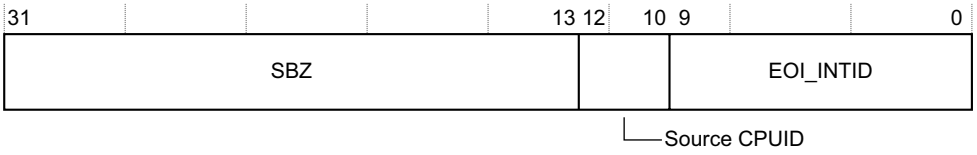


Figure 4-38 ICCEOIR Register bit assignments

Table 4-36 shows the ICCEOIR Register bit assignments.

Table 4-36 ICCEOIR Register bit assignments

Bits	Name	Description
[31:13]	-	SBZ.
[12:10]	Source CPUID	After the Cortex-A9 processor completes the interrupt service routine for an SGI, it sets this to the source CPUID of the SGI that it serviced[a]: b000 = CPU0 generated the SGI that the ICCEOIR_INTID field contains b001 = CPU1 generated the SGI that the ICCEOIR_INTID field contains b010 = CPU2 generated the SGI that the ICCEOIR_INTID field contains b011 = CPU3 generated the SGI that the ICCEOIR_INTID field contains.
[9:0]	EOI_INTID	After the Cortex-A9 processor completes its interrupt service routine, it sets this field to the ID of the interrupt that it serviced: 15-0 = SGI [15:0] 27-16 = Reserved 31-28 = PPI [3:0] 255-32 = SPI [223:0] 256 - 1023= reserved[b].

a. For PPIs, SPIs, and single-processor implementations the CPU must set this field to b000.
b. There is no requirement for the Cortex-A9 processor to signal a response for these management interrupts because the Interrupt Controller changes the interrupt status to inactive, after the Cortex-A9 processor reads the ICCIAR Register.

4.8.6 Running Priority Register

The ICCRPR Register characteristics are:

Purpose Provides the Cortex-A9 processor with the interrupt priority level of the highest priority interrupt that is active on that Cortex-A9 processor.

Usage constraints
- If there are no interrupts running on the Cortex-A9 processor then when it reads the register the Cortex-A9 processor interface returns `0xFF`, corresponding to the lowest priority level.
- When the Cortex-A9 processor performs a non-secure read of the register and the highest priority interrupt that is running on the Cortex-A9 processor is secure then the Cortex-A9 processor interface returns `0x00`, corresponding to the highest priority level.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-30 on page 4-60.

Figure 4-39 shows the ICCRPR Register bit assignments.

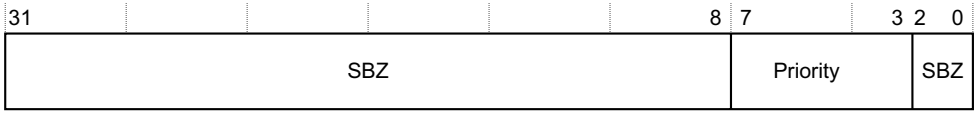


Figure 4-39 ICCRPR Register bit assignments

Table 4-37 shows the ICCRPR Register bit assignments.

Table 4-37 ICCRPR Register bit assignments

Bits	Name	Description
[31:8]	-	SBZ.
[7:3]	Priority	Returns the priority level of the highest priority interrupt that is running on the Cortex-A9 processor. **Note** The priority of an interrupt can have a range from `0x00`, the highest level, to `0x1F`, the lowest level.
[2:0]	-	SBZ.

4.8.7 Highest Pending Interrupt Register

The ICCHPIR Register characteristics are:

Purpose Provides the Cortex-A9 processor with the interrupt ID of the highest priority interrupt that is pending for that Cortex-A9 processor.

Usage constraints There are no usage constraints.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-30 on page 4-60.

Figure 4-40 on page 4-73 shows the ICCHPIR Register bit assignments.

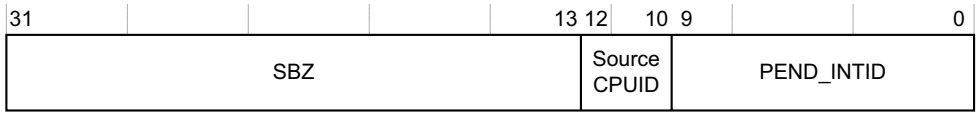


Figure 4-40 ICCHPIR Register bit assignments

Table 4-38 shows the ICCHPIR Register bit assignments.

Table 4-38 ICCHPIR Register bit assignments

Bits	Name	Description
[31:13]	-	SBZ.
[12:10]	Source Processor ID	Returns the Processor ID of the Cortex-A9 processor that is requesting the software interrupt[a]: b000 = CPU0 generated the SGI that the PEND_INTID field contains b001 = CPU1 generated the SGI that the PEND_INTID field contains b010 = CPU2 generated the SGI that the PEND_INTID field contains b011 = CPU3 generated the SGI that the PEND_INTID field contains.
[9:0]	PEND_INTID	Returns the ID of the highest priority pending interrupt: 15-0 = SGI [15:0] 31-16 = PPI [15:0] 255-32 = SPI [223:0} 256 - 1021 = reserved 1022 = the interrupt that requires servicing is Non-secure[b] 1023 = no outstanding interrupts. **Note** The Cortex-A9 processor interface ignores interrupts that are in the active and pending state.

a. For PPIs, SPIs, and single-processor implementations this field is always b000.
b. This response can only occur when the Cortex-A9 processor performs a secure read and the AckCtl bit is 0 in the ICPICR Register.

4.8.8 Aliased Non-secure Binary Point Register

The ICCABPR Register characteristics are:

Purpose Enables a Cortex-A9 processor operating in the secure domain to set the position of the binary point in the bin_pt_ns Register.

Usage constraints The ICCABPR Register is an alias of the bin_pt_ns Register. See see Figure 4-36 on page 4-65 and Table 4-34 on page 4-66. You can only access the ICCABPR Register when the Cortex-A9 processor is in Secure state.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-30 on page 4-60.

4.8.9 Processor Interface Implementer Identification Register

The ICPIIR Register characteristics are:

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

Usage constraints There are no usage constraints.

Configurations Available in all Cortex-A9 multiprocessor configurations.

Attributes See the register summary in Table 4-30 on page 4-60.

Figure 4-41 shows the ICPIIR Register bit assignments.

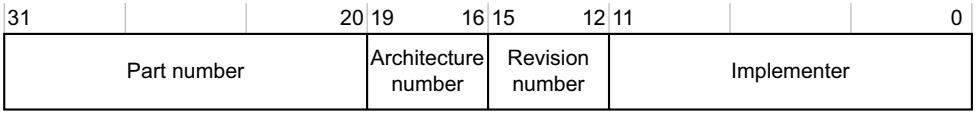


Figure 4-41 ICPIIR Register bit assignments

Table 4-39 shows the ICPIIR Register bit assignments.

Table 4-39 ICPIIR Register bit assignments

Bits	Name	Description
[31:20]	Part number	Identifies the peripheral. The part number of the Interrupt Controller is `0x390`.

Table 4-39 ICPIIR Register bit assignments (continued)

Bits	Name	Description
[19:16]	Architecture version	Identifies the architecture version. The version of the Interrupt Controller is `0x1`.
[15:12]	Revision number	Returns the revision number of the Interrupt Controller. The implementer defines the format of this field. The revision number of the Interrupt Controller is `0x0`.
[11:0]	Implementer	Returns the JEP106 code of the company that implemented the Cortex-A9 processor interface RTL. It uses the following construct: **[11:8]** the JEP106 continuation code of the implementer **[7]** 0 **[6:0]** the JEP106 code [6:0] of the implementer. The implementer number is `0x43B`.

Chapter 5
Timer and Watchdog Registers

This chapter describes the timer and watchdog registers. It contains the following sections:

5.1 About the timer and watchdog blocks

Both timer and watchdog blocks have the following features:

- a 32-bit counter that generates an interrupt when it reaches zero
- an 8-bit prescaler to enable better control of the period
- configurable single-shot or auto-reload modes
- configurable starting values for the counter
- the clock for these blocks is the same clock as the Interrupt Controller clock.

The watchdog can be configured as a timer. See *Clocking* on page 7-2 for a description of **CLK**, **PERIPHCLK**, and **PERIPHCLKEN**.

5.1.1 Calculating timer intervals

The timer interval is calculated using the following equation:

$$\left(\frac{(\text{PRESCALER\_value}+1) \times (\text{Load\_value}+1) \times N}{\text{SCU CLK\_frequency}}\right)$$

This equation can be used to calculate the period between two events out of the timers and the watchdog time-out time. N is the ratio between **CLK** and **PERIPHCLK**.

5.1.2 Security extensions

See *SCU Secure Access Control Register (SSAC)* on page 3-12 for information about using timers in Secure or Non-secure state.

5.2 Timer and watchdog registers

Addresses are relative to the base address of the timer and watchdog region defined by the private memory map (see Chapter 2 *Cortex-A9 MPCore Private Memory Region*). All timer and watchdog registers are word-accessible only.

Use **nPERIPHRESET** to reset these registers, except the Watchdog Reset Status Register.

nWDRESET resets the Watchdog Reset Status Register. See *Resets* on page A-3.

Table 5-1 shows the timer and watchdog registers. All registers not described in Table 5-1 are Reserved.

Table 5-1 Timer and watchdog registers

Offset	Type	Reset Value	Description
0x00	RW	0x00000000	*Timer Load Register*
0x04	RW	0x00000000	*Timer Counter Register*
0x08	RW	0x00000000	*Timer Control Register* on page 5-4
0x0C	RW	0x00000000	*Timer Interrupt Status Register* on page 5-5
0x20	RW	0x00000000	*Watchdog Load Register* on page 5-5
0x24	RW	0x00000000	*Watchdog Counter Register* on page 5-6
0x28	RW	0x00000000	*Watchdog Control Register* on page 5-6
0x2C	RW	0x00000000	*Watchdog Interrupt Status Register* on page 5-7
0x30	RW	0x00000000	*Watchdog Reset Status Register* on page 5-9
0x34	WO	-	*Watchdog Disable Register* on page 5-9

5.2.1 Timer Load Register

The Timer Load Register contains the value copied to the Timer Counter Register when it decrements down to zero with auto reload mode enabled. Writing to the Timer Load Register means that you also write to the Timer Counter Register.

5.2.2 Timer Counter Register

The Timer Counter Register is a decrementing counter.

The Timer Counter Register decrements if the timer is enabled using the timer enable bit in the Timer Control Register. If the Cortex-A9 processor belonging to the timer is in debug state, the counter does not decrement until the Cortex-A9 processor returns to non debug state.

When the Timer Counter Register reaches zero and auto reload mode is enabled, it reloads the value in the Timer Load Register and then decrements from that value. If auto reload mode is not enabled the Timer Counter Register decrements down to zero and stops.

When the Timer Counter Register reaches zero, the timer interrupt status event flag is set and the interrupt ID 29 is set as pending in the Interrupt Distributor, if interrupt generation is enabled in the Timer Control Register.

Writing to the Timer Counter Register or Timer Load Register forces the Timer Counter Register to decrement from the newly written value.

5.2.3 Timer Control Register

Figure 5-1 shows the Timer Control Register format.

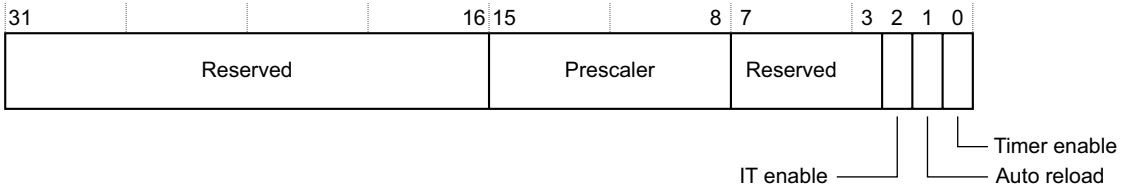


Figure 5-1 Timer Control Register format

Table 5-2 shows the Timer Control Register bit assignments.

Table 5-2 Timer Control Register bit assignments

Bits	Name	Description
[31:16]	-	SBZ
[15:8]	Prescaler	The prescaler modifies the clock period for the decrementing event for the Counter Register. See *Calculating timer intervals* on page 5-2 for the equation.
[7:3]	-	SBZ

Table 5-2 Timer Control Register bit assignments (continued)

Bits	Name	Description
[2]	IT Enable	If set, the interrupt ID 29 is set as pending in the Interrupt Distributor when the event flag is set in the Timer Status Register.
[1]	Auto-reload	1'b0 = Single shot mode. Counter decrements down to zero, sets the event flag and stops. 1'b1 = Auto-reload mode. Each time the Counter Register reaches zero, it is reloaded with the value contained in the Load Register and then continues decrementing.
[0]	Timer Enable	Global timer enable 1'b0 = Timer is disabled and the counter does not decrement. All registers can still be read or/and written 1'b1 = Timer is enabled and the counter decrements normally.

5.2.4 Timer Interrupt Status Register

Figure 5-2 shows the Timer Interrupt Status Register format.

The event flag is a sticky bit that is automatically set when the Counter Register reaches zero. If the timer interrupt is enabled, Interrupt ID 29 is set as pending in the Interrupt Distributor after the event flag is set.The event flag is cleared when written to 1. Trying to write a zero to the event flag or a one when it is not set has no effect.

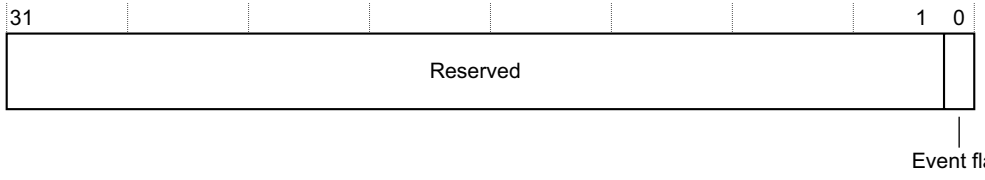


Figure 5-2 Timer Interrupt Status Register format

5.2.5 Watchdog Load Register

The Watchdog Load Register contains the value copied to the Watchdog Counter Register when it decrements down to zero with auto reload mode enabled, in Timer mode. Writing to the Watchdog Load Register means that you also write to the Watchdog Counter Register.

5.2.6 Watchdog Counter Register

The Watchdog Counter Register is a down counter.

It decrements if the Watchdog is enabled using the Watchdog enable bit in the Watchdog Control Register. If the Cortex-A9 processor belonging to the Watchdog is in debug state, the counter does not decrement until the Cortex-A9 processor returns to non debug state.

When the Watchdog Counter Register reaches zero and auto reload mode is enabled, and in timer mode, it reloads the value in the Watchdog Load Register and then decrements from that value. If auto reload mode is not enabled or the watchdog is not in timer mode, the Watchdog Counter Register decrements down to zero and stops.

When in watchdog mode the only way to update the Watchdog Counter Register is to write to the Watchdog Load Register. When in timer mode the Watchdog Counter Register is write accessible.

The behavior of the watchdog when the Watchdog Counter Register reaches zero depends on its current mode:

Timer mode When the Watchdog Counter Register reaches zero, the watchdog interrupt status event flag is set and the interrupt ID 30 is set as pending in the Interrupt Distributor, if interrupt generation is enabled in the Watchdog Control Register.

Watchdog mode

If a software failure prevents the Watchdog Counter Register from being refreshed, the Watchdog Counter Register reaches zero, the Watchdog reset status flag is set and the associated **WDRESETREQ** reset request output pin is asserted. The external reset source is then responsible for resetting all or part of the Cortex-A9 MPCore design.

5.2.7 Watchdog Control Register

Figure 5-3 shows the Watchdog Control Register format.

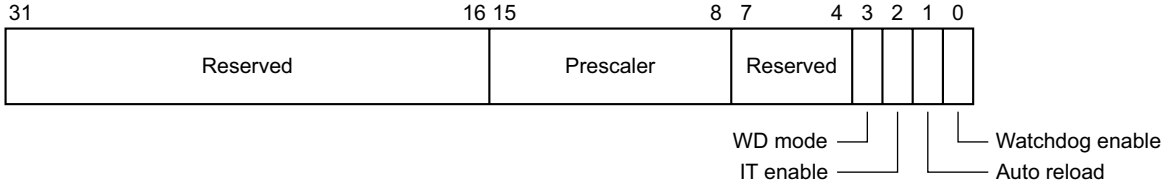


Figure 5-3 Watchdog Control Register format

Table 5-3 shows the Watchdog Control Register bit assignments.

Table 5-3 Watchdog Control Register bit assignments

Bits	Name	Description
[31:16]	-	Read as zero. Do not modify.
[15:8]	Prescaler	The prescaler modifies the clock period for the decrementing event for the Counter Register. See *Calculating timer intervals* on page 5-2.
[7:4]	-	Read as zero. Do not modify.
[3]	Watchdog mode	1'b0 = Timer mode, default Writing a zero to this bit has no effect. You must use the Watchdog Disable Register to put the watchdog into timer mode. See *Watchdog Disable Register* on page 5-9. 1'b1 = Watchdog mode.
[2]	IT Enable	If set, the interrupt ID 30 is set as pending in the Interrupt Distributor when the event flag is set in the watchdog Status Register. In watchdog mode this bit is ignored.
[1]	Auto-reload	1'b0 = Single shot mode. Counter decrements down to zero, sets the event flag and stops. 1'b1 = Auto-reload mode. Each time the Counter Register reaches zero, it is reloaded with the value contained in the Load Register and then continues decrementing. In watchdog mode this bit is ignored.
[0]	Watchdog Enable	Global watchdog enable 1'b0 = Watchdog is disabled and the counter does not decrement. All registers can still be read and /or written 1'b1 = Watchdog is enabled and the counter decrements normally.

5.2.8 Watchdog Interrupt Status Register

Figure 5-4 shows the Watchdog Interrupt Status Register format.

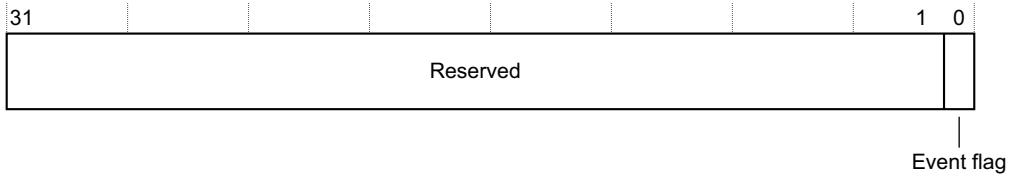


Figure 5-4 Watchdog Interrupt Status Register format

The event flag is a sticky bit that is automatically set when the Counter Register reaches zero in timer mode. If the watchdog interrupt is enabled, Interrupt ID 30 is set as pending in the Interrupt Distributor after the event flag is set. The event flag is cleared when written with a value of 1. Trying to write a zero to the event flag or a one when it is not set has no effect.

5.2.9 Watchdog Reset Status Register

Figure 5-5 shows the Watchdog Reset Status Register format.

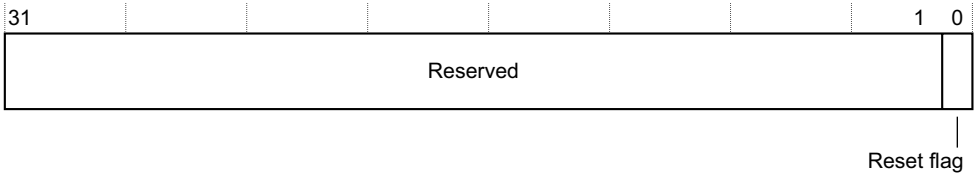


Figure 5-5 Watchdog Reset Status Register format

The reset flag is a sticky bit that is automatically set when the Counter Register reaches zero and a reset request is sent accordingly. (In watchdog mode)

The reset flag is cleared when written with a value of 1. Trying to write a zero to the reset flag or a one when it is not set has no effect. This flag is not reset by normal Cortex-A9 processor resets but has its own reset line that must not be asserted when the Cortex-A9 processor reset assertion is the result of a watchdog reset request with **WDRESETREQ**. This distinction enables software to differentiate between a normal boot sequence, reset flag is zero, and one caused by a previous watchdog time-out, reset flag set to one.

5.2.10 Watchdog Disable Register

Use the Watchdog Disable Register to switch from watchdog to timer mode. The software must write `0x12345678` then `0x87654321` successively to the Watchdog Disable Register so that the watchdog mode bit in the Watchdog Control Register is set to zero.

If one of the values written to the Watchdog Disable Register is incorrect or if any other write occurs in between the two word writes, the watchdog remains in its current state. To reactivate the Watchdog, the software must write 1 to the watchdog mode bit of the Watchdog Control Register. See *Watchdog Control Register* on page 5-6.

Chapter 6
Level 2 Memory Interface

This chapter describes the L2 memory interface. It contains the following sections:

6.1 Cortex-A9 MPCore L2 interface

This section describes the Cortex-A9 MPCore Level 2 interface in:

- *About the Cortex-A9 MPCore L2 interface*
- *Cortex-A9 processor supported AXI transactions* on page 6-3
- *AXI transaction IDs* on page 6-3
- *Accelerator Coherency Port* on page 6-7
- *Exclusive L2 cache* on page 6-10.

6.1.1 About the Cortex-A9 MPCore L2 interface

The Cortex-A9 MPCore L2 interface can have two 64-bit wide AXI bus masters. In a two bus master configuration there is also an option to configure address filtering. See *Address filtering* on page 3-2 for more information.

Table 6-1 shows the AXI master interface attributes.

Table 6-1 AXI master interface attributes

Attribute	Format
Write Issuing Capability	10 per processor, including: • eight non-cacheable writes • two evictions. If the ACP is implemented, eight more write transactions can be issued.
Read Issuing Capability	9 per processor, including: • four instruction reads • four linefill reads. • one non-cacheable read. If the ACP is implemented, eight more read transactions can be issued.
Combined Issuing Capability	21 per processor. If the ACP is implemented, 16 more transactions can be issued.
Write ID Capability	32
Write Interleave Capability	1
Write ID Width	6
Read ID Capability	32
Read ID Width	6

The AXI protocol and meaning of each AXI signal are not described in this document. For more information see *AMBA AXI Protocol v1.0 Specification*.

6.1.2 Cortex-A9 processor supported AXI transactions

Cortex-A9 master ports generate only a subset of all possible AXI transactions.

For coherent and noncoherent write-back write-allocate transactions the supported transactions are:

- WRAP4 64-bit for read transactions (linefills)
- INCR4 64-bit for write transactions (evictions)

For noncoherent non-cacheable transactions:

- INCRN (N:1-16) 32-bit read transfers
- INCRN (N:1-8) 64-bit read transfers
- INCR1 8-bit, 16-bit, 32-bit, and 64-bit read transfers
- INCR1 8-bit, 16-bit, 32-bit, and 64-bit write transfers
- INCR1 8-bit, 16-bit, 32-bit, 64-bit exclusive read transfers
- INCR1 8-bit, 16-bit, 32-bit, 64-bit exclusive write transfers
- INCR1 32-bit RW (locked) for swap
- INCR1 8-bit RW (locked) for swap.

The following points apply to AXI transfers:

- Wrap burst are only read transfers, 64-bit, 4 beats
- Incr single can be any size for read or write
- Incr burst (more than one beat) are only 32-bit or 64-bit
- No transfer is marked as FIXED
- Write transfers with all byte strobes low can occur.

6.1.3 AXI transaction IDs

There are several possible sources for the AXI transactions a Cortex-A9MP processor issues on its AXI master ports. This section describes the AXI transaction IDs and AXI USER bits in the following sections:

- *ARIDMx[5:0] encodings* on page 6-4
- *AWIDMx[5:0] encodings* on page 6-4.
- *ARUSERMx[6:0] encodings* on page 6-5
- *AWUSERMx[8:0] encodings* on page 6-6.

ARIDMx[5:0] encodings

This section describes the ARIDMx[5:0] encodings for read transactions. As Table 6-2 shows, the **ARIDMx[2]** encodings distinguish between transactions originating from Cortex-A9 processors and transactions originating from the ACP:

- **ARIDMx[2]** = 0 the transaction originates from one of the Cortex-A9 processors.
- **ARIDMx[2]** = 1 the transaction originates from the ACP.

Table 6-2 ARID encodings

	Transaction types	
	Cortex-A9 transactions	**ACP transactions**
ARIDMx[2]	**ARIDMx[2]** = 0	**ARIDMx[2]** = 1
ARIDMx[5:3]	Transaction type: b000 non-cacheable b010 data linefill buffer 0 b011 data linefill buffer 0 b100 instruction linefill b101 instruction linefill b110 instruction linefill b111 instruction linefill	ACP read IDs **ARIDMx[5:3] = ARIDS[2:0]**
ARIDMx[1:0]	Cortex-A9 processor: b00 CPU0 b01 CPU1 b10 CPU2 b11 CPU3	unused

AWIDMx[5:0] encodings

This section describes the **AWIDMx[5:0**] encodings for write transactions. As Table 6-3 on page 6-5 shows, the **AWIDMx[2]** encodings distinguish between transactions originating from Cortex-A9 processors and transactions originating from the ACP:

- **AWIDMx[2]** = 0 the transaction originates from one of the Cortex-A9 processors.

- **AWIDMx[2]** = 1 the transaction originates from the ACP.

Table 6-3 AWIDMx encodings

	Transaction types	
	Cortex-A9 transactions	**ACP transactions**
AWIDMx[2]	**AWIDMx[2]** = 0	**AWIDMx[2]** = 1
AWIDMx[5:3]	b000 non-cacheable b010 eviction b011 eviction b100 eviction b101 eviction	ACP read IDs **AWIDMx[5:3] = AWIDS[2:0]**
AWIDMx[1:0]	b00 CPU0 b01 CPU1 b10 CPU2 b11 CPU3	unused

6.1.4 AXI USER encodings

This section describes the implementation-specific AXI USER bit encodings on the master ports in the following sections:

- *ARUSERMx[6:0] encodings*
- *AWUSERMx[8:0] encodings* on page 6-6.

ARUSERMx[6:0] encodings

This section describes the **ARUSERMx[6:0]** encodings for read transactions. As Table 6-4 on page 6-6 shows, the value and the meaning of the **ARUSERMx** encodings depend on the source of the transaction. originating from Cortex-A9 processors and transactions originating from the ACP:

- **ARIDMx[2]** = 0 the transaction originates from one of the Cortex-A9 processors.

- **ARIDMx[2]** = 1 the transaction originates from the ACP.

Table 6-4 ARUSERMx[6:0] encodings

	Transaction types	
	Cortex-A9 transactions **ARIDMx[2] = 0**	**ACP transactions** **ARIDMx[2] = 1**
ARUSERMx[6]	b0 Reserved	ACP USER bits ARUSERMx[6:1] = **ARUSERS[6:1]**[a]
ARUSERMx[5]	Prefetch hint	
ARUSERMx[4:1]	Inner attributes b0000 Strongly Ordered 0001 Device b0011 Normal Memory NonCacheable b0110 WriteThrough b0111 Write Back no Write Allocate b1111 Write Back Write Allocate	
ARUSERMx[0]	Shared bit 1 Coherent request 0 Non-coherent request	

a. Each master agent connected to the ACP can determine its corresponding AXI USER signals. To maintain consistency ARM recommends that the ACP AXI USER signal encodings match those of the Cortex-A9 processors.

AWUSERMx[8:0] encodings

This section describes the **AWUSERMx[8:0]** encodings for read transactions. As Table 6-5 on page 6-7 shows, the value and the meaning of the **AWUSERMx** encodings depend on the source of the transaction:

- **ARIDMx[2]** = 0 the transaction originates from one of the Cortex-A9 processors.

- **ARIDMx[2]** = 1 the transaction originates from the ACP.

Table 6-5 AWUSERMx[8:0] encodings

	Transaction types	
	Cortex-A9 transactions **ARIDMx[2] = 0**	**ACP transactions** **ARIDMx[2] = 1**
AWUSERMx[8]	b0 Reserved	
AWUSERMx[7]	b0 Reserved	
AWUSERMx[6]	Clean eviction information	ACP USER bits AWUSERMx[8:0] = **AWUSERS[4:0]**[a]
AWUSERMx[5]	L1 eviction information	
AWUSERMx[4:1]	Inner attributes b0000 Strongly Ordered b0001 Device b0011 Normal Memory NonCacheable b0110 WriteThrough b0111 Write Back no Write Allocate b1111 Write Back Write Allocate	
AWUSERMx[0]	Shared bit b0 Non-coherent request b1 Coherent request	

a. Each master agent connected to the ACP can determine its corresponding AXI USER signals. To maintain consistency ARM recommends that the ACP AXI USER signal encodings match those of the Cortex-A9 processors.

6.1.5 Accelerator Coherency Port

The *Accelerator Coherency Port* (ACP) is an optional AXI 64-bit slave port that can be connected to a DMA engine or a non-cached peripheral.

This interface acts as a standard AMBA 3 AXI slave, and supports all standard read and write transactions without any additional coherence requirements placed on attached components.

The accelerator coherency slave port can:

- do noncoherent accesses that are forwarded to an AXI master.

To indicate that an access is non-coherent use **AWUSER[0]** = 0 alongside **AWVALID** for write requests and **ARUSER[0]** = 0 alongside **ARVALID** for read requests. **AxCACHE** and other **AxUSER** signals are not interpreted by the ACP but only forwarded. Locked attributes for noncoherent ACP requests are kept.

- do coherent accesses that are forwarded to the SCU tag RAMs.

  To indicate that an access is coherent use **AWUSER[0]** = 1 alongside **AWVALID** for write requests and **ARUSER[0]** = 1 alongside **ARVALID** for read requests. **AxCACHE** and other **AxUSER** signals are not interpreted by the ACP but only forwarded. Locked attributes for coherent ACP requests are ignored.

Accesses that match the format of Cortex-A9 processor accesses done on cache lines are non-blocking. Those formats are:

- a wrapped burst of four doublewords, address aligned on a 64-bit boundary, with length = 3 and size = `0x3`.
- an incremental burst of four doublewords with the first address aligned on a 32-byte boundary.

Caution

For these non-blocking accesses the data strobes must all be set. If the data strobes are not all set, this can result in data corruption.

Write accesses

For write transactions to any coherent memory region, the SCU enforces coherence before the write is forwarded to the memory system. The transaction might also optionally allocate into the L2 cache.

Read accesses

For read transactions to any coherent memory region, the SCU enforces coherence.

Hazards

The ACP is a standard AXI slave port. As such, any hazard must be treated according to the *AMBA AXI Protocol v1.0 Specification* for every master connected to this slave port.

WFE/SEV synchronization

A peripheral connected on the coherency port or any other external agent can participate in the WFE/SEV event communication of the Cortex-A9 MPCore processor by using the **EVENTI** pin. When this pin is asserted, it sends an event message to all the Cortex-A9 processors in the cluster. This is similar to executing a SEV instruction on one core of the cluster. This enables the external agent to signal to the cores that it has released a semaphore and that the processors can leave the power saving mode. The **EVENTI** input pin must remain high at least one **CPUCLK** clock cycle to be visible by the processors.

The external agent can see that at least one of the Cortex-A9 internal processor has executed an SEV instruction by checking the **EVENTO** pin. This pin is set high for one **CPUCLK** clock cycle when any of the internal Cortex-A9 processor executes an SEV instruction.

6.2 Exclusive L2 cache

The Cortex-A9MPCore processor can be connected to an L2 cache that supports an exclusive cache mode. This mode must be activated both in the Cortex-A9 MPCore processor and in the L2 cache controller.

See the Auxiliary Control Register, ACTLR, in the *Cortex-A9 TRM*.

In this mode, the data cache of the Cortex-A9 processor and the L2 cache are exclusive. At any time, a given address can only be cached in either L1 data caches or in the L2 cache, but not in both. This has the effect of greatly increasing the usable space and efficiency of an L2 cache connected to the Cortex-A9 processor. When exclusive cache configuration is selected:

- Data cache line replacement policy is modified so that a victim line always gets evicted to L2 memory, even if it is clean.
- If an address is dirty in the L2 cache controller, a read request to this address from the core causes a writeback to external memory.

6.3 Using the STRT instruction

Table 6-6 shows core modes and corresponding **APROT** values.

Table 6-6 Core mode and APROT values

User or Privileged core mode	Type of access	Value of APROT
User	Cacheable read access	User
Privileged		Privileged
User	Non-cacheable read access	User
Privileged		Privileged
-	Cacheable write access	Always marked as Privileged
User	Non-cacheable write access	User
Privileged	Non-cacheable write access	Privileged, except when using STRT

Take particular care with non-cacheable write accesses when using the STRT instruction. To put the correct information on the external bus ensure one of the following:

- The access is to Strongly-ordered memory.

  This ensures that the STRT instruction does not merge in the store buffer.

- The access is to Device memory.

  This ensures that the STRT instruction does not merge in the store buffer.

- A drain write buffer command is issued before the STRT and after the STRT.

  This prevents an STRT from merging into an existing slot at the same 64-bit address, or merging with another write at the same 64-bit address.

Chapter 7
Clocking, Resets, and Power Management

This chapter describes the clocking and reset options available for Cortex-A9 MPCore processors. It contains the following sections:

7.1 Clocking

The Cortex-A9 MPCore processor has these functional clock inputs:

CLK

This is the main clock of the Cortex-A9 processor.

All Cortex-A9 processors in the Cortex-A9 MPCore processor and the SCU are clocked with a distributed version of CLK.

PERIPHCLK

The Interrupt Controller, and private timers and watchdog are clocked with **PERIPHCLK**.

PERIPHCLK must be synchronous with **CLK**, and the **PERIPHCLK** clock period, N, must be configured as a multiple of the **CLK** clock period. This multiple N must be equal to, or greater than two.

PERIPHCLKEN

This is the clock enable signal for the Interrupt Controller and timers. The PERIPHCLKEN signal is generated at **CLK** clock speed.
PERIPHCLKEN HIGH on a CLK rising edge indicates that there is a corresponding **PERIPHCLK** rising edge.

Figure 7-1 shows an example with the PERIPHCLK clock period N as three.

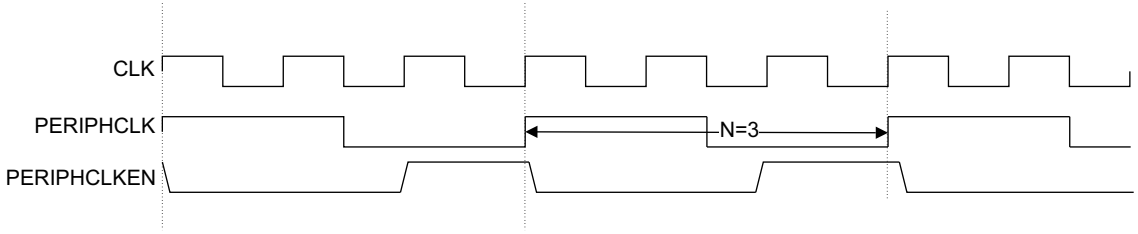


Figure 7-1 Three-to-one timing ratio

7.1.1 Synchronous clocking

The Cortex-A9 MPCore processor does not have any asynchronous interfaces. So, all the bus interfaces and the interrupt signals must be synchronous with reference to **CLK**.

AXI master interface clocking

The Cortex-A9 MPCore Bus Interface Unit supports the following AXI bus ratios:

- Integer ratios through clock enable (1:1, 2:1, 3:1, …)

- Half-integer ratios through clock enable: 1.5, 2.5 and 3.5 ratios.

These ratios are configured through external pins and CP15 registers. In all cases AXI transfers remain synchronous. There is no requirement for an asynchronous AXI interface with integer and half integer ratios. To support this, the following signals qualify input and output signals on AXI:

- **INCLKLENM0** and **OUTCLKLENM0**
- **INCLKLENM1** and **OUTCLKLENM1**.

Figure 7-2 shows the master clock and slave clock with a timing ratio of three to two.

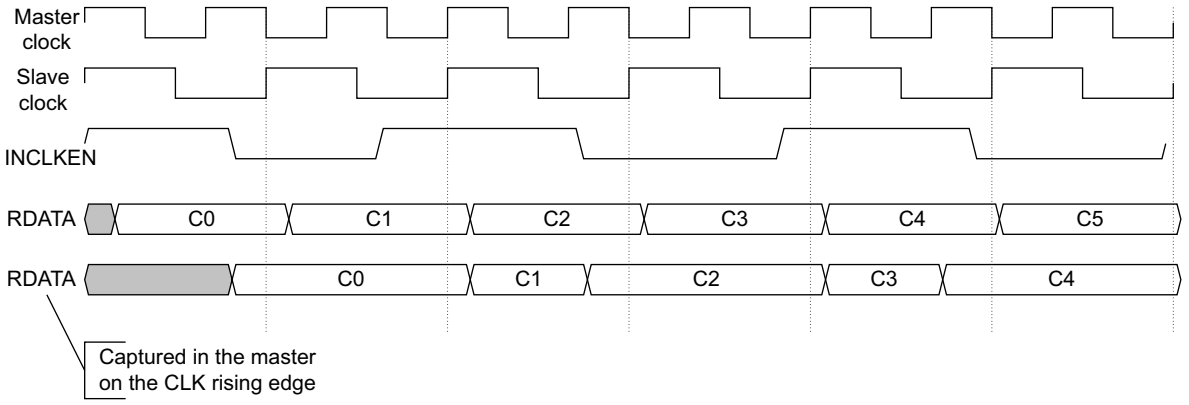


Figure 7-2 Three-to-two ratio

Figure 7-3 shows the master clock and slave clock with a timing ratio of five to two. The INCLKEN generator can use a synchronization counter that determines when **INCLKEN** is asserted. In this case that is when the counter has the value one or four.

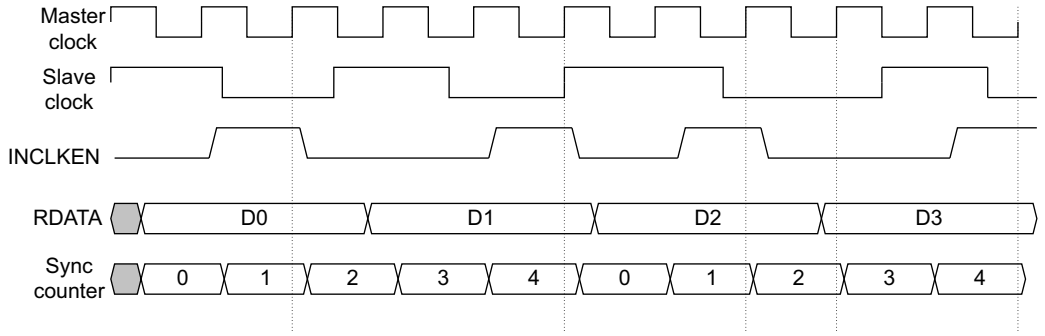


Figure 7-3 Five-to-two ratio

ACP clocking

The ACP supports integer AXI bus ratios only.

Figure 7-4 shows a timing example where ACKLENS is used with a 3:1 clock ratio between **CLK** and the ACP AXI clock, **ACLK**.

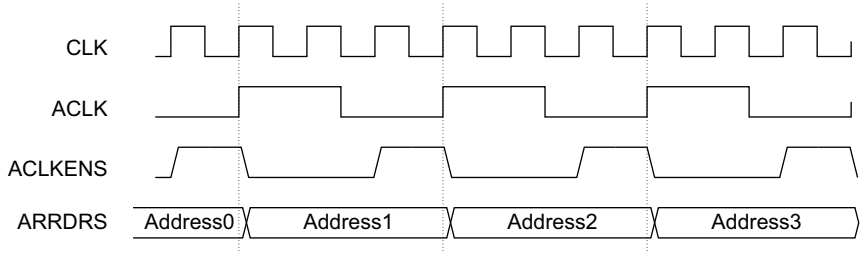


Figure 7-4 ACLKENS timing example

The ACP slave port samples the AXI input requests, and the AXI output values, only on the rising edge of **CLK** when **ACLKENS** is HIGH.

7.2 Reset

The Cortex-A9 MPCore processor has multiple reset domains with the following reset inputs:

nCPURESET[3:0] The **nCPURESET[3:0]**signals are the Cortex-A9 processor resets that initialize the majority of the Cortex-A9 processor logic, except the CP14 debug logic.

nDBGRESET[3:0] The **nDBGRESET** signals are the resets that initialize the CP14 debug logic for each Cortex-A9 processor.

nDERESET[3:0] The **nDERESET[3:0]** signals are the resets that initialize the CP14 debug logic for each Cortex-A9 processor.

nSCURESET The **nSCURESET** signal is the reset that initializes the majority of SCU logic in the Cortex-A9 MPCore processor.

nPERIPHRESET

The **nPERIPHRESET** signal is the reset that initializes the Interrupt Controller and the Timers.

nWDRESET[3:0] The **nWDRESET[3:0]** signals are used to individually reset each watchdog reset status flag. These resets must be asserted for power-on reset but not if one Cortex-A9 reset assertion is caused by a watchdog reset request with **WDRESETREQ**.

All of these are active LOW signals that reset logic in the Cortex-A9 MPCore processor.

7.3 Reset modes

The reset signals present in the Cortex-A9 MPCore processor design enable you to reset different parts of the design independently.

7.3.1 Power-on reset

You must apply power-on or *cold* reset to the Cortex-A9 MPCore processor when power is first applied to the system. In the case of power-on reset, the leading (falling) edge of the reset signals, **nSCURESET, nCPURESET[3:0]** and **nWDRESET[3:0]**, do not have to be synchronous to **CLK** but the rising edge must be.

You must assert the reset signals for at least nine **CLK** cycles to ensure correct reset behavior.

On power-on, perform the following reset sequence:

1. Apply all resets.
2. Apply at least nine **CLK** cycles, plus at least one cycle in each other clock domain, or more if the documentation for other components requests it. There is no harm in applying more clock cycles than this, and maximum redundancy can be achieved by for example applying 15 cycles on every clock domain.
3. Stop the **CLK** clock.
4. Wait for the equivalent of approximately 10 cycles, depending on your implementation. This compensates for clock and reset tree latencies.
5. Release resets.
6. Wait for the equivalent of another approximately 10 cycles, again to compensate for clock and reset tree latencies.
7. Restart the **CLK** clock.

7.3.2 Cortex-A9 MPCore reset

A processor or *warm* reset initializes the majority of the logic of a Cortex-A9 processor, apart from its debug logic. Breakpoints and watchpoints are retained during a warm reset.

An additional reset,**nDERESET[3:0]** controls each Data Engine unit independently of each Cortex-A9 processorreset.The Data Engine can be either the FPU or the MPE. This reset can be used to hold the Data Engine in a reset state so that the power to the Data Engine can be safely removed without placing any logic within the Data Engine unit in a different state.

7.3.3 Debug reset

nDBGRESET resets the debug hardware within each Cortex-A9 processor of the Cortex-A9 MPCore processor, including breakpoints and watchpoints values.

Table 7-1 shows the reset states for Cortex-A9 MPCore processors. [n] refers to the Cortex-A9 processor that initiates a reset.

Table 7-1 Cortex-A9 MPCore reset signals

Signals	Resets					Normal
	Power-on or cold reset	DAP-requested	Individual power-on	Software	From watchdog	
nPRESETDBG	0	0	1	1	1	1
nSCURESET nPERIPHRESET	0	1	1	1	0	1
nDBGRESET[3:0]	All 0	All 0	[n]=0	All 1	All 1	All 1
nWDRESET[3:0]	All 0	All 1	[n]=0	[n]=0	All 1	All 1
nCPURESET[3:0] nDERESET[3:0]	All 0	All 1	[n]=0	[n]=0	All 0	All 1

7.4 About power consumption control

The features of the Cortex-A9 MPCore processor that improve energy efficiency include:

- accurate branch and return prediction, reducing the number of incorrect instruction fetch and decode operations
- use of physically addressed caches, reducing the number of cache flushes and refills, saving energy in the system
- the use of MicroTLBs reduces the power consumed in translation and protection lookups each cycle
- the caches use sequential access information to reduce the number of accesses to the Tag RAMs and to unwanted Data RAMs.

In the Cortex-A9 MPCore processor extensive use is also made of gated clocks and gates to disable inputs to unused functional blocks. Only the logic actively in use to perform a calculation consumes any dynamic power.

7.5 Individual Cortex-A9 Processor Power Control

Place holders for clamps are inserted around each Cortex-A9 processor so that implementation of different power domains can be eased. It is the responsibility of software to signal to the Snoop Control Unit and the Distributed Interrupt Controller that a Cortex-A9 processor is shut off so that the Cortex-A9 processor can be seen as non-existent in the cluster. Each Cortex-A9 processor can be in one of the following modes:

Run mode Everything is clocked and powered-up

WFI mode The CPU clock is stopped. Only logic required for wake-up is still active.

Dormant mode

Everything is powered off except RAM arrays that are in retention mode.

Shutdown Everything is powered-off.

Table 7-2 shows the individual power modes.

Table 7-2 Cortex-A9 MPCore power modes

Mode	Cortex-A9 processor logic	RAM arrays	Wake-up mechanism
Run Mode	Powered-up Everything clocked	Powered-up	N/A
WFI/WFE	Powered-up Only wake-up logic clocked	Powered-up	Wake-up on interrupts (external or timer/WD). L1 memory system only wake-up in case of SCU coherency request.
Dormant	Powered-off	Retention state/voltage	External wake-up event to power controller.
Shutdown	Powered-off	Powered-off	External wake-up event to power controller.

Entry to Dormant or powered-off mode must be controlled through an external power controller. The CPU Status Register in the SCU is used in conjunction with CPU WFI entry flag to signal to the power controller the power domain that it can cut, using the PWRCTL bus. See *SCU CPU Power Status Register* on page 3-7.

7.5.1 Run mode

Run mode is the normal mode of operation, where all of the functionality of the Cortex-A9 processor is available.

7.5.2 Wait for interrupt (WFI/WFE) mode

Wait for Interrupt mode disables all the clocks of a Cortex-A9 processor, while keeping its logic powered up. This reduces the power drawn to the static leakage current.

The transition from the WFI mode to the Run mode is caused by:

- an interrupt, masked or unmasked
- an imprecise data abort, regardless of the value of the CPSRA bit
- a debug request, regardless of whether debug is enabled
- a cp15 forwarding request from another processor
- a reset.

The transition from the WFE mode to the Run mode is caused by:

- an interrupt, unless masked
- a debug request, regardless of whether debug is enabled
- a cp15 forwarding request from another processor
- a previous exception return on the same processor
- a reset
- the assertion of the **EVENTI** input signal
- the execution of an SEV instruction on any processor in a multiprocessor system.

The debug request can be generated by an externally generated debug request, using the **EDBGRQ** pin on the Cortex-A9 processor, or from a Debug Halt instruction issued to the Cortex-A9 processor through the Debug ABP bus.

Entry into WFI Mode is performed by executing the Wait For Interrupt instruction.

Entry into WFE Mode is performed by executing the Wait For Event instruction.

To ensure that the memory system is not affected by the entry into the Standby state, the following operations are performed:

- A Data Synchronization Barrier, so ensuring that all explicit memory accesses occurring in program order before the WFI has completed. This avoids any possible deadlocks that could be caused in a system where memory access triggers or enables an interrupt that the CPU is waiting for.
- Any other memory accesses that have been started at the time that the WFI instruction is executed are completed as normal. This ensures that the Level 2 memory system does not see any disruption caused by the WFI.
- The debug bus remains active throughout a WFI.

7.5.3 Dormant mode

Dormant mode is designed to enable the Cortex-A9 processor to be powered down, while leaving the caches powered up and maintaining their state.

The RAM blocks that are to remain powered up must be implemented on a separate power domain, and there is a requirement to clamp all of the inputs to the RAMs to a known logic level (with the chip enable being held inactive). This clamping is not implemented in gates as part of the default synthesis flow because it would contribute to a tight critical path. Implementations that want to implement Dormant mode must add these clamps around the RAMs, either as explicit gates in the RAM power domain, or as pull-down transistors that clamp the values while the Cortex-A9 processor is powered down.The RAM blocks that must remain powered up during Dormant mode are:

- all Data RAMs associated with the cache
- all Tag RAMs associated with the cache

Before entering Dormant mode, the state of the Cortex-A9 processor, excluding the contents of the RAMs that remain powered up in dormant mode, must be saved to external memory. These state saving operations must ensure that the following occur:

- All ARM registers, including CPSR and SPSR registers are saved.
- All CP15 registers are saved.
- All debug-related state must be saved.
- the Cortex-A9 processor must correctly set the CPU Status Register in the SCU so that it enters Dormant Mode. See *SCU CPU Power Status Register* on page 3-7.
- A Data Synchronization Barrier instruction is executed to ensure that all state saving has been completed.
- The Cortex-A9 processor then communicates with the power controller that it is ready to enter dormant mode by performing a WFI instruction so that power control output reflects the value of SCU CPU Status Register (see *SCU CPU Power Status Register* on page 3-7).
- On entry into Dormant mode, the Reset signal to the Cortex-A9 processor must be asserted by the external power control mechanism.

Transition from Dormant mode to Run mode is triggered by the external power controller. The external power controller must assert reset to the Cortex-A9 processor until the power is restored. After power is restored, the Cortex-A9 processor leaves reset, and by interrogating the power control register in SCU, can determine that the saved state must be restored.

7.5.4 Shutdown mode

Shutdown mode has the entire device powered down, and all state, including cache, must be saved externally by software. The part is returned to the run state by the assertion of reset. This state saving is performed with interrupts disabled, and finishes with a DSB operation. The Cortex-A9 processor then communicates with a power controller that the device is ready to be powered down in the same manner as when entering Dormant Mode.

7.5.5 Communication to the Power Management Controller

Communication between the Cortex-A9 processor and the external Power Management Controller can be performed using the **PWRCTLOn** Cortex-A9 MPCore output signals and Cortex-A9 MPCore input clamp signals.

PWRCTLOn Cortex-A9 MPCore output signals

These signals constrain the external Power Management Controller. The value of **PWRCTLOn** depends on the value of the SCU CPU Status Register (see *SCU CPU Power Status Register* on page 3-7). The SCU CPU Status Register value is only copied to **PWRCTLOn** after the Cortex-A9 processor signals that it is ready to entry low power mode by executing a WFI instruction and subsequent **STANDBYWFI** pin assertion.

Cortex-A9 MPCore input signals

The external Power Management Controller uses **BISTCLAMP**, **DEBUGCLAMP**, **CPUCLAMP[3:0]**, **DECLAMP[3:0]**, and **RAMCLAMP[4:0]** to isolate Cortex-A9 MPCore power domains from one another before they are turned off. These signals are only meaningful if the Cortex-A9 MPCore processor has been implemented with power clamps designed in.

7.6 IEM Support

The IEM infrastructure is intended to be supported at the system level to enable you to choose the level to use in the SOC to separate different power domains.

There are placeholders between Cortex-A9 MPCore logic and RAM arrays so that implementation of level-shifters for these parts can be on a different power domain.

7.6.1 Cortex-A9 MPCore power domains

The Cortex-A9 MPCore processor can support up to fourteen power domains:

- four power domains, one for each of the Cortex-A9 processors, apart from their Data Engines
- four power domains, one for each of the Cortex-A9 processor Data Engines
- four power domains, one for each of the Cortex-A9 processor caches and TLB RAMs
- one power domain for SCU duplicated TAG RAMs
- one power domain for remaining logic, the SCU logic cells, and private peripherals.

Figure 7-5 shows all the power domains and where placeholders are inserted for power domain isolation.

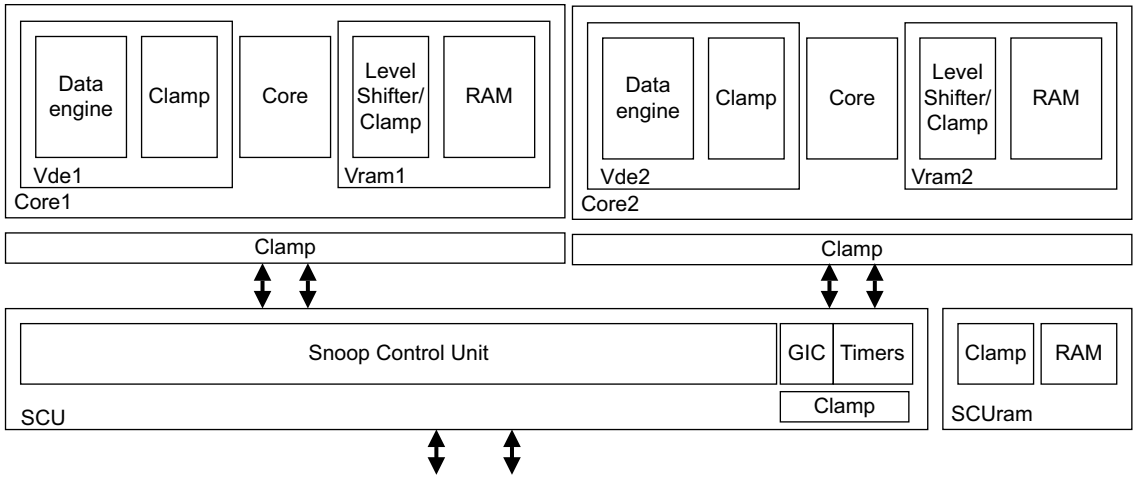


Figure 7-5 Cortex-A9 MPCore processor power management

Chapter 8
Cortex-A9 MPCore Debug and Trace

This chapter describes some of the debug and trace considerations in Cortex-A9 MPCore designs. It contains the following sections:

8.1 External Interface Signals

Figure 8-1 shows the CortexA9 MP external interface signals.

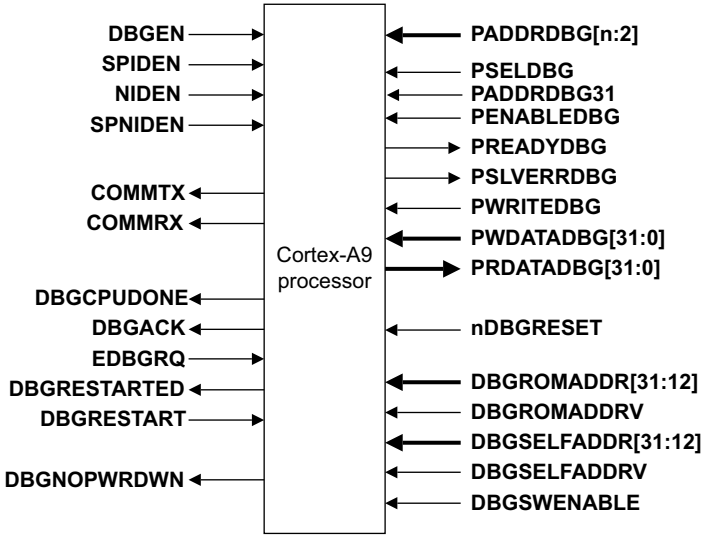


Figure 8-1 External debug interface signals in CortexA9 MPCore designs

In CortexA9 MPCore designs the width of **PADDRDBG** varies according to the number of Cortex-A9 processors present as Table 8-1shows.

Table 8-1 PADDRDBG width

PADDRDBG width	Cortex-A9 processors
PADDRDBG[12:2]	One
PADDRDBG[13:2]	Two
PADDRDBG[14:2]	Three
PADDRDBG[14:2]	Four

The CortexA9 MPCore external debug interface does not implement:

- **DBGTRIGGER**
- **DBGPWRDUP**
- **DBGOSLOCKINIT**.

8.2 CortexA9 MPCore CoreSight Memory Map

In a CoreSight system the ROM table provides information about the debug components.

You can use the *Debug Self Address Offset Registers* (DBGDSARs) and *Debug ROM Address Register* (DBGDRAR) to define the base address of the ROM table and location of the debug components and the individual Cortex-A9 processors in the system.

To select the CortexA9MPCore itself use **PSELDBG**.

To access the individual Cortex-A9 processors in the system use **PADDRDBG[14:13]** with the following values:

- 00 for access to the CPU0 components
- 01 for access to the CPU1 components
- 10 for access to the CPU2 components
- 11 for access to the CPU3 components.

Use **PADDRDBG[12]** = 0 to access the debug area of the selected Cortex-A9 processor.

Use **PADDRDBG[12]** = 1 to access the *Performance Monitoring Unit* (PMU) area of the selected Cortex-A9 processor.

Appendix A
Signal Descriptions

This appendix describes the Cortex-A9 MPCore signals. It contains the following sections:

A.1 Clock signals

Table A-1 shows the clock signals.

Table A-1 Cortex-A9 MPCore clocks

Signal	I/O	Description
CLK	I	Global clock
PERIPHCLK	I	Clock for the timer and Interrupt Controller
PERIPHCLKEN	I	Clock enable for the timer and Interrupt Controller

See Chapter 7 *Clocking, Resets, and Power Management*.

A.2 Resets

Table A-2 shows the reset signals.

Table A-2 Reset signals

Signal	I/O	Description
nCPURESET[3:0]	I	Individual Cortex-A9 processor resets
nSCURESET	I	SCU global reset
nWDRESET[3:0]	I	Individual watchdog resets
nPERIPHRESET	I	Timer and interrupt controller reset
nDBGRESET[3:0]	I	Compute engine debug logic reset
nDERESET[3:0]	I	Date engines reset

See Chapter 7 *Clocking, Resets, and Power Management*.

Table A-3 shows the watchdog request reset signal.

Table A-3 Watchdog request reset signal

Signal	I/O	Description
WDRESETREQ[3:0]	O	Watchdog reset request

See Chapter 5 *Timer and Watchdog Registers*.

A.3 Interrupt lines

Table A-4 shows the interrupt line signals.

Table A-4 Interrupt line signals

Signal	I/O	Description
INT[n:0][a]	I	Interrupt distributor interrupt lines. n can be 31, 63,…, up to 223 by increments of 32. If there are no interrupt lines this pin is removed. See Chapter 4 *Interrupt Controller*.
nIRQ[3:0]	I	Cortex-A9 processor legacy IRQ request input lines. Active-LOW interrupt request: 0 = activate interrupt 1 = do not activate interrupt. The processor treats the **nIRQ** input as level sensitive. The **nIRQ** input must be asserted until the processor acknowledges the interrupt.
nFIQ[3:0]	I	Cortex-A9 processor private FIQ request input lines. Active-LOW fast interrupt request: 0 = activate fast interrupt 1 = do not activate fast interrupt. The processor treats the **nFIQ** input as level sensitive. The **nFIQ** input must be asserted until the processor acknowledges the interrupt.

a. The minimum pulse width of signals driving external interrupt lines is onePERIPH**CLK** cycle. Because this signal is level-sensitive, to generate an interrupt you must ensure it is held LOW until the processor sends a suitable interrupt response.

A.4 Configuration signals

Table A-5 shows the configuration signals.

Table A-5 Configuration signals

Signal	I/O	Description
CFGEND[3:0]	I	Endianness configuration. Forces the EE bit in the CP15 c1 Control Register (SCTLR) to 1 at reset so that the Cortex-A9 processor boots with big-endian data handling. 0 = EE bit is LOW 1 = EE bit is HIGH This pin is only sampled during reset of the processor.
CFGNMFI[3:0]	I	Configures fast interrupts to be nonmaskable: 0 = clear the NMFI bit in the CP15 c1 Control Register 1 = set the NMFI bit in the CP15 c1 Control Register. This pin is only sampled during reset of the processor.
CFGSDISABLE	I	Disables write access to some system control processor registers: 0 = not enabled 1 = enabled. See *Using CFGSDISABLE* on page 4-12.
CLUSTERID[3:0]	I	Value read in Cluster ID register field, bits[11:8] of the MPIDR.
CP15SDISABLE[3:0]	I	Disables write access to some system control processor registers.
FILTEREN	I	For use with configurations with two master ports. Enables filtering of address ranges at reset. See *SCU Control Register* on page 3-4 for information on setting this signal.
FILTERSTART[31:20]	I	For use with configurations with two master ports. Specifies the start address for address filtering at reset. See *Filtering Start Address Register* on page 3-10.
FILTEREND[31:20]	I	For use with configurations with two master ports. Specifies the end address for address filtering. See *Filtering End Address Register* on page 3-10.
PERIPHBASE[31:13]	I	Specifies the base address for Timers, Watchdogs, Interrupt Controller, and SCU registers. Only accessible with memory-mapped accesses. This value can be retrieved by a Cortex-A9 processor using the cp15 Configbase Register.

Table A-5 Configuration signals (continued)

Signal	I/O	Description
SMPnAMP[3:0]	O	Signals AMP or SMP mode for each Cortex-A9 processor.
TEINIT[3:0]	I	Default exception handling state. When set to: 0 = ARM 1 = Thumb
VINITHI[3:0]	I	Controls the location of the exception vectors at reset: 0 = start exception vectors at address `0x00000000` 1 = start exception vectors at address `0xFFFF0000`. This pin is only sampled during reset of the processor.

A.5 Standby and wait for event signals

Table A-6 shows the standby and wait for event signals.

Table A-6 Standby and wait for event signals

Signal	I/O	Description
EVENTI	I	Event input for Cortex-A9 processor wake-up from WFE state.
EVENTO	O	Event output. This signal is active when one SEV instruction is executed.
STANDBYWFE[3:0]	O	Indicates if a Cortex-A9 processor is in WFE state.
STANDBYWFI[3:0]	O	Indicates that a Cortex-A9 processor is in Standby mode.

See *Individual Cortex-A9 Processor Power Control* on page 7-9.

A.6 Power management signals

Table A-7 shows power control interface signals.

Table A-7 Power control interface signals

Signal	I/O	Description
BISTCLAMP	I	BIST interface clamp control signal.
CPUCLAMP[3:0]	I	Interrupt interface clamps control signals: **CPUCLAMP[3]** CPU3 interface **CPUCLAMP[2]** CPU2 interface **CPUCLAMP[1]** CPU1 interface **CPUCLAMP[0]** CPU0 interface.
CPURAMCLAMP[3:0]	I	Enables the clamp cells in Dormant mode.
SCURAMCLAMP	I	Enables the SCU clamp cells in Dormant mode.
DECLAMP[3:0]	I	Enables the Data Engine clamp cells in Dormant mode.
PWRCTLI0[1:0]	I	Reset value for CPU0 status register [1:0.]
PWRCTLI1[1:0]	I	Reset value for CPU1 status register [3:2].
PWRCTLI2[1:0]	I	Reset value for CPU2 status register [5:4].
PWRCTLI3[1:0]	I	Reset value for CPU3 status register [7:6].
PWRCTLO0[1:0]	O	b0x CPU0 must be powered on b10 CPU0 can enter dormant mode b11 CPU0 can enter powered-off mode.

Table A-7 Power control interface signals (continued)

Signal	I/O	Description
PWRCTLO1[1:0]	O	b0x CPU1 must be powered on b10 CPU1 can enter dormant mode b11 CPU1 can enter powered-off mode. This signal exists only if CPU1 is present.
PWRCTLO2[1:0]	O	b0x CPU2 must be powered on b10 CPU2 can enter dormant mode b11 CPU2 can enter powered-off mode. This signal exists only if CPU2 is present.
PWRCTLO3[1:0]	O	b0x CPU3 must be powered on b10 CPU3 can enter dormant mode b11 CPU3 can enter powered-off mode This signal exists only if CPU3 is present.

See *SCU CPU Power Status Register* on page 3-7 and *Communication to the Power Management Controller* on page 7-12.

A.7 AXI interfaces

In Cortex-A9 designs there can be two AXI master ports and an Accelerator Coherence Port, an AXI slave. The following sections describe the AXI interfaces:

- *AXI Master0 signals*
- *AXI Master1 signals* on page A-15
- *AXI ACP signals* on page A-20.

A.7.1 AXI Master0 signals

The following sections describe the AXI Master0 interface signals:

- *Write address signals for AXI Master0*
- *Write data channel signals* on page A-12
- *Write response channel signals* on page A-12
- *Read data channel signals* on page A-13
- *Read data channel signals* on page A-14
- *AXI Master0 Clock enable signals* on page A-15.

Write address signals for AXI Master0

Table A-14 on page A-15 shows the write address signals for AXI Master0.

Table A-8 Write address signals for AXI Master0

Signal	I/O	Description
AWADDRM0[31:0]	O	Address.
AWBURSTM0[1:0]	O	Burst type: b01 = INCR incrementing burst b10 = WRAP Wrapping burst. Other values are Reserved.
AWCACHEM0[3:0]	O	Cache type giving additional information about cacheable characteristics.
AWIDM0[5:0]	O	Request ID See *AXI USER encodings* on page 6-5.
AWLENM0[3:0]	O	Burst length that gives the exact number of transfers: b0000 = 1 data transfer b0001 = 2 data transfers b0010 = 3 data transfers b0011 = 4 data transfers.

Table A-8 Write address signals for AXI Master0 (continued)

Signal	I/O	Description
AWLOCKM0[1:0]	O	Lock type: b00 = normal access b01 = exclusive access b10 = locked access.
AWPROTM0[2:0]	O	Protection Type.
AWREADYM0	I	Address ready.
AWSIZEM0[1:0]	O	Burst size: b000 = 8-bit transfer b001 = 16-bit transfer b010 = 32-bit transfer b011 = 64-bit transfer.
AWUSERM0[8:0]	O	Sideband information: [0] shared [4:1] inner attributes b0000 = Strongly-ordered b0001 = Device b0011 = Normal Memory Non-Cacheable b0110 = Write-Through b0111 = Write-Back no Write Allocate b1111 = Write-Back Write Allocate [5] level 1 eviction [6] clean eviction [7] reserved [8] reserved. See *AXI USER encodings* on page 6-5.
AWVALIDM0	O	Address valid.

Write data channel signals

Table A-15 on page A-17 shows the write data signals for AXI Master0.

Table A-9 Write data signals for AXI Master0

Signal	I/O	Description
WDATAM0[63:0]	O	Write data
WIDM0[5:0]	O	Write ID
WLASTM0	O	Write last indication
WREADYM0	I	Write ready
WSTRBM0[7:0]	O	Write byte lane strobe
WVALIDM0	O	Write valid

Write response channel signals

Table A-16 on page A-17 shows the write response signals for AXI Master0.

Table A-10 Write response signals for AXI Master0

Signal	I/O	Description
BIDM0[5:0]	I	Response ID
BREADYM0	O	Response ready
BRESPM0[1:0]	I	Write response
BVALIDM0	I	Response valid

Read data channel signals

Table A-17 on page A-18 shows the read address signals for AXI Master0.

Table A-11 Read address signals for AXI Master0

Signal	I/O	Description
ARADDRM0[31:0]	O	Address.
ARBURSTM0[1:0]	O	Burst type: b01 = INCR incrementing burst b10 = WRAP Wrapping burst. Other values are Reserved.
ARCACHEM0[3:0]	O	Cache type giving additional information about cacheable characteristics.
ARIDM0[5:0]	O	Request ID See *AXI USER encodings* on page 6-5
ARLENM0[3:0]	O	Burst length that gives the exact number of transfers: b0000 = 1 data transfer b0001 = 2 data transfers b0010 = 3 data transfers b0011 = 4 data transfers.
ARLOCKM0[1:0]	O	Lock type: b00 = normal access b01 = exclusive access b10 = locked access.
ARPROTM0[2:0]	O	Protection Type
ARREADYM0	I	Address ready.

Table A-11 Read address signals for AXI Master0 (continued)

Signal	I/O	Description
ARSIZEM0[1:0]	O	Burst size: b000 = 8-bit transfer b001 = 16-bit transfer b010 = 32-bit transfer b011 = 64-bit transfer.
ARUSERM0[6:0]	O	Sideband information: [0] shared bit [4:1] inner attributes b0000 = Strongly-ordered b0001 = Device b0011 = Normal Memory Non-Cacheable b0110 = Write-Through b0111 = Write-Back no Write Allocate b1111 = Write-Back Write Allocate [5] prefetch hint [6] reserved. See *AXI USER encodings* on page 6-5.
ARVALIDM0	O	Address valid.

Read data channel signals

Table A-18 on page A-19 shows the read data signals for AXI Master0.

Table A-12 Read data signals for AXI Master0

Signal	I/O	Description
RVALIDM0	I	Read valid
RDATAM0[63:0]	I	Read data
RRESPM0[1:0]	I	Read response
RLASTM0	I	Read Last indication
RIDM0[5:0]	I	Read ID
RREADYM0	O	Read ready

AXI Master0 Clock enable signals

This section describes the AXI Master0 clock enable signals.

Table A-13 AXI Master0 clock enable signals

Signal	I/O	Source	Description
INCLKENM0	I	Clock generator	Clock enable for the AXI bus that enables the AXI interface to operate at either: • integer ratios of the system clock • half integer ratios of the system clock. See Chapter 7 *Clocking, Resets, and Power Management*.
OUTCLKENM0	I		Clock enable for the AXI bus that enables the AXI interface to operate at either: • integer ratios of the system clock • half integer ratios of the system clock. See Chapter 7 *Clocking, Resets, and Power Management*.

A.7.2 AXI Master1 signals

The following sections describe the AXI Master1 interface signals:

- *Write address signals for AXI Master1*
- *Write data channel signals* on page A-17
- *Write response channel signals* on page A-17
- *Read data channel signals* on page A-18
- *Read data channel signals* on page A-19
- *AXI Master1 Clock enable signals* on page A-20.

Write address signals for AXI Master1

Table A-14 shows the write address signals for AXI Master1.

Table A-14 Write address signals for AXI Master1

Signal	I/O	Description
AWADDRM1[31:0]	O	Address.
AWBURSTM1[1:0]	O	Burst type: b01 = INCR incrementing burst b10 = WRAP Wrapping burst. Other values are Reserved.
AWCACHEM1[3:0]	O	Cache type giving additional information about cacheable characteristics.

Table A-14 Write address signals for AXI Master1 (continued)

Signal	I/O	Description
AWIDM1[5:0]	O	Request ID
AWLENM1[3:0]	O	Burst length that gives the exact number of transfers: b0000 = 1 data transfer b0001 = 2 data transfers b0010 = 3 data transfers b0011 = 4 data transfers.
AWLOCKM1[1:0]	O	Lock type: b00 = normal access b01 = exclusive access b10 = locked access.
AWPROTM1[2:0]	O	Protection Type.
AWREADYM1	I	Address ready.
AWSIZEM1[1:0]	O	Burst size: b000 = 8-bit transfer b001 = 16-bit transfer b010 = 32-bit transfer b011 = 64-bit transfer.
AWUSERM1[8:0]	O	Sideband information: [8] reserved. [7] full line at 0 [6] clean eviction [5] level 1 eviction [4:1] inner attributes: b0000 = Strongly-ordered b0001 = Device b0011 = Normal Memory Non-Cacheable b0110 = Write-Through b0111 = Write-Back no Write Allocate b1111 = Write-Back Write Allocate [0] shared. See *AXI USER encodings* on page 6-5.
AWVALIDM1	O	Address valid.

Write data channel signals

Table A-15 shows the write data signals for AXI Master1.

Table A-15 Write data signals for AXI Master1

Signal	I/O	Description
WDATAM1[63:0]	O	Write data
WIDM1[5:0]	O	Write ID
WLASTM1	O	Write last indication
WREADYM1	I	Write ready
WSTRBM1[7:0]	O	Write byte lane strobe
WVALIDM1	O	Write valid

Write response channel signals

Table A-16 shows the write response signals for AXI Master1.

Table A-16 Write response signals for AXI Master1

Signal	I/O	Description
BIDM1[5:0]	I	Response ID
BREADYM1	O	Response ready
BRESPM1[1:0]	I	Write response
BVALIDM1	I	Response valid

Read data channel signals

Table A-17 shows the read address signals for AXI Master1.

Table A-17 Read address signals for AXI Master1

Signal	I/O	Description
ARADDRM1[31:0]	O	Address.
ARBURSTM1[1:0]	O	Burst type: b01 = INCR incrementing burst b10 = WRAP Wrapping burst. Other values are Reserved.
ARCACHEM1[3:0]	O	Cache type giving additional information about cacheable characteristics.
ARIDM1[5:0]	O	Request ID *ARIDMx[5:0] encodings* on page 6-4.
ARLENM1[3:0]	O	Burst length that gives the exact number of transfers: b0000 = 1 data transfer b0001 = 2 data transfers b0010 = 3 data transfers b0011 = 4 data transfers.
ARLOCKM1[1:0]	O	Lock type: b00 = normal access b01 = exclusive access b10 = locked access.
ARPROTM1[2:0]	O	Protection type
ARREADYM1	I	Address ready.

Table A-17 Read address signals for AXI Master1 (continued)

Signal	I/O	Description
ARSIZEM1[1:0]	O	Burst size: b000 = 8-bit transfer b001 = 16-bit transfer b010 = 32-bit transfer b011 = 64-bit transfer.
ARUSERM1[6:0]	O	Sideband information: [6] reserved. [5]prefetch hint [4:1] inner attributes: b0000 = Strongly-ordered b0001 = Device b0011 = Normal Memory Non-Cacheable b0110 = Write-Through b0111 = Write-Back no Write Allocate b1111 = Write-Back Write Allocate [0] shared bit *ARUSERMx[6:0] encodings* on page 6-5
ARVALIDM1	O	Address valid.

Read data channel signals

Table A-18 shows the read data signals for AXI Master1.

Table A-18 Read data signals for AXI Master1

Signal	I/O	Description
RVALIDM1	I	Read valid
RDATAM1[63:0]	I	Read data
RRESPM1[1:0]	I	Read response
RLASTM1	I	Read Last indication
RIDM1[5:0]	I	Read ID
RREADYM1	O	Read ready

AXI Master1 Clock enable signals

This section describes the AXI Master1 clock enable signals.

Table A-19 AXI Master1 clock enable signals

Signal	I/O	Source	Description
INCLKENM1	I	Clock generator	Clock enable for the AXI bus that enables the AXI interface to operate at either: • integer ratios of the system clock • half integer ratios of the system clock. See Chapter 7 *Clocking, Resets, and Power Management*.
OUTCLKENM1	I		Clock enable for the AXI bus that enables the AXI interface to operate at either: • integer ratios of the system clock • half integer ratios of the system clock. See Chapter 7 *Clocking, Resets, and Power Management*.

A.7.3 AXI ACP signals

The following sections describe the AXI ACP interface signals:

- *Write address signals for AXI ACP*
- *Write data channel signals* on page A-22
- *Write response channel signals* on page A-22
- *Read data channel signals* on page A-23
- *Read data channel signals* on page A-24
- *ACLKENS* on page A-25.

Write address signals for AXI ACP

Table A-14 on page A-15 shows the AXI write address signals for AXI ACP.

Table A-20 Write address signals for AXI ACP

Signal	I/O	Description
AWADDRS[31:0]	I	Address.
AWBURSTS[1:0]	I	Burst type: b01 = INCR incrementing burst b10 = WRAP Wrapping burst. Other values are Reserved.
AWCACHES[3:0]	I	Cache type giving additional information about cacheable characteristics.

Table A-20 Write address signals for AXI ACP (continued)

Signal	I/O	Description
AWIDS[2:0]	I	Request ID
AWLENS[3:0]	I	Burst length that gives the exact number of transfers: b0000 = 1 data transfer b0001 = 2 data transfers b0010 = 3 data transfers b0011 = 4 data transfers.
AWLOCKS[1:0]	I	Lock type: b00 = normal access b01 = exclusive access b10 = locked access.
AWPROTS[2:0]	I	Protection Type.
AWREADYS	O	Address ready.
AWSIZES[1:0]	I	Burst size: b000 = 8-bit transfer b001 = 16-bit transfer b010 = 32-bit transfer b011 = 64-bit transfer.
AWUSERS[6:0]	I	Sideband information: [6] cacheable dside. [5] inner shared [4:1] inner attributes: b0000 = Strongly-ordered b0001 = Device b0011 = Normal Memory Non-Cacheable b0110 = Write-Through b0111 = Write-Back no Write Allocate b1111 = Write-Back Write Allocate [0] shared. See *AXI USER encodings* on page 6-5.
AWVALIDS	I	Address valid.

Write data channel signals

Table A-15 on page A-17 shows the AXI write data signals for AXI ACP.

Table A-21 Write data signals for AXI ACP

Signal	I/O	Description
WDATAS[63:0]	I	Write data
WIDS[2:0]	I	Write ID
WLASTS	I	Write last indication
WREADYS	O	Write ready
WSTRBS[7:0]	I	Write byte lane strobe
WVALIDS	I	Write valid

Write response channel signals

Table A-16 on page A-17 shows the AXI write response signals for AXI ACP.

Table A-22 Write response signals for AXI ACP

Signal	I/O	Description
BIDS[2:0]	O	Response ID
BREADYS	I	Response ready
BRESPS[1:0]	O	Write response
BVALIDS	O	Response valid

Read data channel signals

Table A-17 on page A-18 shows the AXI read address signals for AXI ACP.

Table A-23 Read address signals for AXI ACP

Signal	I/O	Description
ARADDRS[31:0]	I	Address.
ARBURSTS[1:0]	I	Burst type: b01 = INCR incrementing burst b10 = WRAP Wrapping burst. Other values are Reserved.
ARCACHES[3:0]	I	Cache type giving additional information about cacheable characteristics.
ARIDS[5:0]	I	Request ID
ARLENS[3:0]	I	Burst length that gives the exact number of transfers: b0000 = 1 data transfer b0001 = 2 data transfers b0010 = 3 data transfers b0011 = 4 data transfers.
ARLOCKS[1:0]	I	Lock type: b00 = normal access b01 = exclusive access b10 = locked access.
ARPROTS[2:0]	I	Protection Type
ARREADYS	O	Address ready

Table A-23 Read address signals for AXI ACP (continued)

Signal	I/O	Description
ARSIZES[1:0]	I	Burst size: b000 = 8-bit transfer b001 = 16-bit transfer b010 = 32-bit transfer b011 = 64-bit transfer.
ARUSERS[4:0]	I	Sideband information: [4:1] inner attribute bits: b0000 = Strongly-ordered b0001 = Device b0011 = Normal Memory Non-Cacheable b0110 = Write-Through b0111 = Write-Back no Write Allocate b1111 = Write-Back Write Allocate. [0] shared bit. See *AXI USER encodings* on page 6-5.
ARVALIDS	I	Address valid.

Read data channel signals

Table A-18 on page A-19 shows the AXI read data signals for AXI ACP.

Table A-24 Read data signals for AXI ACP

Signal	I/O	Description
RVALIDS	O	Read valid
RDATAS[63:0]	O	Read data
RRESPS[1:0]	O	Read response
RLASTS	O	Read Last indication
RIDS[2:0]	O	Read ID
RREADYS	I	Read ready

ACLKENS

Table A-25 shows the **ACLKEN** slave signal.

Table A-25 ACLKENS signal

Signal	I/O	Description
ACLKENS	I	Bus clock enable. See *ACP clocking* on page 7-4.

A.8 Performance monitoring signals

Table A-26 shows the performance monitoring signals. There are as many **PMUEVENT** buses as there are Cortex-A9 processors in the design.

Table A-26 Performance monitoring signals

Signal	I/O	Description
PMUEVENTn[28:0]	O	Performance Monitoring Unit event signals for CPUn. **PMUEVENTn[0]** `0x00` software increment **PMUEVENTn[1]** `0x01` instruction fetch that causes a refill **PMUEVENTn[2]** `0x02` instruction fetch that causes a TLB refill **PMUEVENTn[3]** `0x03` data read or write operation that causes a refill **PMUEVENTn[4]** `0x04` data read or write operation that causes a cache access **PMUEVENTn[5]** `0x05` data read or write operation that causes a TLB refill **PMUEVENTn[6]** `0x06` data read architecturally executed **PMUEVENTn[7]** `0x07` data write architecturally executed. **PMUEVENTn[8]** `0x68` number of instructions decoded **PMUEVENTn[9]** `0x09` exception taken **PMUEVENTn[10]** `0x0A` exception return architecturally executed **PMUEVENTn[11]** `0x0B` change to ContextID retired **PMUEVENTn[12]** `0x0C` software change of PC **PMUEVENTn[13]** `0x0D` immediate branch architecturally executed **PMUEVENTn[14]** `0x6E` number of predictable function returns **PMUEVENTn[15]** `0x0F` unaligned access architecturally executed **PMUEVENTn[16]** `0x10` branch mispredicted or not predicted. - `0x11` reserved **PMUEVENTn[17]** `0x12` branches or other change in program flow **PMUEVENTn[18]** `0x40` Java bytecode executed **PMUEVENTn[19]** `0x41` software Java bytecode executed **PMUEVENTn[20]** `0x42` Jazelle backward branches executed **PMUEVENTn[21]** `0x50` coherent linefill request that misses in other uniprocessors **PMUEVENTn[22]** `0x51` request for coherent linefill that hits in other uniprocessors **PMUEVENTn[23]** `0x60` instruction cache dependent stalls **PMUEVENTn[24]** `0x61` data cache dependent stalls **PMUEVENTn[25]** `0x62` TLB miss dependent stalls **PMUEVENTn[26]** `0x63` STREX passed **PMUEVENTn[27]** `0x64` STREX failed **PMUEVENTn[28]** `0x65` data eviction.
PMUIRQ[3:0]	O	Interrupt requests by system metrics, one per Cortex-A9 processor.

A.9 Parity error signals

Table A-27 shows parity error reporting signals. The number of sets of **PARITYFAIL** signals corresponds to the number of Cortex-A9 processors present in the design.

Table A-27 Error reporting signals

Signal	I/O	Description
PARITYFAILn[7:0]	O	Parity output pin from the RAM array for Cortex-A9 processor n. Indicates a parity fail. 0 no parity fail 1 parity fail Bit [7] BTAC parity error Bit [6] GHB parity error Bit [5] Instruction tag RAM parity error Bit [4] Instruction data RAM parity error Bit [3] Main TLB parity error Bit [2] D outer RAM parity error Bit [1] Data tag RAM parity error Bit [0] Data data RAM parity error.
PARITYSCU[3:0]	O	Parity output pin from the SCU tag RAMs. ORed output from each Cortex-A9 processor present in the design.
SCUEVABORT	O	An external abort has occurred during a coherency eviction.

A.10 MBIST interface

Table A-28 shows the MBIST interface signals.

Table A-28 MBIST interface signals

Signal	I/O	Description
MBISTADDR[10:0]	I	MBIST address.
MBISTARRAY[19:0]	I	MBIST arrays used for testing RAMs.
MBISTENABLE	I	Activates MBIST mode.
MBISTWRITEEN	I	Global write enable.
MBISTREADEN	I	Global read enable.

The size of some MBIST signals depends on whether the implementation has parity support or not. Table A-29 shows these signals with parity support implemented.

Table A-29 MBIST signals with parity support implemented

Signal	I/O	Description
MBISTBE[31:0]	I	MBIST write enable.
MBISTINDATA[71:0]	I	MBIST data in.
MBISTOUTDATA[287:0]	O	MBIST data out.

Table A-30 shows these signals without parity support implemented.

Table A-30 MBIST signals without parity support implemented

Signal	I/O	Description
MBISTBE[25:0]	I	MBIST write enable.
MBISTINDATA[63:0]	I	MBIST data in.
MBISTOUTDATA[255:0]	O	MBIST data out.

See *Cortex-A9 MBIST Controller TRM* for a description of MBIST.

A.11 Scan test signals

Table A-31 shows the scan test signals.

Table A-31 Scan test signals

Name	I/O	Description
SCANMODE	I	In scan test mode.
SE	I	Scan enable.

A.12 External Debug interface

The following sections describe the external debug interface signals:

- *Authentication interface*
- *APB interface signals* on page A-31
- *Cross trigger interface signals* on page A-32
- *Miscellaneous interface signals* on page A-32.

A.12.1 Authentication interface

Table A-32 shows the authentication interface signals.

Table A-32 Authentication interface signals

Signal	I/O	Description
DBGEN[3:0]	I	Invasive debug enable: 0 = not enabled 1 = enabled.
NIDEN[3:0]	I	Noninvasive debug enable: 0 = not enabled 1 = enabled.
SPIDEN[3:0]	I	Secure privileged invasive debug enable: 0 = not enabled 1 = enabled.
SPNIDEN[3:0]	I	Secure privileged noninvasive debug enable: 0 = not enabled 1 = enabled.

A.12.2 APB interface signals

Table A-33 shows the APB interface signals.

Table A-33 APB interface signals

Signal	I/O	Description
PADDRDBG[N:2]	I	Programming address. The width of N:2 depends on the configuration: [12:2] A uniprocessor or multiprocessor configuration with a single Cortex-A9 processor [13:2] A multiprocessor configuration with two Cortex-A9 processors [14:2] A multiprocessor configuration with three or four Cortex-A9 processors.
PADDRDBG31	I	APB address bus bit [31]: 0 = not an external debugger access 1 = external debugger access
PENABLEDBG	I	APB clock enable.
PSELDBG	I	Selects the external debug interface: 0 = debug registers not selected 1 = debug registers selected.
PWDATADBG	I	APB data enable.
PWRITEDBG	I	APB read and write signal.
PRDATADBG	O	APB slave ready. An APB slave can assert **PREADY** to extend a transfer.
PREADYDBG	O	Used to extend a transfer by inserting wait states
PSLVERRDBG	O	APB slave transfer error: 0 = no transfer error 1 = transfer error.

A.12.3 Cross trigger interface signals

Table A-34 shows the CTI signals

Table A-34 Cross trigger interface signals

Signal	I/O	Description
EDBGRQ[3:0]	I	External debug request: 0 = no external debug request 1 = external debug request. The processor treats the **EDBGRQ** input as level sensitive. The **EDBGRQ** input must be asserted until the processor asserts **DBGACK**.
DBGACK[3:0]	O	Debug acknowledge signal
DBGCPUDONE[3:0]	O	Debug acknowledge signal 0 = not enabled 1 = enabled.
DBGRESTART[3:0]	I	Causes the core to exit from Debug state. It must be held HIGH until **DBGRESTARTED** is deasserted. 0 = not enabled 1 = enabled.
DBGRESTARTED[3:0]	O	Used with **DBGRESTART** to move between Debug state and Normal state. 0 = not enabled 1 = enabled.

A.12.4 Miscellaneous interface signals

Table A-35 on page A-33 shows the miscellaneous interface signals.

Table A-35 Miscellaneous debug signals

Signal	I/O	Description
COMMRX[3:0]	O	Comms Channels Receive. Receive portion of Data Transfer Register full flag: 0 = empty 1 = full.
COMMTX[3:0]	O	Comms Channels Transmit. Transmit portion of Data Transfer Register full flag: 0 = empty 1 = full.
DBGNOPWRDWN[3:0]	O	Debugger has requested a Cortex-A9 processor is not powered down.
DBGSWENABLE[3:0]	I	When HIGH only the external debug agent can modify debug registers. 0 = not enabled. This is the default. 1 = Enabled. Access by the software through the extended cp14 interface is permitted. External cp14 and external debug accesses are permitted.
DBGROMADDR[31:12]	I	Specifies bits [31:12] of the ROM table physical address. If the address cannot be determined tie this signal off to zero.
DBGROMADDRV	I	Valid signal for **DBGROMADDR**. If the address cannot be determined tie this signal LOW.
DBGSELFADDR[31:12]	I	Specifies bits [31:12] of the two's complement signed offset from the ROM Table physical address to the physical address where the debug registers are memory-mapped. If the offset cannot be determined tie this signal off to zero.
DBGSELFADDRV	I	Valid signal for **DBGSELFADDR**. If the offset cannot be determined tie this signal LOW.

A.13 PTM interface signals

Table A-36 shows the PTM interface signals. There can be as many PTM interface signal buses as there are Cortex-A9 processors in the design.

Table A-36 PTM interface signals

Signal	I/O	Description
WPTFIFOEMPTYn	O	There are no speculative waypoints in the PTM interface FIFO.
WPTCOMMITn	O	Number of waypoints committed this cycle. It is valid to indicate a valid waypoint and commit it in the same cycle.
WPTCONTEXTIDn[31:0]	O	Context ID for the waypoint. This signal must be true regardless of the condition code of the waypoint.
WPTENABLE	I	Enable waypoint. When set, enables the Cortex-A9 processor to output waypoints.
WPTEXCEPTIONTYPEn	O	Exception type: b0001 = Halting Debug b0010 = Secure Monitor b0100 = Imprecise Data Abort b0101 = T2EE trap b1000 = Reset b1001 = UNDEF b1010 = SVC b1011 = Prefetch abort/Software Breakpoint b1100 = Precise data abort/software watchpoint b1101 = IRQ b1111 = FIQ.
WPTFLUSHn	O	Flush signal from core exception FIFO. All as yet uncommitted waypoints are flushed.
WPTLINKn	O	The waypoint is a branch and updates the link register. Only HIGH if **WPTTYPE[2:0]** is a direct branch or an indirect branch.
WPTnSECUREn	O	Instructions following the current waypoint are executed in Non-secure state. An instruction is in Non-secure state if the NS bit is set and the processor is not in secure monitor mode.

Table A-36 PTM interface signals (continued)

Signal	I/O	Description
WPTPCn [31:0]	O	Waypoint last executed address indicator. This is the base LR in the case of an exception. Must be 0 for a reset exception, when it must not be traced.Equal to 0 if the waypoint is reset exception.
WPTT32nT16n	O	Indicates the size of the last executed address when in Thumb state: 0 = 16-bit instruction 1 = 32-bit instruction. Must be set if previous **WPTTARGETTBIT** was zero (ARM state).
WPTTAKENn	O	The waypoint passed its condition codes. The address is still used, irrespective of the value of this signal. Must be set for all waypoints except branch.
WPTTARGETJBITn	O	J bit for waypoint destination. This signal is LOW if **WPTTRACEPROHIBITED** is asserted.
WPTTARGETPCn[31:0]	O	Waypoint target address. Bit [1] must be zero if T-bit is zero. Bit [0] must be zero if J-bit is zero. The Value is zero if **WPTTYPE** is either prohibit or debug.
WPTTARGETTBITn	O	T bit for waypoint destination This signal is LOW if **WPTTRACEPROHIBITED** is asserted.

Table A-36 PTM interface signals (continued)

Signal	I/O	Description
WPTTRACEPROHIBITEDn	O	Trace is prohibited for the current waypoint target. Indicates entry to prohibited region. No more waypoints are traced until trace can resume. Indication that PTM clocks can be stopped. This signal must be permanently asserted if **NIDEN** and **DBGEN** are both LOW, after the in-flight waypoints have exited the core. Only one **WPTVALID** cycle can be seen with **WPTTRACEPROHIBITED** set. Trace stops with this waypoint and the next waypoint seen is an Isync packet.
WPTTYPEn[2:0]	O	Waypoint Type. b000 = Direct Branch b001 = Indirect Branch b010 = Exception b011 = DMB b100 = Debug entry/Trace prohibited b101 = Debug exit (require addresses of first instruction) b110 = Invalid b111 = Invalid. Must only take valid states when **WPTVALID** is HIGH. Debug Entry must be followed by Debug Exit. **Note** Debug exit does not reflect the execution of an instruction.
WPTVALIDn	O	Waypoint is confirmed as valid.

Appendix B
Revisions

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

Table B-1 Issue A

Change	Location
First release	-

Table B-2 Differences between issue A and issue B

Change	Location
Clarify the relationship between the GIC (PL390) and the Cortex-A9 Interrupt Controller	Chapter 4 *Interrupt Controller*.
Parity error option added	Table 1-1 on page 1-5.
Clarify the role of the SCU with reference to data coherency and the non-support of instruction cache coherency	*About the SCU* on page 3-2.
Added information about exclusive accesses and address filtering	*Address filtering* on page 3-2.

Table B-2 Differences between issue A and issue B (continued)

Change	Location
SSAC description corrected	*SCU Secure Access Control Register (SSAC)* on page 3-12.
SSAC bit assignments corrected	Table 3-10 on page 3-13.
Change STI, Software Triggered Interrupt, to SGI, Software Generated Interrupt	Throughout Chapter 4 *Interrupt Controller*.
INTID descriptions extended and clarified	Throughout Chapter 4 *Interrupt Controller*.
Reset information added	*Timer and watchdog registers* on page 5-3.
AXI transaction IDs section extended	*AXI transaction IDs* on page 6-3.
AXI USER encodings section added	*AXI USER encodings* on page 6-5.
EVENTI information extended and **EVENTO** information added	*WFE/SEV synchronization* on page 6-9.
CLUSTERID[3:0] description corrected	*Configuration signals* on page A-5.
DBGEN[3:0] description added	Table A-32 on page A-30.

Glossary

This glossary describes some of the terms used in ARM manuals. Where terms can have several meanings, the meaning presented here is intended.

Abort

A mechanism that indicates to a core that the value associated with a memory access is invalid. An abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction or data memory. An abort is classified as either a Prefetch or Data Abort, and an internal or External Abort.

See also Data Abort, External Abort and Prefetch Abort.

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 details 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.

Advanced Peripheral Bus (APB)
A simpler bus protocol than AXI and AHB. It is designed for use with ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports. Connection to the main system bus is through a system-to-peripheral bus bridge that helps to reduce system power consumption.

AHB *See* Advanced High-performance Bus.

AHB Access Port (AHB-AP)
An optional component of the DAP that provides an AHB interface to a SoC.

AHB-AP *See* AHB Access Port.

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.

Advanced Trace Bus (ATB)
A bus used by trace devices to share CoreSight capture resources.

APB *See* Advanced Peripheral Bus.

Architecture The organization of hardware and/or software that characterizes a processor and its attached components, and enables devices with similar characteristics to be grouped together when describing their behavior, for example, Harvard architecture, instruction set architecture, ARMv6 architecture.

ARM instruction A word that specifies an operation for an ARM processor to perform. ARM instructions must be word-aligned.

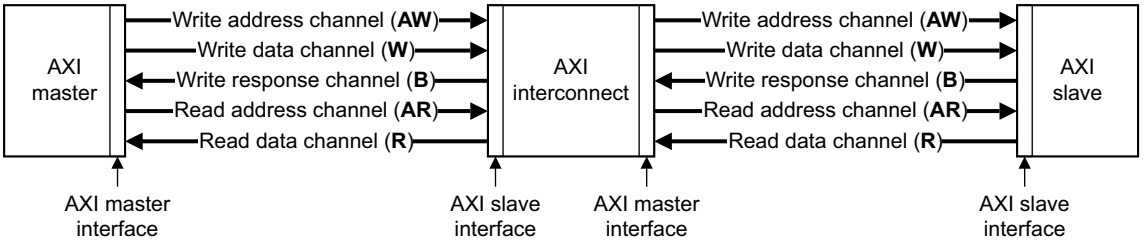
ARM state A processor that is executing ARM (32-bit) word-aligned instructions is operating in ARM state.

AXI *See* Advanced eXtensible Interface.

AXI channel order and interfaces

The block diagram shows:

- the order that AXI channel signals are described in
- 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. This is specified instead of write or read issuing capability for master interfaces that use a combined storage for active write and read transactions.

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. This is specified instead of write or read acceptance capability for slave interfaces that use a combined storage for active write and read transactions.

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.

Beat

Alternative word for an individual transfer within a burst. For example, an INCR4 burst comprises four beats.

See also Burst.

Block address

An address that comprises a tag, an index, and a word field. The tag bits identify the way that contains the matching cache entry for a cache hit. The index bits identify the set being addressed. The word field contains the word address that can be used to identify specific words, halfwords, or bytes within the cache entry.

See also Cache terminology diagram on the last page of this glossary.

Burst

A group of transfers to consecutive addresses. Because the addresses are consecutive, there is no requirement to supply an address for any of the transfers after the first one. This increases the speed at which the group of transfers can occur. Bursts over AHB buses are controlled using the **HBURST** signals to specify if transfers are single, four-beat, eight-beat, or 16-beat bursts, and to specify how the addresses are incremented.

See also Beat.

Byte An 8-bit data item.

Byte-invariant In a byte-invariant system, the address of each byte of memory remains unchanged when switching between little-endian and big-endian operation. When a data item larger than a byte is loaded from or stored to memory, the bytes making up that data item are arranged into the correct order depending on the endianness of the memory access. The ARM architecture supports byte-invariant systems in ARMv6 and later versions. When byte-invariant support is selected, unaligned halfword and word memory accesses are also supported. Multi-word accesses are expected to be word-aligned.

See also Word-invariant.

Byte lane strobe An AHB signal, **HBSTRB**, that is used for unaligned or mixed-endian data accesses to determine which byte lanes are active in a transfer. One bit of **HBSTRB** corresponds to eight bits of the data bus.

Cache A block of on-chip or off-chip fast access memory locations, situated between the processor and main memory, used for storing and retrieving copies of often used instructions and/or data. This is done to greatly reduce the average speed of memory accesses and so to increase processor performance.

See also Cache terminology diagram on the last page of this glossary.

Cache contention When the number of frequently-used memory cache lines that use a particular cache set exceeds the set-associativity of the cache. In this case, main memory activity increases and performance decreases.

Cache hit A memory access that can be processed at high speed because the instruction or data that it addresses is already held in the cache.

Cache line The basic unit of storage in a cache. It is always a power of two words in size (usually four or eight words), and is required to be aligned to a suitable memory boundary.

See also Cache terminology diagram on the last page of this glossary.

Cache line index The number associated with each cache line in a cache way. Within each cache way, the cache lines are numbered from 0 to (set associativity) -1.

See also Cache terminology diagram on the last page of this glossary.

Cache lockdown To fix a line in cache memory so that it cannot be overwritten. Cache lockdown enables critical instructions and/or data to be loaded into the cache so that the cache lines containing them are not subsequently reallocated. This ensures that all subsequent accesses to the instructions/data concerned are cache hits, and therefore complete as quickly as possible.

Cache miss A memory access that cannot be processed at high speed because the instruction/data it addresses is not in the cache and a main memory access is required.

Cache set A cache set is a group of cache lines (or blocks). A set contains all the ways that can be addressed with the same index. The number of cache sets is always a power of two.

See also Cache terminology diagram on the last page of this glossary.

Cache way A group of cache lines (or blocks). It is 2 to the power of the number of index bits in size.

See also Cache terminology diagram on the last page of this glossary.

Clean A cache line that has not been modified while it is in the cache is said to be clean. To clean a cache is to write dirty cache entries into main memory. If a cache line is clean, it is not written on a cache miss because the next level of memory contains the same data as the cache.

See also Dirty.

Coherency *See* Memory coherency.

Cold reset Also known as power-on reset. Starting the processor by turning power on. Turning power off and then back on again clears main memory and many internal settings. Some program failures can lock up the processor and require a cold reset to enable the system to be used again. In other cases, only a warm reset is required.

See also Warm reset.

Communications channel
The hardware used for communicating between the software running on the processor, and an external host, using the debug interface. When this communication is for debug purposes, it is called the Debug Comms Channel. In an ARMv6 compliant core, the communications channel includes the Data Transfer Register, some bits of the Data Status and Control Register, and the external debug interface controller, such as the DBGTAP controller in the case of the JTAG interface.

Condition field A four-bit field in an instruction that specifies a condition under which the instruction can execute.

Conditional execution
If the condition code flags indicate that the corresponding condition is true when the instruction starts executing, it executes normally. Otherwise, the instruction does nothing.

Context The environment that each process operates in for a multitasking operating system. In ARM processors, this is limited to mean the Physical Address range that it can access in memory and the associated memory access permissions.

See also Fast context switch.

Control bits The bottom eight bits of a Program Status Register (PSR). The control bits change when an exception arises and can be altered by software only when the processor is in a privileged mode.

Coprocessor A processor that supplements the main processor. It carries out additional functions that the main processor cannot perform. Usually used for floating-point math calculations, signal processing, or memory management.

Copy back *See* Write-back.

Core A core is that part of a processor that contains the ALU, the datapath, the general-purpose registers, the Program Counter, and the instruction decode and control circuitry.

Core module In the context of an ARM Integrator, a core module is an add-on development board that contains an ARM processor and local memory. Core modules can run standalone, or can be stacked onto Integrator motherboards.

Core reset *See* Warm reset.

CPSR *See* Current Program Status Register

Current Program Status Register (CPSR)
The register that holds the current operating processor status.

Data Abort An indication from a memory system to a core that it must halt execution of an attempted illegal memory access. A Data Abort is attempting to access invalid data memory.

See also Abort, External Abort, and Prefetch Abort.

Data cache A block of on-chip fast access memory locations, situated between the processor and main memory, used for storing and retrieving copies of often used data. This is done to greatly reduce the average speed of memory accesses and so to increase processor performance.

DBGTAP *See* Debug Test Access Port.

Debugger A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.

Dirty A cache line in a write-back cache that has been modified while it is in the cache is said to be dirty. A cache line is marked as dirty by setting the dirty bit. If a cache line is dirty, it must be written to memory on a cache miss because the next level of memory contains data that has not been updated. The process of writing dirty data to main memory is called cache cleaning.

See also Clean.

DNM *See* Do Not Modify.

Do Not Modify (DNM)
In Do Not Modify fields, the value must not be altered by software. DNM fields read as Unpredictable values, and must only be written with the same value read from the same field on the same processor. DNM fields are sometimes followed by RAZ or RAO in parentheses to show which way the bits must read for future compatibility, but programmers must not rely on this behavior.

Doubleword A 64-bit data item. The contents are taken as being an unsigned integer unless otherwise stated.

Doubleword-aligned
A data item having a memory address that is divisible by eight.

Exception A fault or error event that is considered serious enough to require that program execution is interrupted. Examples include attempting to perform an invalid memory access, external interrupts, and undefined instructions. When an exception occurs, normal program flow is interrupted and execution is resumed at the corresponding exception vector. This contains the first instruction of the interrupt handler to deal with the exception.

Exception service routine
See Interrupt handler.

Exception vector *See* Interrupt vector.

External Abort An indication from an external memory system to a core that it must halt execution of an attempted illegal memory access. An External Abort is caused by the external memory system as a result of attempting to access invalid memory.

See also Abort, Data Abort and Prefetch Abort.

Flat address mapping
A system of organizing memory in which each Physical Address contained within the memory space is the same as its corresponding Virtual Address.

Front of queue pointer
Pointer to the next entry to be written to in the write buffer.

Halfword A 16-bit data item.

Halt mode One of two mutually exclusive debug modes. In halt mode all processor execution halts when a breakpoint or watchpoint is encountered. All processor state, coprocessor state, memory and input/output locations can be examined and altered by the JTAG interface.

See also Monitor debug-mode.

High vectors Alternative locations for exception vectors. The high vector address range is near the top of the address space, rather than at the bottom.

Host A computer that provides data and other services to another computer. Especially, a computer providing debugging services to a target being debugged.

IEEE 754 standard *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std. 754-1985*. The standard that defines data types, correct operation, exception types and handling, and error bounds for floating-point systems. Most processors are built in compliance with the standard in either hardware or a combination of hardware and software.

IEM *See* Intelligent Energy Manager.

IGN *See* Ignore.

Ignore (IGN) Must ignore memory writes.

Illegal instruction An instruction that is architecturally Undefined.

IMB *See* Instruction Memory Barrier.

Implementation-defined
Means that the behavior is not architecturally defined, but must be defined and documented by individual implementations.

Implementation-specific
Means that 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.

Imprecise tracing A filtering configuration where instruction or data tracing can start or finish earlier or later than expected. Most cases cause tracing to start or finish later than expected.

For example, if **TraceEnable** is configured to use a counter so that tracing begins after the fourth write to a location in memory, the instruction that caused the fourth write is not traced, although subsequent instructions are. This is because the use of a counter in the **TraceEnable** configuration always results in imprecise tracing.

Index *See* Cache index.

Index register A register specified in some load or store instructions. The value of this register is used as an offset to be added to or subtracted from the base register value to form the virtual address, which is sent to memory. Some addressing modes optionally enable the index register value to be shifted prior to the addition or subtraction.

Instruction cache A block of on-chip fast access memory locations, situated between the processor and main memory, used for storing and retrieving copies of often used instructions. This is done to greatly reduce the average speed of memory accesses and so to increase processor performance.

Instruction cycle count
The number of cycles for which an instruction occupies the Execute stage of the pipeline.

Instruction Memory Barrier (IMB)
An operation to ensure that the prefetch buffer is flushed of all out-of-date instructions.

Intelligent Energy Manager (IEM)
A technology that enables dynamic voltage scaling and clock frequency variation to be used to reduce power consumption in a device.

Internal scan chain A series of registers connected together to form a path through a device, used during production testing to import test patterns into internal nodes of the device and export the resulting values.

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.

Invalidate To mark a cache line as being not valid by clearing the valid bit. This must be done whenever the line does not contain a valid cache entry. For example, after a cache flush all lines are invalid.

Joint Test Action Group (JTAG)
The name of the organization that developed standard IEEE 1149.1. This standard defines a boundary-scan architecture used for in-circuit testing of integrated circuit devices. It is commonly known by the initials JTAG.

JTAG *See* Joint Test Action Group.

Line *See* Cache line.

Load/store architecture
A processor architecture where data-processing operations only operate on register contents, not directly on memory contents.

Load Store Unit (LSU)
The part of a processor that handles load and store transfers.

LSU *See* Load Store Unit.

Macrocell A complex logic block with a defined interface and behavior. A typical VLSI system comprises several macrocells (such as a processor, an ETM, and a memory block) plus application-specific logic.

Memory bank One of two or more parallel divisions of interleaved memory, usually one word wide, that enable reads and writes of multiple words at a time, rather than single words. All memory banks are addressed simultaneously and a bank enable or chip select signal determines which of the banks is accessed for each transfer. Accesses to sequential word addresses cause accesses to sequential banks. This enables the delays associated with accessing a bank to occur during the access to its adjacent bank, speeding up memory transfers.

Memory coherency A memory is coherent if the value read by a data read or instruction fetch is the value that was most recently written to that location. Memory coherency is made difficult when there are multiple possible physical locations that are involved, such as a system that has main memory, a write buffer and a cache.

Memory Management Unit (MMU)
Hardware that controls caches and access permissions to blocks of memory, and translates virtual addresses to physical addresses.

Microprocessor *See* Processor.

Miss *See* Cache miss.

MMU *See* Memory Management Unit.

Modified Virtual Address (MVA)
A Virtual Address produced by the ARM processor can be changed by the current Process ID to provide a *Modified Virtual Address* (MVA) for the MMUs and caches.

See also Fast Context Switch Extension.

Monitor debug-mode
One of two mutually exclusive debug modes. In Monitor debug-mode the processor enables a software abort handler provided by the debug monitor or operating system debug task. When a breakpoint or watchpoint is encountered, this enables vital system interrupts to continue to be serviced while normal program execution is suspended.

See also Halt mode.

Multi-ICE A JTAG-based tool for debugging embedded systems.

MVA *See* Modified Virtual Address.

PA *See* Physical Address.

Penalty The number of cycles in which no useful Execute stage pipeline activity can occur because an instruction flow is different from that assumed or predicted.

Power-on reset *See* Cold reset.

Prefetching In pipelined processors, the process of fetching instructions from memory to fill up the pipeline before the preceding instructions have finished executing. Prefetching an instruction does not mean that the instruction must be executed.

Prefetch Abort An indication from a memory system to a core that it must halt execution of an attempted illegal memory access. A Prefetch Abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction memory.

See also Data Abort, External Abort and Abort.

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.

Physical Address (PA)

The MMU performs a translation on *Modified Virtual Addresses* (MVA) to produce the *Physical Address* (PA) which is given to AHB to perform an external access. The PA is also stored in the data cache to avoid the necessity for address translation when data is cast out of the cache.

See also Fast Context Switch Extension.

Read Reads are defined as memory operations that have the semantics of a load. That is, the ARM instructions LDM, LDRD, LDC, LDR, LDRT, LDRSH, LDRH, LDRSB, LDRB, LDRBT, LDREX, RFE, STREX, SWP, and SWPB, and the Thumb instructions LDM, LDR, LDRSH, LDRH, LDRSB, LDRB, and POP. Java instructions that are accelerated by hardware can cause a number of reads to occur, according to the state of the Java stack and the implementation of the Java hardware acceleration.

RealView ICE A system for debugging embedded processor cores using a JTAG interface.

Region A partition of instruction or data memory space.

Remapping Changing the address of physical memory or devices after the application has started executing. This is typically done to enable RAM to replace ROM when the initialization has been completed.

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.

Saved Program Status Register (SPSR)
The register that holds the CPSR of the task immediately before the exception occurred that caused the switch to the current mode.

SBO *See* Should Be One.

SBZ *See* Should Be Zero.

SBZP *See* Should Be Zero or Preserved.

Scan chain A 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.

SCREG The currently selected scan chain number in an ARM TAP controller.

Set *See* Cache set.

Set-associative cache
In a set-associative cache, lines can only be placed in the cache in locations that correspond to the modulo division of the memory address by the number of sets. If there are *n* ways in a cache, the cache is termed *n*-way set-associative. The set-associativity can be any number greater than or equal to 1 and is not restricted to being a power of two.

Should Be One (SBO)
Should be written as 1 (or all 1s for bit fields) by software. Writing a 0 produces Unpredictable results.

Should Be Zero (SBZ)
Should be written as 0 (or all 0s for bit fields) by software. Writing a 1 produces Unpredictable results.

Should Be Zero or Preserved (SBZP)
Should be written as 0 (or all 0s for bit fields) by software, or preserved by writing the same value back that has been previously read from the same field on the same processor.

SPSR *See* Saved Program Status Register

Synchronization primitive
The memory synchronization primitive instructions are those instructions that are used to ensure memory synchronization. That is, the LDREX, STREX, SWP, and SWPB instructions.

Tag The upper portion of a block address used to identify a cache line within a cache. The block address from the CPU is compared with each tag in a set in parallel to determine if the corresponding line is in the cache. If it is, it is said to be a cache hit and the line can be fetched from cache. If the block address does not correspond to any of the tags, it is said to be a cache miss and the line must be fetched from the next level of memory.

See also Cache terminology diagram on the last page of this glossary.

TAP *See* Test access port.

Test Access Port (TAP)
The collection of four mandatory and one optional terminals that form the input/output and control interface to a JTAG boundary-scan architecture. The mandatory terminals are **TDI**, **TDO**, **TMS**, and **TCK**. The optional terminal is **TRST**. This signal is required in ARM cores because it is used to reset the debug logic.

Thumb instruction A halfword that specifies an operation for an ARM processor in Thumb state to perform. Thumb instructions must be halfword-aligned.

Thumb state A processor that is executing Thumb (16-bit) halfword aligned instructions is operating in Thumb state.

TLB *See* Translation Look-aside Buffer.

Translation Lookaside Buffer (TLB)
A cache of recently used translation table entries that avoid the overhead of translation table walking on every memory access. Part of the Memory Management Unit.

Translation table A table, held in memory, that contains data that defines the properties of memory areas of various fixed sizes.

Translation table walk
The process of doing a full translation table lookup. It is performed automatically by hardware.

Trap An exceptional condition in a VFP coprocessor that has the respective exception enable bit set in the FPSCR register. The user trap handler is executed.

Undefined Indicates an instruction that generates an Undefined instruction trap. See the *ARM Architecture Reference Manual* for more details on ARM exceptions.

UNP *See* Unpredictable.

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.

VA *See* Virtual Address.

Virtual Address (VA)
: The MMU uses its translation tables to translate a Virtual Address into a Physical Address. The processor executes code at the Virtual Address, which might be located elsewhere in physical memory.

Warm reset
: Also known as a core reset. Initializes the majority of the processor excluding the debug controller and debug logic. This type of reset is useful if you are using the debugging features of a processor.

Watchpoint
: A watchpoint is a mechanism provided by debuggers to halt program execution when the data contained by a particular memory address is changed. Watchpoints are inserted by the programmer to enable inspection of register contents, memory locations, and variable values when memory is written to test that the program is operating correctly. Watchpoints are removed after the program is successfully tested. *See also* Breakpoint.

Way
: *See* Cache way.

WB
: *See* Write-back.

Word
: A 32-bit data item.

Word-invariant
: In a word-invariant system, the address of each byte of memory changes when switching between little-endian and big-endian operation, in such a way that the byte with address A in one endianness has address A EOR 3 in the other endianness. As a result, each aligned word of memory always consists of the same four bytes of memory in the same order, regardless of endianness. The change of endianness occurs because of the change to the byte addresses, not because the bytes are rearranged. The ARM architecture supports word-invariant systems in ARMv3 and later versions. When word-invariant support is selected, the behavior of load or store instructions that are given unaligned addresses is instruction-specific, and is in general not the expected behavior for an unaligned access. It is recommended that word-invariant systems use the endianness that produces the required byte addresses at all times, apart possibly from very early in their reset handlers before they have set up the endianness, and that this early part of the reset handler uses only aligned word memory accesses.

  See also Byte-invariant.

Write
: Writes are defined as operations that have the semantics of a store. That is, the ARM instructions SRS, STM, STRD, STC, STRT, STRH, STRB, STRBT, STREX, SWP, and SWPB, and the Thumb instructions STM, STR, STRH, STRB, and PUSH. Java instructions that are accelerated by hardware can cause a number of writes to occur, according to the state of the Java stack and the implementation of the Java hardware acceleration.

Write-back (WB)
: In a write-back cache, data is only written to main memory when it is forced out of the cache on line replacement following a cache miss. Otherwise, writes by the processor only update the cache. (Also known as copyback).

Write buffer A block of high-speed memory, arranged as a FIFO buffer, between the data cache and main memory, whose purpose is to optimize stores to main memory.

Write completion The memory system indicates to the processor that a write has been completed at a point in the transaction where the memory system is able to guarantee that the effect of the write is visible to all processors in the system. This is not the case if the write is associated with a memory synchronization primitive, or is to a Device or Strongly-ordered region. In these cases the memory system might only indicate completion of the write when the access has affected the state of the target, unless it is impossible to distinguish between having the effect of the write visible and having the state of target updated.

This stricter requirement for some types of memory ensures that any side-effects of the memory access can be guaranteed by the processor to have taken place. You can use this to prevent the starting of a subsequent operation in the program order until the side-effects are visible.

Write-through (WT) In a write-through cache, data is written to main memory at the same time as the cache is updated.

WT *See* Write-through.

Cache terminology diagram

The diagram illustrates the following cache terminology:

- block address
- cache line
- cache set
- cache way
- index
- tag.