KeyStone I Architecture
ARM CorePac

User Guide

Literature Number: SPRUHC4B
May 2012

Release History

Release	Date	Description/Comments
SPRUHC4B	May 2012	• Removed OCP-related expression from the AXI-to-VBUS Bridge chapter (Page 3-2) • Removed OCP-related expression from the ARM CorePac Block Diagram (Page 2-4)
SPRUHC4A	April 2012	• Removed narrow burst transfers and unaligned accesses chapters in the previous version (Page 3-3)
SPRUHC4	February 2012	Initial Release

Contents

List of Tables

List of Figures

Preface

About This Manual

This document describes the ARM CorePac in the KeyStone I Architecture but it does not contain details about the ARM core itself.

IMPORTANT NOTE—The information in this document should be used in conjunction with information in the device-specific Keystone Architecture data manual that applies to the part number of your device.

Notational Conventions

This document uses the following conventions:

- Commands and keywords are in **boldface** font.
- Arguments for which you supply values are in *italic* font.
- Terminal sessions and information the system displays are in `screen font`.
- Information you must enter is in **`boldface screen font`**.
- Elements in square brackets ([ ]) are optional.

Notes use the following conventions:

Note—Means reader take note. Notes contain helpful suggestions or references to material not covered in the publication.

The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

CAUTION—Indicates the possibility of service interruption if precautions are not taken.

WARNING—Indicates the possibility of damage to equipment if precautions are not taken.

Related Documentation from Texas Instruments

C66x CorePac User Guide	SPRUGW0
Interrupt Controller (INTC) for KeyStone Devices User Guide	SPRUGW4
C66x DSP Cache User Guide	SPRUGY8

Trademarks

C6000 ia a trademark of Texas Instruments Incorporated.

Chapter 1

Introduction

IMPORTANT NOTE—The information in this document should be used in conjunction with information in the device-specific Keystone Architecture data manual that applies to the part number of your device.

1.1 Overview

The ARM CorePac of the KeyStone devices handles transactions between the ARM core A8 processor, the L3 interconnect, and the ARM Interrupt Controller (ARM INTC). The ARM CorePac integrates the Cortex™-A8 processor with additional logic for protocol conversion, emulation, interrupt handling, and debug enhancements. The Cortex™-A8 (it is called the *ARM A8 Core* in this document) is an ARMv7-compatible, dual-issue, in-order execution engine with integrated L1 and L2 caches and a NEON SIMD media processing unit.

An interrupt controller is included in the ARM CorePac to handle host interrupt requests in the system.

The ARM CorePac includes CoreSight-compliant logic to allow the debug subsystem access to the ARM A8 Core debug and emulation resources, including the embedded trace macrocell.

The ARM CorePac has three functional clock domains including a high-frequency clock domain used by the ARM A8 Core. The high-frequency domain is isolated from the rest of the device by asynchronous bridges.

1.2 Terminology Used in This Document

The following acronyms and abbreviations appear in this user guide.

Term	Definition
AINTC	ARM Interrupt Controller
AMBA	Advanced Microcontroller Bus Architecture
APB	AMBA Advanced Peripheral Bus
ATB	AMBA Advanced Trace Bus
AXI	Advanced eXtensible Interface (Latest AMBA interface from ARM Ltd.)
CAM	Content Addressable Memory
CTI	Cross Trigger Interface
CoreSight	The infrastructure for monitoring, tracing, and debugging a complete system on chip
Cortex™-A8	ARM v7 R2P3 core. it is also called as “ARM A8 Core” in this document
ETB	Embedded Trace Buffer
ICECrusherCS	TI debug/emulation logic
INTC	Interrupt Controller
MPU	Memory Protection Unit
OCP	Open Core Protocol
OMAP3430	TI Wireless Business Unit Chip
PRCM	Power Reset Control Module
SIMD	Single Instruction Multiple Data
SECMON	Security Monitor
TLB	Translation Lookaside Buffer

Chapter 2

Overview of the KeyStone I ARM CorePac

This chapter describes the architecture of the ARM CorePac for KeyStone I devices and how the subsystem is structured.

2.1 Purpose of the ARM CorePac

The KeyStone I ARM CorePac contains a high-performance general-purpose ARM Cortex™-A8 microprocessor (ARM A8 Core).

The KeyStone I ARM CorePac architecture is defined around the ARM A8 Core. The logic of the ARM A8 Core performs three main functions; as long as these functions are performed correctly, program execution correctness will be maintained by the ARM A8 Core.

The three main functions of the ARM A8 Core are

- Memory operations

  Memory operations are managed by the AXI2VBUS Bridge (Chapter 3 on page 3-1) to move data in and out of the ARM A8 Core according to the AXI protocol and ordering rules and move data in and out of the ARM CorePac according to the OCP protocol and ordering rules.
- Exceptions and interrupts handling

  Exception and Interrupts functions are described in Chapter 4 on page 4-1. The interrupt controller ensures that all interrupts are delivered to the ARM A8 Core according to the model expected by software.
- Debug/emulations support

  The debug/emulation functions are described in Chapter 5 on page 5-1. Essentially, the support logic enables external debugger to access the debug and emulation resources in the ARM A8 Core.

Note—the ARM caching operation is only limited to DDR3 memory. ARM core must be configured to have all the rest of the memory spaces (MSMC memory, DSP CorePac L2 and AEMIF memory) as non-cacheable.

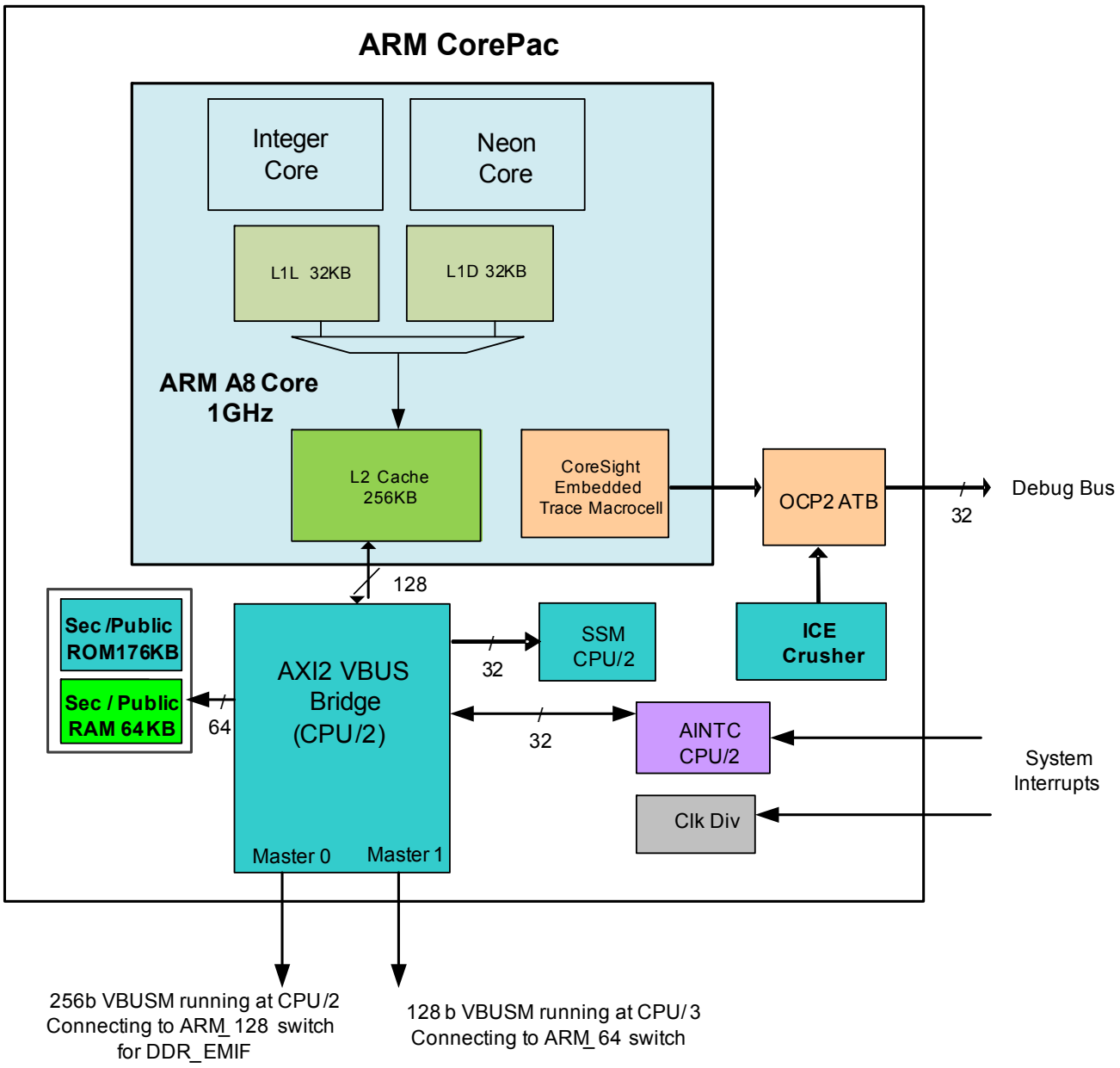
2.2 Features

- ARM Microprocessor
  - Cortex™-A8 revision R2P3 as ARM Core
  - ARM Architecture v7-A instruction set
  - 2-issue, in-order execution pipeline
  - 32KB/32KB I and D L1 cache
  - 256 KB Integrated L2 cache
  - Includes the Neon Media coprocessor which implements the Advanced SIMD media processing architecture
  - Includes the VFP coprocessor which implements the VFPv3 architecture and is fully compliant with IEEE 754 standard
  - The external interface uses the AXI protocol configured to 128-bit data width
  - Includes the Embedded Trace Macrocell (ETM) support for non-invasive debugging
  - Implements the ARMv7 debug with watch-point and breakpoint registers and 32-bit Advanced Peripheral Bus (APB) slave interface to CoreSight debug systems.
- AXI2VBUS Bridge (AXI — OCP — VBUS)
  - Support OCP 2.2
  - Single Request Multiple Data Protocol on two ports
  - Multiple targets, including three OCP ports (128-bit, 64-bit, and 32-bit)
- Interrupt Controller
  - Support up to 128 interrupt requests
  - Backward compatible with OMAP3430 software
- SSM (Secure State Machine)
  - The ARM A8 Core will be able to generate secure transactions within the system. The Secure State Machine (SSM) will exist but not be used because runtime security (Trustzone) on the ARM A8 Core will not be supported. Even though it is not used, SSM is connected to ARM CorePac to preserve correct operation of the ARM.
- Emulation/ Debug
  - On a separate voltage and power domain
  - Compatible with CoreSight architecture
- Features not supported in the ARM CorePac
  - Local reset
  - Run-time security
  - Boot mode other than boot through internal boot ROM
  - Multiple power domains (all modules are always on)
  - Slave ports (ARM CorePac does not have a slave port. The SoC masters cannot access the internal memory space of ARM CorePac)

2.3 Functional Block Diagram

The following block diagram shows the top level of ARM CorePac KeyStone I devices.

Figure 2-1 KeyStone I ARM CorePac Block Diagram



2.4 External Interfaces

The KeyStone I ARM CorePac has three external interfaces.

2.4.1 AMBA AXI Interface

The AXI bus interface is the main interface to the L3-interconnect and SDRAM. It performs L2 cache fills and non-cacheable accesses for both instructions and data. The data bus width is configured to 128-bit wide running at half the CPU clock speed. The AXI bus connects directly to the AXI2VBUS Bridge which decodes the address and forwards the requests to the internal peripherals (AINTC, Debug, Trace) or external interfaces. The AXI bus is a split-transaction bus that supports multiple outstanding requests with support for out-of-order responses.

2.4.2 AMBA APB Interface

The Cortex-A8 processor implements an APB slave interface that enables access to the ETM, CTI, and the debug registers. The APB interface is compatible with the CoreSight architecture which is the ARM architecture for multiprocessor trace and debug.

2.4.3 AMBA ATB Interface

The ARM A8 Core implements an ATB interface that outputs trace information for debugging. The ATB interface is compatible with the CoreSight architecture.

2.5 Reference

This user guide does not contain a detailed description and register map of ARM A8 Core and other related submodules except the ARM Interrupt Controller.

For more detailed information about the ARM processor Cortex-A8 and submodules, visit the ARM information website (see the ARM 926EJ-S Technical Reference Manual):
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0344k/index.html.

Chapter 3

AXI-to-VBUS Bridge

3.1 Overview

The basic function of the AXI2VBUS Bridge is to forward requests from the ARM A8 Core to any of the six targets and to route responses from the targets back to the ARM A8 Core. To accomplish this, the AXI2VBUS Bridge decodes the request address, performs protocol conversions including remapping from AXI to VBUSM, and forwards the request to the target based on the address map of each DSP. The AXI2VBUS Bridge must enforce the correct ordering when requests are sent to different targets. When the responses come back, it must prioritize them and return them to the ARM A8 Core to complete the transaction.

The AXI2VBUS Bridge must handle error conditions correctly. When enabled, it sends an interrupt signal to the ARM Interrupt Controller when a bus error is received. It also accepts a late signal from the SSM to prevent a request from going out when a security violation is detected.

The AXI2VBUS Bridge implements a time-out mechanism for requests that did not receive a response; this mechanism is enabled by a signal from the ICECrusherCS and is used in debug mode to prevent a system hang.

3.2 Hardware Interfaces

The AXI2VBUS Bridge has one internal Slave port. The slave port is connected to the AXI bus of the ARM A8 Core. The AXI port is 128 bits wide and is the main interface on the ARM A8 Core to the DDR memory controller. The AXI port supports multiple outstanding transactions and uses the Single-Request-Multiple-Data (SRMD) handshake protocol.

The AXI2VBUS Bridge has five master interfaces. Table 3-1 lists the master interfaces on the AXI2VBUS Bridge.

Table 3-1 AXI2VBUS Bridge Master Interfaces

Master Ports	Descriptions
Master Port 0	128-bit port connected directly to the DDR_EMIF to provide a low latency, high bandwidth connections to the DDR_EMIF
Master Port 1	64-bit port connected to the SCR to enable access to wide range of devices
INTC Configuration Interface	32-bit interface for access to INTC module (ARM CorePac internal)
SSM Configuration Interface	32-bit interface for access to SSM module (ARM CorePac internal)
ROM/RAM interface	Direct interface to the secure/public ROM and RAM(ARM CorePac internal)

The AXI2VBUS Bridge clock domain is synchronous to the ARM A8 Core but runs at the ARM A8 Core clock frequency.

3.3 Operation of the AXI2VBUS Bridge

3.3.1 Handling Read and Write Responses

The AXI2VBUS Bridge routes read and write responses from the VBUSM side to the read response channel and write response channel on the AXI side, respectively.

The AXI2VBUS Bridge routes requests from ARM A8 Core to five different targets. It is possible for multiple targets to return data in the same cycle; the AXI2VBUS Bridge must arbitrate these responses and forward the higher priority responses back to the AXI response bus. The arbitration is based on fixed priority and is outlined in Table 3-2.

Table 3-2 Priority of Response Arbitration

Priority	Master Port Response
1	ROM/RAM interface
2	Master Port 1
3	Master Port 1
4	SSM Configuration Interface
5	INTC Configuration Interface

The data buffers in ROM/RAM Interface and Master port 1 are needed for data width conversions between the interface and the AXI bus. These data buffers and the data buffers in Master port 0 are also used to handle narrow-burst-transfers. It is possible for multiple masters to return data in the same cycle; the AXI2VBUS Bridge picks the response based on the priority outlined in Table 3-2.

3.3.2 Peripheral Ports

Peripheral interfaces refer to the two Master ports to access the SSM and INTC modules. These are called peripheral interfaces because on the ARM11 (or earlier systems), there was a separate interface for connecting to SSM and INTC configuration space. This interface was called a peripheral interface and to access it the program must use a non-shared device memory attribute for the memory accesses.

The ARM A8 Core does not have a separate port; the peripheral port is merged into the main AXI bus. The AXI2VBUS Bridge needs only to decode the address range to determine if a request must be sent to the SSM or INTC configuration space.

3.3.3 Endian Conversion

ARM A8 Core operates in little endian mode only; when the device runs in big endian mode, the AXI2VBUS bridges are responsible for the endian conversion for the ARM. This conversion is automatically done by hardware based on the device pin *lendian* input.

For the AXI2VBUS bridge for the 128b port, all transactions are for the DDR_EMIF data space. The endianness conversion is controlled by the big_endian enable bit, which is an inverted input from the device pin *lendian*.

There are some transactions for the AXI2VBUS bridge for the 64b ARM port. especially going to the MMR space.that do not require endianness conversion. These transactions require an identical image regardless of the device endianness, assuming all the access to the MMR space is based on 32b.

There are two problematic cases which can cause data access problem when mixed endian mode is used (ARM: LE and DSP: BE)

Special Case 1: Shared memory access for Bit field operation Shared bit field memory access from ARM A8 Core and DSP when the device runs in big endian mode may cause trouble, because the AXI2VBUS bridge will do endianness conversion for ARM data. For example, if DSP(BE) write 0x0000ABCD to the shared memory, it will be read by ARM (LE) as 0xCDAB0000. This means bit field 0 ~ 7 for DSP will be bit field 24~ 31 for ARM.

Recommended: Swap data before writing to or after reading from shared memory. For example, ARM should write 0xCDAB0000 if 0x0000ABCD could be written to the shared memory. Same swapping could be done for reading.

Special Case 2: 16-bit flash memory access through EMIF16 ARM cannot correctly program 16-bit flash through EMIF16 when mixed endian mode is used (ARM: LE and DSP: BE) because 16-bit flash transfers 16-bit data at a time as a unit but AXI2VBUS bridge do byte level conversion (endian conversion). For example, if ARM tries to write 0xABCD to 16-bit flash, the AXI2vBUS bridge will convert it to 0xCDAB and this will be delivered to flash. But 16-bit flash cannot accept this because the expected command was 0xABCD and not 0xCDAB.(flash doesn't work right when it gets wrong command)

Recommended:

1. Do not write or read 16-bit flash from ARM when DSP is BE. If DSP is also LE, ARM can write and read without having any problem.
2. Use 8-bit flash instead of 16-bit flash.
3. Swap 16-bit data before writing to flash. If user wants to write 0xABCD to flash, SW should swap it to 0xCDAB before writing.

3.3.4 Word Swap

Word swap is used only when big endian mode is enabled; word swap has no impact when big endian is disabled (little endian). This word swap will be done automatically by ARM CorePac hardware. DSP data manual summarizes the transactions to/from the address range with word swap in KeyStone I device.

3.3.5 Address Swap

Address swap is applied only to the AXI2VBUS bridge 64b port. It swaps the address for the transactions for example, between address 0x3xxx_xxxx and 0x4xxx_xxxx. This means that when the bridge receives a transaction with address 0x3xxx_xxxx from the ARM, it sends out the same transaction with address 0x4xxx_xxxx.

When the bridge receives a transaction with address 0x4xxx_xxxx from ARM, it sends out the same transaction with address 0x3xxx_xxxx. For the returning data, the bridge also swaps the address 0x3xxx_xxxx and 0x4xxx_xxxx before the bridge returns the data to the ARM.

Chapter 4

ARM Interrupt Controller

4.1 Overview

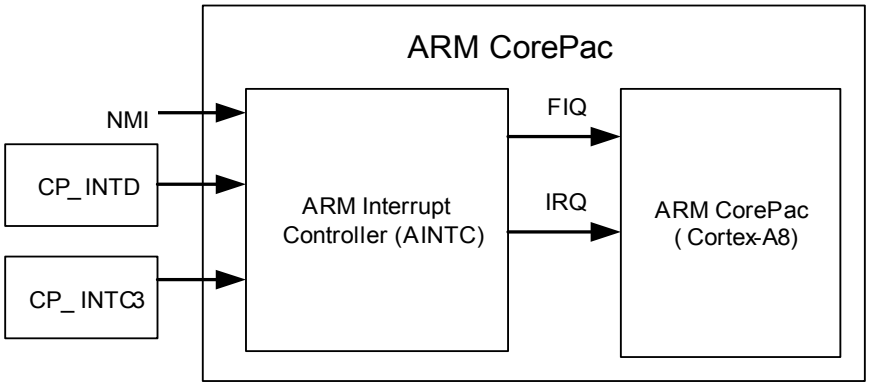
The host ARM Interrupt Controller (AINTC) prioritizes all service requests from the system peripherals and generates either an IRQ or FIQ interrupt to the host. The type of interrupt (IRQ or FIQ) and the priority of the interrupt inputs are programmable. The AINTC can process up to 128 requests that can be steered/prioritized as ARM A8 Core FIQ or IRQ interrupt requests.

The general features of the AINTC are as follows:

- level-sensitive interrupt inputs from CP_INTD and CP_INTC3
- Individual priority for each interrupt input
- Each interrupt can be steered to FIQ or IRQ
- Independent priority sorting for FIQ and IRQ

The following figure shows the internal interrupt scheme of ARM CorePac.

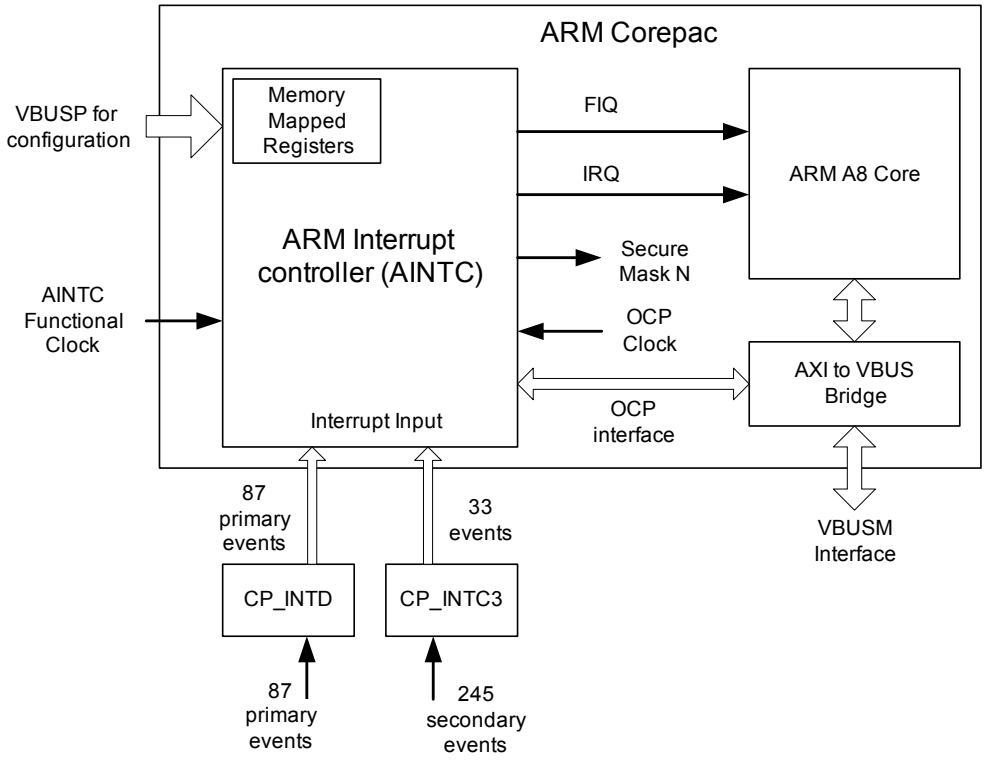
Figure 4-1 ARM Interrupt Controller (AINTC) Highlight



4.2 AINTC Integration

The AINTC is the interface between multiple incoming interrupts (from CP_INTC3 and CP_INTD) and the two interrupt inputs of the ARM A8 Core. The two interrupts inputs to ARM are FIQ and IRQ. Figure 4-2 shows the integration of the AINTC in the ARM CorePac.

Figure 4-2 ARM CorePac AINTC Integration



The AINTC is connected directly to the ARM A8 Core by an ARM peripheral port. Consequently, the AINTC is accessible and visible only by the ARM A8 Core.

4.2.1 Clocking and Reset

The Interrupt controller runs at half the rate of the ARM A8 Core functional clock. The interface clock, used for register access, runs at the rate of the interconnect bus clock (equal to the rate of the ARM CorePac reference clock).

Table 4-1 lists the AINTC resets.

Table 4-1 Resets to ARM Interrupt Controller

Type	Name	Source	Activation
Hardware	ARM top level reset	PRCM	Active Low
Software	Softreset	AINTC_SYSCONFIG[1]SOFTRESET bit	Active High

4.2.2 Interrupt Request Lines

Table 4-2 lists the incoming and outgoing interrupt lines of the ARM INTC.

Table 4-2 Interrupt Inputs and Outputs for the Interrupt Controller

Type	Number	Name	Mapping	Comments
Interrupt request inputs	Up to 128	See mapping table 5-3	See Table 4-3	Inputs to ARM INTC module, source from various modules
Interrupt request outputs	2	AINTC_FIQ, AINTC_IRQ	AINTC_FIQ, AINTC_IRQ	FIQ is fast interrupt to ARM and IRQ is normal input to ARM

The CPU interrupts on the KeyStone I ARM A8 Core are configured through the enhanced interrupt selector within ARM. The AINTC allow up to 128 system events to be routed to any of the two ARM interrupt inputs within ARM A8 Core. The following table shows the mapping of system events to the interrupt controller inputs for ARM. Event numbers 0 through 6 and 10 correspond to the default interrupt mapping of the device. The remaining events must be mapped using software. ARM can independently enable or disable any of the system events available to it through software.

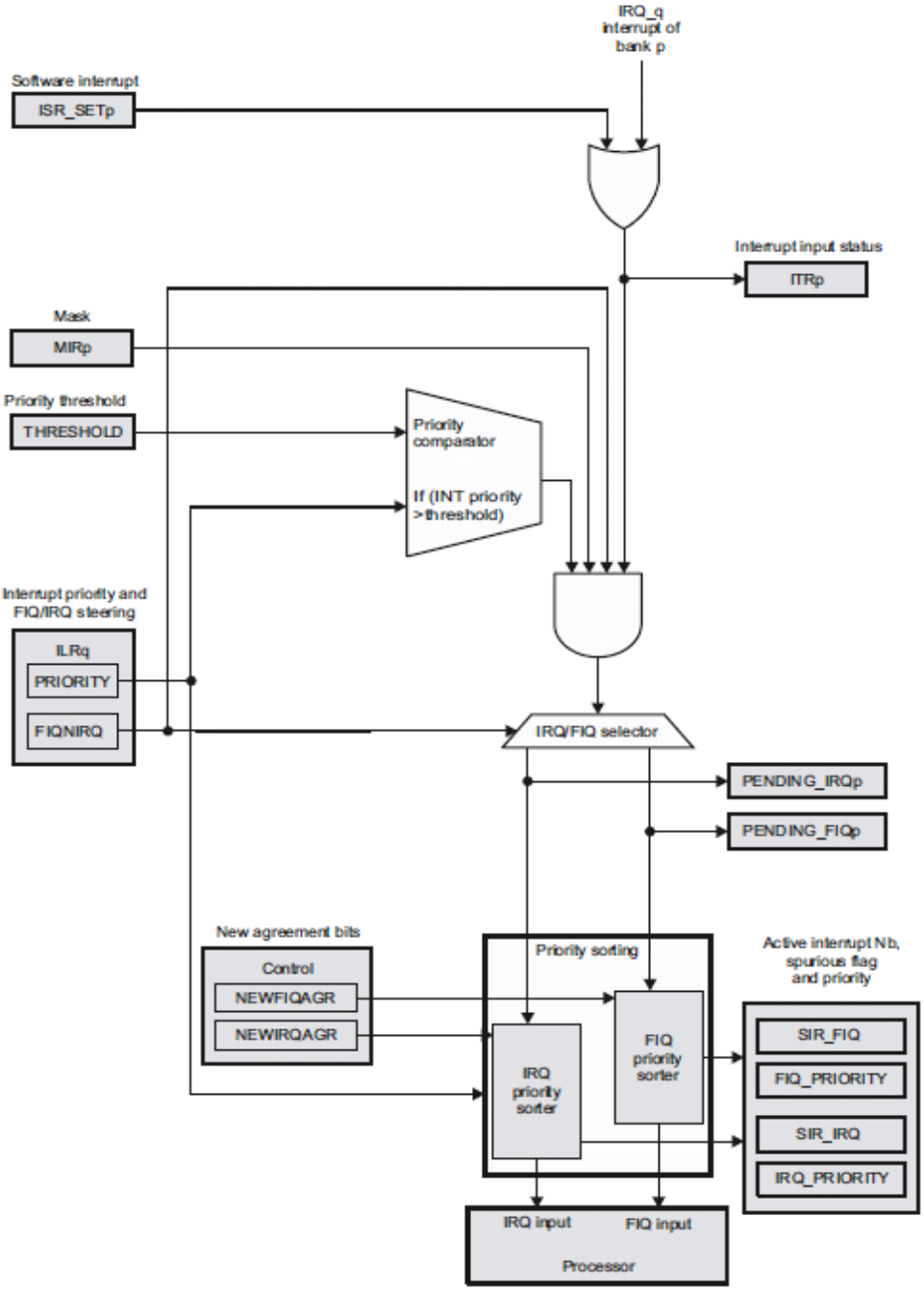
AINTC interrupt inputs are from CP_INTC3 and INTD (CP_INTD). Nearly all events from IPs are pulse-based interrupts and need to be routed to ARM as primary inputs. Since ARM requires level based interrupts, they must be converted to level interrupts by INTD before reaching ARM. by default, the polarity of the INTD input is active high and pulse based type. level based type inputs are bypassed.

Note—User can get detail about CP_INTC3 and CP_INTD interrupt input, output information from DSP Data manual.

4.3 Interrupt Controller Functional Description

The interrupt controller processes incoming interrupts by masking and priority sorting to produce the interrupt signals for the processor to which it is attached. Figure 4-3 shows the top-level view of interrupt processing.

Figure 4-3 Top-Level Block Diagram



4.3.1 Interrupt Processing

4.3.1.1 Input Selection

The AINTC supports only level-sensitive incoming interrupt detection. A peripheral asserting an interrupt maintains it until software has handled the interrupt and instructed the peripheral to deassert the interrupt.

A software interrupt is generated if the corresponding bit in the AINTC_ISR_SETn register is set (register bank number: n = [0,1,2,3] for the AINTC, 128 incoming interrupt lines are supported).

The software interrupt clears when the corresponding bit in the AINTC_ISR_CLEARn register is written. Typical use of this feature is software debugging.

4.3.1.2 Masking

4.3.1.2.1 Individual Masking

Detection of interrupts on each incoming interrupt line can be enabled or disabled independently by the AINTC_MIRn interrupt mask register. In response to an unmasked incoming interrupt, the AINTC can generate one of two types of interrupt requests to the processor:

- IRQ: low-priority interrupt request
- FIQ: fast interrupt request

The type of interrupt request is determined by the AINTC_ILRm[0] FIQNIRQ bit (m= [0,127]). The current incoming interrupt status before masking is readable from the AINTC_ITRn register. After masking and IRQ/FIQ selection, and before priority sorting is done, the interrupt status is readable from the following registers:

- AINTC_PENDING_IRQn
- AINTC_PENDING_FIQn

4.3.1.2.2 Priority Masking

To enable faster processing of high-priority interrupts, a programmable priority masking threshold is provided (the AINTC_THRESHOLD[7:0] PRIORITYTHRESHOLD field). This priority threshold allows preemption by higher priority interrupts; all interrupts of lower or equal priority than the threshold are masked. However, priority 0 can never be masked by this threshold; a priority threshold of 0 is treated the same way as priority 1. PRIORITY and PRIORITYTHRESHOLD fields values can be set between 0x0 and 0x7F; 0x0 is the highest priority and 0x7F is the lowest priority. When priority masking is not necessary, a priority threshold value of 0xFF disables the priority threshold mechanism. This value is also the reset default for backward compatibility with previous versions of the AINTC.

4.3.1.2.3 Priority Sorting

A priority level (0 being the highest) is assigned to each incoming interrupt line. Both the priority level and the interrupt request type are configured by the AINTC_ILRm register. If more than one incoming interrupt with the same priority level and interrupt request type occur simultaneously, the highest-numbered interrupt is serviced first. When one or more unmasked incoming interrupts are detected, the AINTC separates

between IRQ and FIQ using the corresponding AINTC_ILRm[0] FIQNIRQ bit. The result is placed in AINTC_PENDING_IRQn or AINTC_PENDING_FIQn. If no other interrupts are currently being processed, AINTC asserts IRQ/FIQ and starts the priority computation.

Priority sorting for IRQ and FIQ can execute in parallel. Each IRQ/FIQ priority sorter determines the highest priority interrupt number. Each priority number is placed in the corresponding AINTC_SIR_IRQ[6:0] ACTIVEIRQ field or AINTC_SIR_FIQ[6:0] ACTIVEFIQ field. The value is preserved until the corresponding AINTC_CONTROL NEWIRQAGR or NEWFIQAGR bit is set. After the interrupting peripheral device has been serviced and the incoming interrupt deasserted, the user must write to the appropriate NEWIRQAGR or NEWFIQAGR bit to indicate to the INTC the interrupt has been handled. If there are any pending unmasked incoming interrupts for this interrupt request type, the AINTC restarts the appropriate priority sorter; otherwise, the IRQ or FIQ interrupt line is deasserted.

4.3.2 Register Protection

If the AINTC_PROTECTION[0] PROTECTION bit is set, access to the AINTC registers is restricted to the supervisor mode. Access to the AINTC_PROTECTION register is always restricted to privileged mode.

4.3.3 Module Power Saving

The AINTC provides an auto-idle function in its three clock domains:

- Interface clock
- Functional clock
- Synchronizer clock

The interface clock auto-idle power-saving mode is enabled if the AINTC_SYSCONFIG[0] AUTOIDLE bit is set to 1. When this mode is enabled and there is no activity on the bus interface, the interface clock is disabled internally to the module, thereby reducing power consumption. When there is new activity on the bus interface, the interface clock restarts without a latency penalty. After reset, this mode is disabled by default.

The functional clock auto-idle power-saving mode is enabled if the AINTC_IDLE[0] FUNCIDLE bit is set to 0. When this mode is enabled and there is no active interrupt (IRQ or FIQ interrupt being processed or generated) or no pending incoming interrupt, the functional clock is disabled internally to the module, thereby reducing power consumption. When a new unmasked incoming interrupt is detected, the functional clock restarts and the AINTC processes the interrupt. If this mode is disabled, the interrupt latency is reduced by one cycle. After reset, this mode is enabled by default.

The synchronizer clock allows external asynchronous interrupts to be resynchronized before they are masked. The synchronizer input clock has an auto-idle power-saving mode enabled if the AINTC_IDLE[1] TURBO bit is set to 1. If the auto-idle mode is enabled, the standby power is reduced, but the IRQ or FIQ interrupt latency increases from four to six functional clock cycles. This feature can be enabled dynamically according to the requirements of the device. After reset, this mode is disabled by default.

4.3.4 Error Handling

The following accesses will cause errors:

- Privilege violation (attempt to access PROTECTION register in user mode or any register in user mode if Protection bit is set)
- Unsupported commands

The following will NOT cause an error response:

- Access to a non-decoded address
- Write to a read-only register

4.3.5 Interrupt Latency

The IRQ/FIQ interrupt generation takes four AINTC functional clock cycles (plus or minus one cycle) if the AINTC_IDLE[1] TURBO bit is set to 0. If the TURBO bit is set to 1, the interrupt generation takes six cycles, but power consumption is reduced while waiting for an interrupt. These latencies can be reduced by one cycle by disabling functional clock auto-idle (AINTC_IDLE[0] FUNCIDLE bit set to 1), but power consumption is increased, so the benefit is minimal.

To minimize interrupt latency when an unmasked interrupt occurs, the IRQ or FIQ interrupt is generated before priority sorting is completed. Priority sorting takes 10 functional clock cycles, which is less than the minimum number of cycles required for the ARM to switch to the interrupt context after reception of the IRQ or FIQ event.

Any read of the AINTC_SIR_IRQ or AINTC_SIR_FIQ register during the priority sorting process stalls until priority sorting is complete and the relevant register is updated. However, the delay between the interrupt request being generated and the interrupt service routine being executed is such that priority sorting always completes before the AINTC_SIR_IRQ or AINTC_SIR_FIQ register is read.

4.4 Interrupt Basic Programming Model

4.4.1 Initialization Sequence

1. Program the AINTC_SYSCONFIG register: If necessary, enable the interface clock autogating by setting the AUTOIDLE bit.
2. Program the AINTC_IDLE register: If necessary, disable functional clock autogating or enable synchronizer autogating by setting the FUNCIDLE bit or TURBO bit accordingly.
3. Program the AINTC_ILRm register for each interrupt line: Assign a priority level and set the FIQNIRQ bit for an FIQ interrupt (by default, interrupts are mapped to IRQ and priority is 0x0 [highest]).
4. Program the AINTC_MIRn register: Enable interrupts (by default, all interrupt lines are masked).
5. Program the INTD Enable Set register: Enable interrupts from INTD (See Interrupt Controller for Keystone devices User's guide to get information about programming CP_INTC3)

Note—To program the AINTC_MIRn register, the AINTC_MIR_SETn and AINTC_MIR_CLEARn registers are provided to facilitate the masking, even if it is possible for backward-compatibility to write directly to the AINTC_MIRn register.

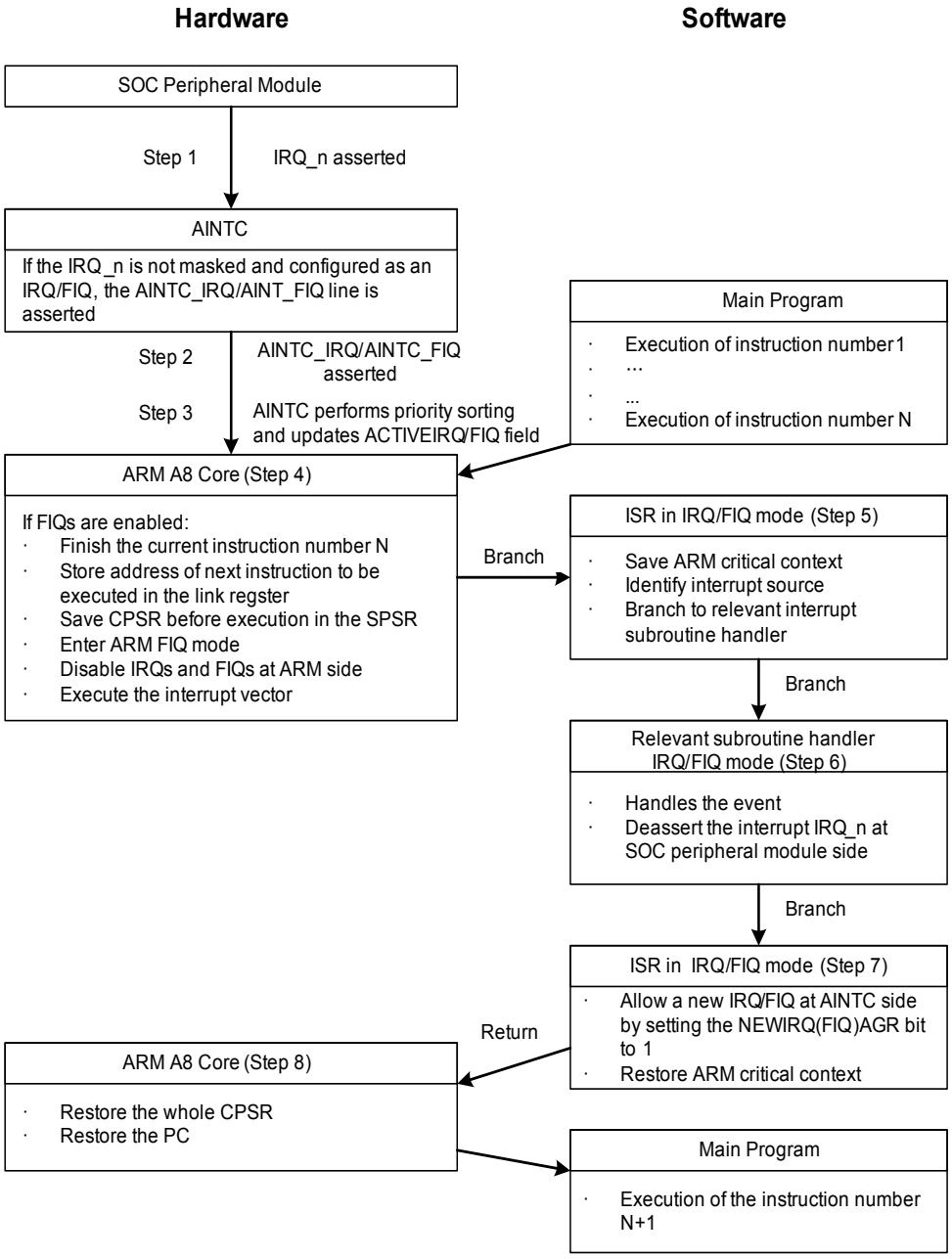
4.4.2 AINTC Processing Sequence

After the AINTC_MIRn and AINTC_ILRm registers are configured to enable and assign priorities to incoming interrupts, the interrupt is processed as described in the following subsections. IRQ and FIQ processing sequences are similar.

1. One or more unmasked incoming interrupts (IRQ_n signals) are received and IRQ or FIQ outputs (AINTC_IRQ/FIQ) are not currently asserted.
2. If the AINTC_ILRm[0] FIQNIRQ bit is set to 0, the AINTC_IRQ output signal is generated. If the FIQNIRQ bit is set to 1, the AINTC_FIQ output signal is generated.
3. The AINTC performs the priority sorting and updates the AINTC_SIR_IRQ[6:0] ACTIVEIRQ /AINTC_SIR_FIQ[6:0] ACTIVEFIQ field with the current interrupt number.
4. During priority sorting, if the IRQ/FIQ is enabled at the host processor side, the host processor automatically saves the current context and executes the ISR as follows:
5. The ISR saves the remaining context, identifies the interrupt source by reading the ACTIVEIRQ/ACTIVEFIQ field, and jumps to the relevant subroutine handler.
6. The subroutine handler executes code specific to the peripheral generating the interrupt by handling the event and deasserting the interrupt condition at the peripheral side or INTD.
7. After the return of the subroutine, the ISR sets the NEWIRQAGR/NEWFIQAGR bit to enable the processing of subsequent pending IRQs/FIQs and to restore ARM context in the following code. Because the writes are posted on an Interconnect bus, to be sure that the preceding writes are done before enabling IRQs/FIQs, a Data Synchronization Barrier is used. This operation ensure that the IRQ/FIQ line is de-asserted before IRQ/FIQ enabling. After that, the AINTC processes any other pending interrupts or deasserts the AINTC_IRQ/AINTC_FIQ signal if there is no interrupt.

Figure 4-4 shows the IRQ/FIQ processing sequence from the originating device peripheral module to the main program interruption.

Figure 4-4 IRQ/FIQ Processing Sequence



The priority sorting mechanism is frozen during an interrupt processing sequence. If an interrupt condition occurs during this time, the interrupt is not lost. It is sorted when the NEWIRQAGR/NEWFIQAGR bit is set (priority sorting is reactivated).

4.4.3 AINTC Preemptive Processing Sequence

Preemptive interrupts, also called nested interrupts, can reduce the latencies for higher priority interrupts. A preemptive ISR can be suspended by a higher priority interrupt. Thus, the higher priority interrupt can be served immediately. Nested interrupts must be used carefully to avoid using corrupted data. Programmers must save corruptible registers and enable IRQ or FIQ at ARM side. IRQ and FIQ processing sequences are similar.

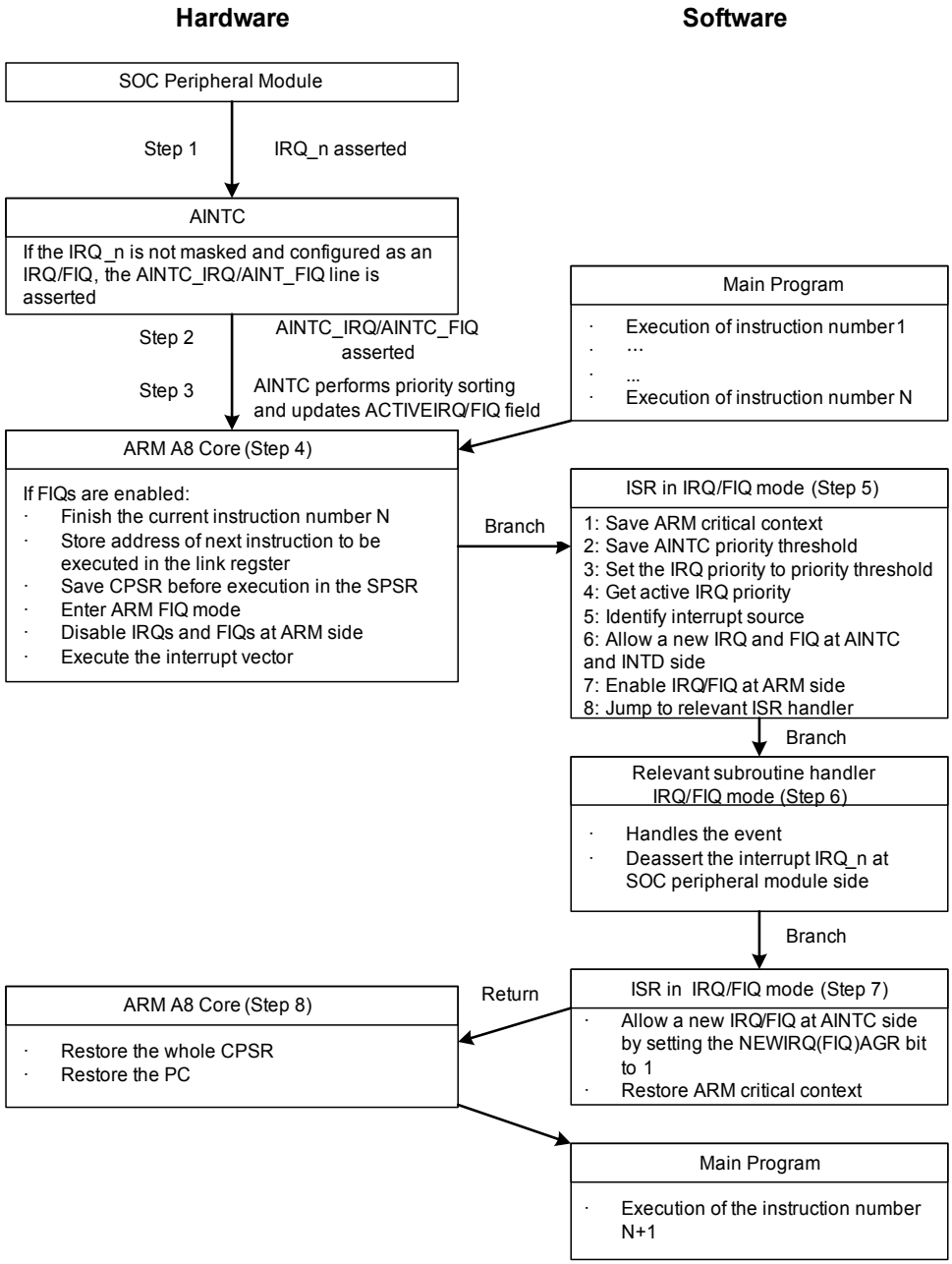
To enable IRQ/FIQ preemption by higher priority IRQs/FIQs, programers can follow this procedure to write the ISR.

At the beginning of an IRQ/FIQ ISR, do the following:

1. Save the ARM critical context registers.
2. Save the AINTC_THRESHOLD PRIORITYTHRESHOLD field before modifying it.
3. Read the active interrupt priority in the AINTC_IRQ_PRIORITY IRQPRIORITY / AINTC_FIQ_PRIORITY FIQPRIORITY field and write it to the PRIORITYTHRESHOLD(1) field.
4. Read the active interrupt number in the AINTC_SIR_IRQ[6:0] ACTIVEIRQ / AINTC_SIR_FIQ[6:0] ACTIVEFIQ field to identify the interrupt source.
5. Write 1 to the appropriate AINTC_CONTROL NEWIRQAGR and (2) NEWFIQAGR bit while an interrupt is still processing to allow only higher priority interrupts to preempt.
6. Because the writes are posted on an Interconnect bus, to be sure that the preceding writes are done before enabling IRQs/FIQs, a Data Synchronization Barrier is used. This operation ensure that the IRQ line is de-asserted before IRQ/FIQ enabling.
7. Enable IRQ/FIQ at ARM side.
8. Jump to the relevant subroutine handler.

Figure 4-5 shows the nested IRQ/FIQ processing sequence from the originating device peripheral module to the main program interruption.

Figure 4-5 Nested IRQ/FIQ Processing Sequence



4.4.4 Interrupt Preemption

To enable preemption by higher priority interrupts, the ISR should read the active interrupt priority and write it to the priority threshold register. Writing a *1* to the appropriate NEW_IRQ_AGR or NEW_FIQ_AGR bits of the CONTROL register while still processing the interrupt will now allow only higher priority interrupts to preempt.

For each level of preemption, the programmer must save the threshold value before modifying it and restore it at the end of that ISR level.

The priority threshold mechanism is enabled automatically when writing a priority in the range of 0x00 to 0x7F as will be read from the IRQ_PRIORITY and FIQ_PRIORITY registers. Writing a value of 0xFF (reset default) disables the priority threshold mechanism.

When the hardware priority threshold is in use, the priorities of interrupts selected as FIQ or IRQ become linked, otherwise they are independent. When linked, it is required that all FIQ priorities be set higher than all IRQ priorities to maintain the relative priority of FIQ over IRQ.

When handling FIQs using the priority threshold mechanism, it is required to write both the New FIQ Agreement and New IRQ Agreement bits at the same time to cover the case that the new priority threshold is applied while an IRQ sorting is in progress. This IRQ will not have been seen by the ARM as it will have been masked on entry to the FIQ ISR.

However, the source of the IRQ will remain active and it will be finally processed when the priority threshold falls to a low enough priority. The precaution of writing to New FIQ Agreement (as well as New IRQ Agreement) is not required during an IRQ ISR as FIQ sorting will not be affected (provided all FIQ priorities are higher than all IRQ priorities).

4.4.5 AINTC Spurious Interrupt Handling

The spurious flag indicates whether the result of the sorting (a window of 10 AINTC functional clock cycles after the interrupt assertion) is invalid. Sorting is invalid if:

- The interrupt that triggered the sorting is no longer active during the sorting.
- A change in the mask has affected the result during the sorting time.

As a result, the values in the AINTC_MIRn, AINTC_ILRm, or AINTC_MIR_SETn registers must not be changed while the corresponding interrupt is asserted. Only the active interrupt input that triggered the sort can be masked before it turn on the sort. If these registers are changed within the 10-cycle window after the interrupt assertion, the resulting values of the following registers become invalid:

- AINTC_SIR_IRQ
- AINTC_SIR_FIQ
- AINTC_IRQ_PRIORITY
- AINTC_FIQ_PRIORITY

This condition is detected for both IRQ and FIQ, and the invalid status is flagged across the SPURIOUSIRQFLAG (see the first note below) and SPURIOUSFIQFLAG (see the second note below) bit fields in the SIR and PRIORITY registers. A *0* indicates valid and a *1* indicates invalid interrupt number and priority. The invalid indication can be tested in software as a false register value.

Note—The AINTC_SIR_IRQ[31:7] SPURIOUSIRQFLAG bit field is a copy of the AINTC_IRQ_PRIORITY[31:7] SPURIOUSIRQFLAG bit field.

Note—The AINTC_SIR_FIQ[31:7] SPURIOUSFIQFLAG bit field is a copy of the AINTC_FIQ_PRIORITY[31:7] SPURIOUSFIQFLAG bit field.

4.5 Interrupt Controller Registers

The following sections describe the registers required to configure the ARM CorePac Interrupt Controller. Each register from AINTC_ITRn to AINTC_PENDING_FIQn contains 32 bits, one bit for each interrupt.

Table 4-3 lists the register offsets for these registers.

Table 4-3 Interrupt Controller Register Address Map

Register Name	Address Offset[1]
AINTC_REVISION	0x00
AINTC_SYSCONFIG	0x10
AINTC_SYSSTATUS	0x14
AINTC_SIR_IRQ	0x40
AINTC_SIR_FIQ	0x44
AINTC_CONTROL	0x48
AINTC_PROTECTION	0x4c
AINTC_IDLE	0x50
AINTC_IRQ_PRIORITY	0x60
AINTC_FIQ_PRIORITY	0x64
AINTC_THRESHOLD	0x68
AINTC_ITRn	0x80 + (0x20 * n)
AINTC_MIRn	0x84 + (0x20 * n)
AINTC_MIR_CLEARn	0x88 + (0x20 * n)
AINTC_MIR_SETn	0x8c + (0x20 * n)
AINTC_ISR_SETn	0x90 + (0x20 * n)
AINTC_ISR_CLEARn	0x94 + (0x20 * n)
AINTC_PENDING_IRQn	0x98 + (0x20 * n)
AINTC_PENDING_FIQn	0x9c + (0x20 * n)
AINTC_ILRm	0x100 + (0x4 * m)
End of Table 4-3	

1. n = 0 to 3, m = 0 to 127

4.5.1 AINTC_REVISION Register

This register contains the IP revision code.

Figure 4-6 AINTC_REVISION Register

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset; -t = TI internal data

Table 4-4 AINTC_REVISION Register Field Definitions

Bit	Field	Description
31-8	Reserved	Read returns reset value
7-4	MAJOR	Major revision
3-0	MINOR	Minor revision
End of Table 4-4		

4.5.2 AINTC_SYSCONFIG Register

This register controls various parameters of the module interface.

Figure 4-7 AINTC_SYSCONFIG Register

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 4-5 AINTC_SYSCONFIG Register Field Definitions

Bit	Field	Description
31-2	Reserved	Read returns reset value
1	SOFTRESET	Software reset. Set this bit to trigger a module reset. The bit is automatically reset by the hardware. Read returns 0. Write 0x0: No functional effect. Write 0x1: The module is reset.
0	AUTOIDLE	Internal interface clock gating strategy 0x0: Interface clock is free-running. 0x1: Automatic interface clock gating strategy is applied, based on the interface bus activity.
End of Table 4-5		

4.5.3 AINTC_SYSSTATUS Register

This register provides status information about the module.

Figure 4-8 AINTC_SYSSTATUS Register

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

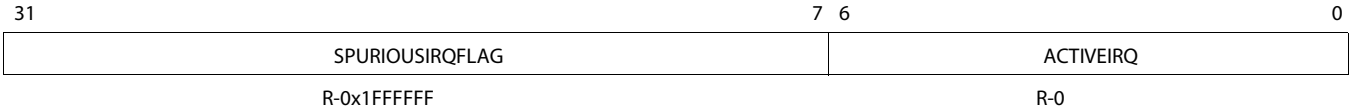
Table 4-6 AINTC_SYSSTATUS Register Field Definitions

Bit	Field	Description
31-1	Reserved	Read returns reset value
0	RESETDONE	Internal reset monitoring. Read 0x0: Internal module reset is ongoing. Read 0x1: Reset complete
End of Table 4-6		

4.5.4 AINTC_SIR_IRQ Register

This register supplies the currently active IRQ interrupt number.

Figure 4-9 AINTC_SIR_IRQ Register



Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 4-7 AINTC_SIR_IRQ Register Field Definitions

Bit	Field	Description
31-7	SPURIOUSIRQFLAG	Spurious IRQ flag
6-0	ACTIVEIRQ	Active IRQ number
End of Table 4-7		

4.5.5 AINTC_SIR_FIQ Register

This register supplies the currently active FIQ interrupt number.

Figure 4-10 AINTC_SIR_FIQ Register

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 4-8 AINTC_SIR_FIQ Register Field Definitions

Bit	Field	Description
31-7	SPURIOUSFIQFLAG	Spurious FIQ flag
6-0	ACTIVEFIQ	Active FIQ number
End of Table 4-8		

4.5.6 AINTC_CONTROL Register

This register contains the new interrupt agreement bits.

Figure 4-11 AINTC_CONTROL Register

31	2	1	0
Reserved		NEWFIQAGR	NEWIRQAGR
R-0		W-0	W-0

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 4-9 AINTC_CONTROL Register Field Definitions

Bit	Field	Description
31-2	Reserved	Read returns reset value
1	NEWFIQAGR	Reset FIQ output and enable new FIQ generation. Write 0x0: No functional effect Write 0x1: Reset FIQ output and enable new FIQ generation
0	NEWIRQAGR	New IRQ generation Write 0x0: No functional effect Write 0x1: Reset IRQ output and enable new IRQ generation.
End of Table 4-9		

4.5.7 AINTC_PROTECTION Register

This register controls protection of the other registers. It can be accessed only in supervisor mode, regardless of the current value of the protection bit.

Figure 4-12 AINTC_PROTECTION Register

31	1	0
Reserved		Protection
R-0		R/W-0

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 4-10 AINTC_PROTECTION Register Field Definitions

Bit	Field	Description
31-1	Reserved	Read returns reset value
0	Protection	Protection mode 0x0: Protection mode is disabled (default). 0x1: Protection mode is enabled. When enabled, all the AINTC registers are accessible only in privileged mode
End of Table 4-10		

4.5.8 AINTC_IDLE Register

This register controls the functional clock auto-idle and the synchronizer clock auto-gating.

Figure 4-13 AINTC_IDLE Register

31	2	1	0
Reserved		TURBO	FUNCIDLE
R-0		R/W-0	R/W-0

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 4-11 AINTC_IDLE Register Field Definitions

Bit	Field	Description
31-2	Reserved	Read returns reset value
1	TURBO	Input synchronizer clock auto-gating 0x0: Input synchronizer clock is free-running (default). 0x1: Input synchronizer clock is auto-gated based on interrupt input activity.
0	FUNCIDLE	Functional clock idle mode 0x0: Functional clock gating strategy is applied (default). 0x1: Functional clock is free-running.
End of Table 4-11		

4.5.9 AINTC_IRQ_PRIORITY Register

This register supplies the currently active IRQ priority level.

Figure 4-14 AINTC_IRQ_PRIORITY Register

31	7	6	0
SPURIOUSIRQFLAG		IRQPRIORITY	
R-0x1FFFFFF		R-0	

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 4-12 AINTC_IRQ_PRIORITY Register Field Definitions

Bit	Field	Description
31-7	SPURIOUSIRQFLAG	Spurious IRQ flag
6-0	IRQPRIORITY	Current IRQ Priority
End of Table 4-12		

4.5.10 AINTC_FIQ_PRIORITY Register

This register supplies the currently active FIQ priority level.

Figure 4-15 AINTC_FIQ_PRIORITY Register

31	7	6	0
SPURIOUSFIQFLAG		FIQPRIORITY	
R-0x1FFFFFF		R-0	

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 4-13 AINTC_FIQ_PRIORITY Register Field Definitions

Bit	Field	Description
31-7	SPURIOUSFIQFLAG	Spurious FIQ flag
6-0	FIQPRIORITY	Current FIQ Priority
End of Table 4-13		

4.5.11 AINTC_THRESHOLD Register

This register sets the priority threshold.

Figure 4-16 AINTC_THRESHOLD Register

31	8	7	0
Reserved		PRIORITYTHRESHOLD	
R-0		R/W-0xFF	

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

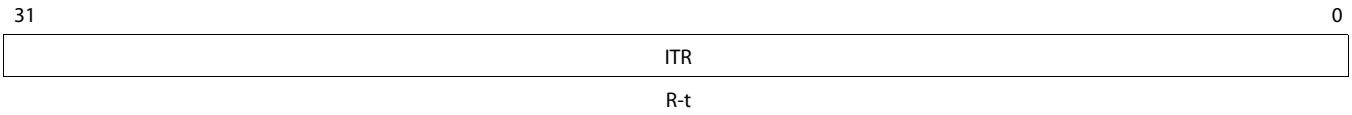
Table 4-14 AINTC_IRQ_PRIORITY Register Field Definitions

Bit	Field	Description
31-8	Reserved	Write 0s for future compatibility. Read returns reset value
7-0	PRIORITYTHRESHOLD	Priority threshold Write 0xFF: Priority threshold disabled Write 0x0 to 0x7F: Priority threshold enabled
End of Table 4-14		

4.5.12 AINTC_ITRn Register

This register shows the raw interrupt input status before masking.

Figure 4-17 AINTC_ITRn Register



Legend: R = Read only; R/W = Read/Write; -t = depends on interrupt inputs

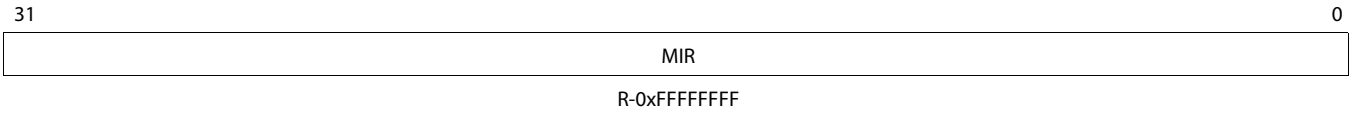
Table 4-15 AINTC_ITRn Register Field Definitions

Bit	Field	Description
31-0	ITR	Interrupt status before masking. each bit corresponds to each event. AINTC_ITR0: event 0 ~ 31 AINTC_ITR1: event 32~ 63 AINTC_ITR2: event 64~ 95 AINTC_ITR3: event 96~ 127
End of Table 4-15		

4.5.13 AINTC_MIRn Register

This register contains the interrupt mask.

Figure 4-18 AINTC_MIRn Register



Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

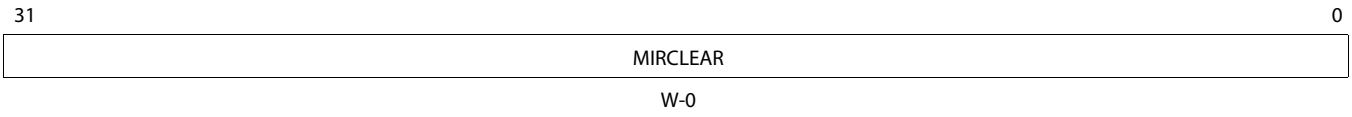
Table 4-16 AINTC_MIRn Register Field Definitions

Bit	Field	Description
31-0	MIR	Interrupt mask. each bit corresponds to each event 0x1: The interrupt is masked 0x0: The interrupt is unmasked AINTC_MIR0: event 0 ~ 31 AINTC_MIR1: event 32~ 63 AINTC_MIR2: event 64~ 95 AINTC_MIR3: event 96~ 127
End of Table 4-16		

4.5.14 AINTC_MIR_CLEARn Register

This register is used to clear the interrupt mask bits.

Figure 4-19 AINTC_MIR_CLEARn Register



Legend: W= Write only; R/W = Read/Write; -*n* = value after reset

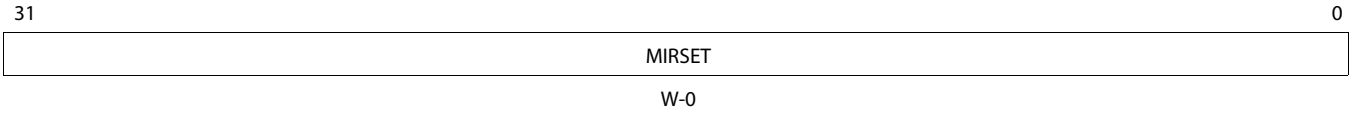
Table 4-17 AINTC_MIR_CLEARn Register Field Definitions

Bit	Field	Description
31-0	MIRCLEAR	Clear the interrupt mask bits. Read returns 0. each bit corresponds to each event. Write 0x1: Clears the MIR mask bit to 0 Write 0x0: No functional effect AINTC_MIR_CLEAR0: event 0 ~ 31 AINTC_MIR_CLEAR1: event 32~ 63 AINTC_MIR_CLEAR2: event 64~ 95 AINTC_MIR_CLEAR3: event 96~ 127
End of Table 4-17		

4.5.15 AINTC_MIR_SETn Register

This register is used to set the interrupt mask bits.

Figure 4-20 AINTC_MIR_SETn Register



Legend: W= Write only; R/W = Read/Write; -*n* = value after reset

Table 4-18 AINTC_MIR_SETn Register Field Definitions

Bit	Field	Description
31-0	MIRSET	Mask the interrupt bits. Read returns 0. each bit corresponds to each event Write 0x1: Set the MIR mask bit to 1 Write 0x0: No functional effect AINTC_MIR_SET0: event 0 ~ 31 AINTC_MIR_SET1: event 32~ 63 AINTC_MIR_SET2: event 64~ 95 AINTC_MIR_SET3: event 96~ 127
End of Table 4-18		

4.5.16 AINTC_ISR_SETn Register

This register is used to set the software interrupt bits. It is also used to read the currently active software interrupts.

Figure 4-21 AINTC_ISR_SETn Register

Legend: W= Write only; R/W = Read/Write; *-n* = value after reset

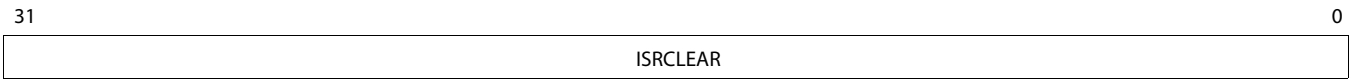
Table 4-19 AINTC_ISR_SETn Register Field Definitions

Bit	Field	Description
31-0	ISRSET	Set the software interrupt bits. Read returns the currently active software interrupts. each bit corresponds to each event Write 0x0: No functional effect Write 0x1: Sets the software interrupt bits to 1 AINTC_ISR_SET0: event 0 ~ 31 AINTC_ISR_SET1: event 32~ 63 AINTC_ISR_SET2: event 64~ 95 AINTC_ISR_SET3: event 96~ 127
End of Table 4-19		

4.5.17 AINTC_ISR_CLEARn Register

This register is used to clear the software interrupt bits.

Figure 4-22 AINTC_ISR_CLEARn Register



Legend: W= Write only; R/W = Read/Write; *-n* = value after reset

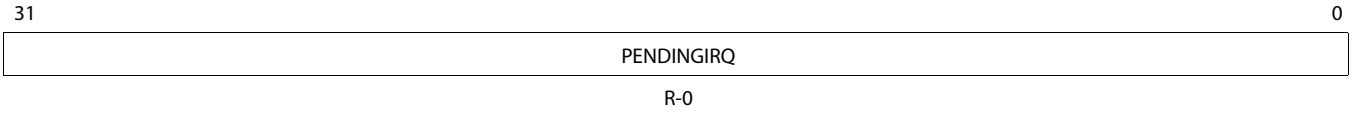
Table 4-20 AINTC_ISR_CLEARn Register Field Definitions

Bit	Field	Description
31-0	ISRCLEAR	Clear the software interrupt bits. Read returns 0. each bit corresponds to each event Write 0x0: No functional effect Write 0x1: Clears the software interrupt bits to 0 AINTC_ISR_CLEAR0: event 0 ~ 31 AINTC_ISR_CLEAR1: event 32~ 63 AINTC_ISR_CLEAR2: event 64~ 95 AINTC_ISR_CLEAR3: event 96~ 127
End of Table 4-20		

4.5.18 AINTC_PENDING_IRQn Register

This register contains the IRQ status after masking.

Figure 4-23 AINTC_PENDING_IRQn Register



Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

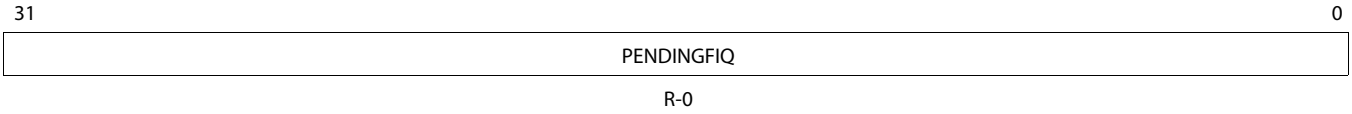
Table 4-21 AINTC_PENDING_IRQn Register Field Definitions

Bit	Field	Description
31-0	PENDINGIRQ	IRQ status after masking. each bit corresponds to each event AINTC_PENDING_IRQ0: event 0 ~ 31 AINTC_PENDING_IRQ1: event 32~ 63 AINTC_PENDING_IRQ2: event 64~ 95 AINTC_PENDING_IRQ3: event 96~ 127
End of Table 4-21		

4.5.19 AINTC_PENDING_FIQn Register

This register contains the FIQ status after masking.

Figure 4-24 AINTC_PENDING_FIQn Register



Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 4-22 AINTC_PENDING_FIQn Register Field Definitions

Bit	Field	Description
31-0	PENDINGFIQ	FIQ status after masking. each bit corresponds to each event AINTC_PENDING_FIQ0: event 0 ~ 31 AINTC_PENDING_FIQ1: event 32~ 63 AINTC_PENDING_FIQ2: event 64~ 95 AINTC_PENDING_FIQ3: event 96~ 127
End of Table 4-22		

4.5.20 AINTC_ILRm Register

These registers contain the priority for the interrupts and the FIQ/IRQ steering. There are total 128 AINTC_ILR registers and each register corresponds to each interrupt.

Figure 4-25 AINTC_ILRm Register (m: 0 ~ 127)

31 9	8 2	1	0
Reserved	PRIORITY	Reserved	FIQNIRQ
R-0	R/W-0	R-0	R/W-0

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 4-23 AINTC_ILRm Register Field Definitions

Bit	Field	Description
31-9	Reserved	Write 0 for future compatibility. Read returns reset value
8-2	PRIORITY	Interrupt priority
1	Reserved	Write 0 for future compatibility. Read returns reset value
0	FIQNIRQ	Interrupt IRQ FIQ mapping. Read returns reset value. Write 0x0: Interrupt is routed to IRQ. Write 0x1: Interrupt is routed to FIQ
End of Table 4-23		

4.6 INTD (CP_INTD) Registers

The following sections describe the registers required to configure the CP_INTD module for ARM CorePac.

Table 4-24 lists the register offsets for these registers.

Table 4-24 INTD Register Address Map

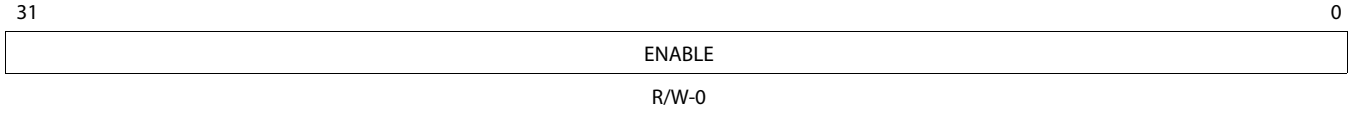
Register Name	Address Offset[1]
INTD_EN_SET	0x100 + (0x4*n)
INTD_EN_CLR	0x180+ (0x4*n)
INTD_STS_SET	0x200+ (0x4*n)
INTD_STS_CLR	0x280+ (0x4*n)
End of Table 4-24	

1. n = 0 to 2

4.6.1 Enable Set Registers (Base Address + 0x100, 0x104, 0x108)

The Enable Set Register enables IP interrupts to map to hosts. There are three registers per host for up to 87 interrupts. the first register programs INTD input interrupt 0 ~ 31(bit 0 is not used), second register programs 32 ~ 63 and the final one programs 64 ~ 86. Reading this register returns the enable values. Writing a 1 to a bit position enables that host/interrupt map.

Figure 4-26 INTD Enable Set Register



Legend: R = Read only R/W = Read/Write; *-n* = value after reset

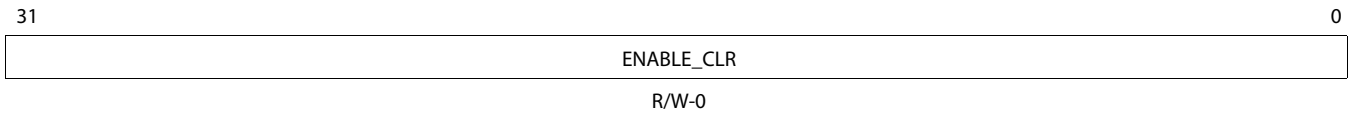
Table 4-25 INTD Enable Set Register Field Definitions

Bit	Field	Description
31: 0	ENABLE	IP interrupt enables. Write a 1 in a bit position to set that bit. Writing a 0 has no effect.
End of Table 4-25		

4.6.2 Enable Clear Registers (Base Address + 0x180, 0x184, 0x188)

The Enable Clear Register disables IP interrupts to map to hosts. There are three registers per host for up to 87 interrupts.(same input interrupt order like Enable Set Registers) Reading this register returns the enable clear values. Writing a 1 to a bit position disables that host/interrupt map.

Figure 4-27 INTD Enable Clear Register



Legend: R = Read only R/W = Read/Write; -*n* = value after reset

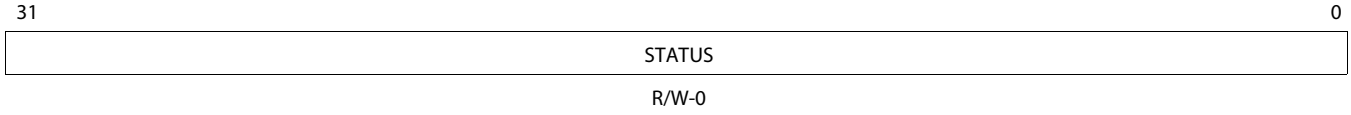
Table 4-26 INTD Enable Clear Register Field Definitions

Bit	Field	Description
31: 0	ENABLE_CLR	IP interrupt enable clear. Write a 1 in a bit position to clear that bit. Writing a 0 has no effect.
End of Table 4-26		

4.6.3 Status Set Registers (Base Address + 0x200, 0x204, 0x208)

The Status Set Register indicates which IP interrupts are active for the host. There are three registers per host for up to 87 interrupts.(same input interrupt order like Enable Set Registers) The mapping of bits in this register matches that of the Enable Registers. The software can write to the set register to set the pending status. An input interrupt will always set the pending status.

Figure 4-28 INTD Status Set Register



Legend: R = Read only R/W = Read/Write; -*n* = value after reset

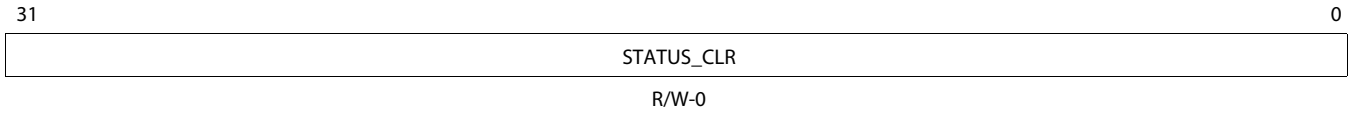
Table 4-27 INTD Status Set Register Field Definitions

Bit	Field	Description
31: 0	STATUS	Host IP interrupt status 0 = not pending 1 = pending Write a 1 in a bit position to set that bit. Writing a 0 has no effect
End of Table 4-27		

4.6.4 Status Clear Registers (Base Address + 0x280, 0x284, 0x288)

The Status Clear Register allows software to clear the pending status. This register is only valid and functional for pulsed interrupts and has no effect for level interrupts. There are three registers per host for up to 87 interrupts.(same input interrupt order like Enable Set Registers) The mapping of bits in this register matches that of the Enable Registers.

Figure 4-29 INTD Status Clear Register



Legend: R = Read only R/W = Read/Write; -*n* = value after reset

Table 4-28 INTD Status Clear Register Field Definitions

Bit	Field	Description
31: 0	STATUS_CLR	Host IP interrupt status 0 = not pending 1 = pending Write a 1 in a bit position to clear that bit. Writing a 0 has no effect
End of Table 4-28		

Chapter 5

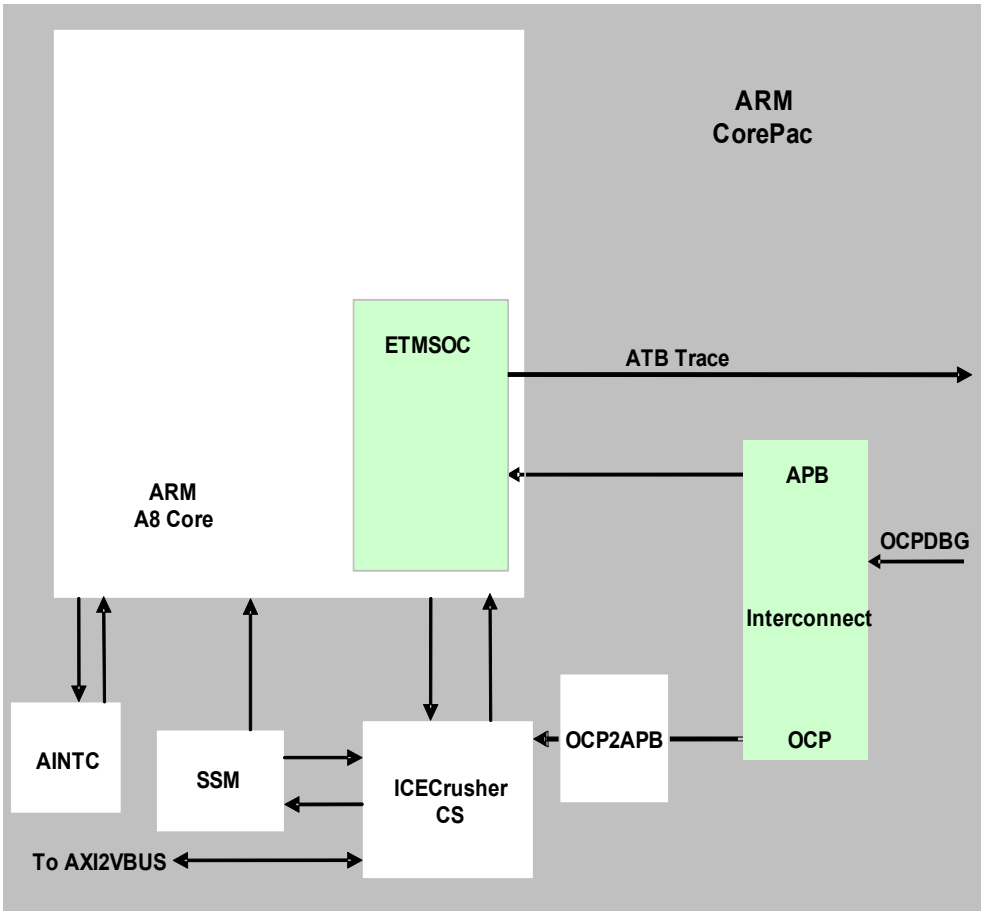
Debug, Emulation, Clock, and Reset

5.1 Debug/Emulation

The ARM A8 Core processor debug/emulation hardware includes a Debug Unit, a Coresight ETM (ETMSOC), and a Cross-Trigger Interface (CTI). The ARM CorePac contains logic to enable the DebugSS (Debug Subsystem) to access and control the ARM A8 Core debug/emulation hardware. Figure 5-1 shows the debug/emulation logic inside the ARM CorePac; the logic includes the ICECrusherCS and the OCP2APB Bridge.

The Debug OCP interface enables the DebugSS to access the Debug and Emulation registers on the Cortex-A8 and the ICECrusherCS. The OCP2APB Bridge converts the OCP protocol to APB and forward the requests to either the ARM A8 Core or the ICECrusherCS based on address mapping.

Figure 5-1 KeyStone ARM CorePac Debug/Emulation Logic



The debug/emulation logic inside the ARM CorePac is on the Emulation Power Domain. The ARM A8 Core is partitioned so that the ETMSOC is on the Emulation Power Domain.

5.1.1 ICECrusherCS

The ARM CorePac uses the ICECrusherCS module, which is similar to the ICECrusher on OMAP3430.

The main functions of the ICECrusherCS module are based on the Monitor mode and Halt mode features of the ARM core, with extended debug capabilities. Access to the ICECrusherCS is achieved via an APB interface. The ICECrusherCS is in the MPU power domain and MPU clock domain while the APB interface is in the Emulation power domain and PCLK clock domain.

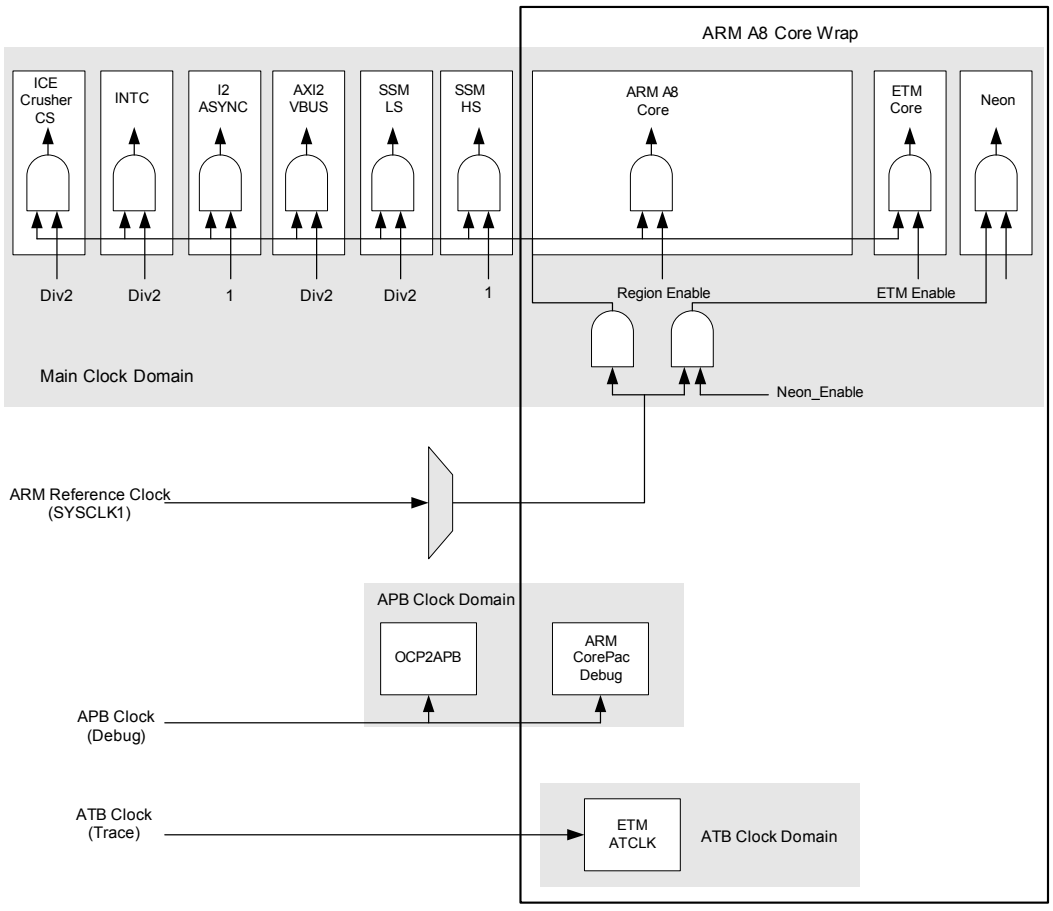
5.2 Clocking and Reset

5.2.1 ARM CorePac Clock Domain

The main chip-level PLL controller provides sys_clk1 to ARM CorePac. The ARM CorePac has an internal divider, which divides the sys_clk1 clock for its internal use. The ARM CorePac does not support local reset. It gets reset whenever the device is under reset. In addition, the interrupt controller inside the ARM CorePac can be reset only during $\overline{\text{POR}}$ and $\overline{\text{ResetFull}}$.

The ARM CorePac has three main clock domains, as shown in Figure 5-2. The three clock domains are Main Clock, APB Clock, and ATB Clock.

Figure 5-2 ARM CorePac Clock Domain

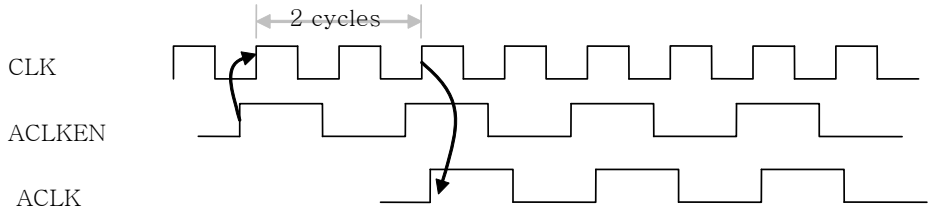


The ARM CorePac uses the reference clock (SYSCLK1) from DSP main PLL for use in the ARM A8 Core and the top logic in the main clock domain. The ARM CorePac top logic is running at half the speed by adding the DIV2 logic in the Regional enables.

The Neon Unit inside the ARM A8 Core has a separate physical clock grid that can be disabled when the ARM A8 Core is not executing Neon instructions. Clock management is fully controlled inside the ARM A8 Core. The Neon Clock is running at full-speed and is synchronized with the ARM A8 Core clock.

To ensure that data transfers happen at the correct clock edge between the ARM A8 Core logic (running at full-speed) and ARM CorePac top logic (running at half-speed), the DIV2 regional enables in ARM CorePac TOP must aligned to a delayed version of ACLKEN. The ARM A8 Core delays ACLKEN by two (full-speed) clock edges to determine the AXI clock edge, so the ARM CorePac top logic must be also aligned to the same clock edge.

Figure 5-3 Aligning Core Clock to ACLK using ACLKEN



The APB Clock is used for the debug logic in the ARM CorePac, which includes the Debug unit inside the ARM A8 Core. The APB clock is derived from the Debug OCP Port. The ATB Clock is used to drive the Trace unit (ETM ATCLK) within the ARM A8 Core.

5.2.2 ARM CorePac Reset

The ARM CorePac does not support local reset. It is reset whenever the device is under reset. In addition, the ARM interrupt controller (AINTC) can only be reset during POR and RESETFULL.

Index

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265