KeyStone II Architecture ARM CorePac

User Guide

Literature Number: SPRUHJ4
October 2012

Release History

Release	Date	Description/Comments
SPRUHJ4	October 2012	Initial Release

Contents

List of Tables

List of Figures

Preface

About This Manual

This document describes the ARM CorePac in the KeyStone II Architecture, but it does not describe the details of 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
Chip Interrupt Controller (CIC) for KeyStone Devices User Guide	SPRUGW4
C66x DSP Cache User Guide	SPRUGY8
TMS320TCI6614 Communications Infrastructure KeyStone SOC Silicon Errata	SPRS671

Trademarks

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 II devices handles transactions between components of the ARM Cortex-A15 (ARM A15 core) processor subsystem in TI SoCs. The ARM A15 Cluster is a 1- to 4-core implementation of the ARMv7-A architecture along with L1 and shared L2 caches. The ARM CorePac implements the ARM A15 cluster as a maximum quad-core variant (dual- or single-core also can be applied) with 4 MB of shared L2 cache.

The ARM CorePac includes CoreSight-compliant logic to allow the debug subsystem access to the ARM A15 core debug and emulation resources, which includes the embedded trace macrocell. The ARM CorePac has two primary clock domains and supports a number of clock/reset pairs for multiple internal interfaces.

The ARM CorePac also has special logic for connecting the AXI and ACP interfaces to the external SOC or memory like DDR or MSMC through the VBUSM interface and it uses a static power domain control scheme for the overall domain.

1.2 Terminology

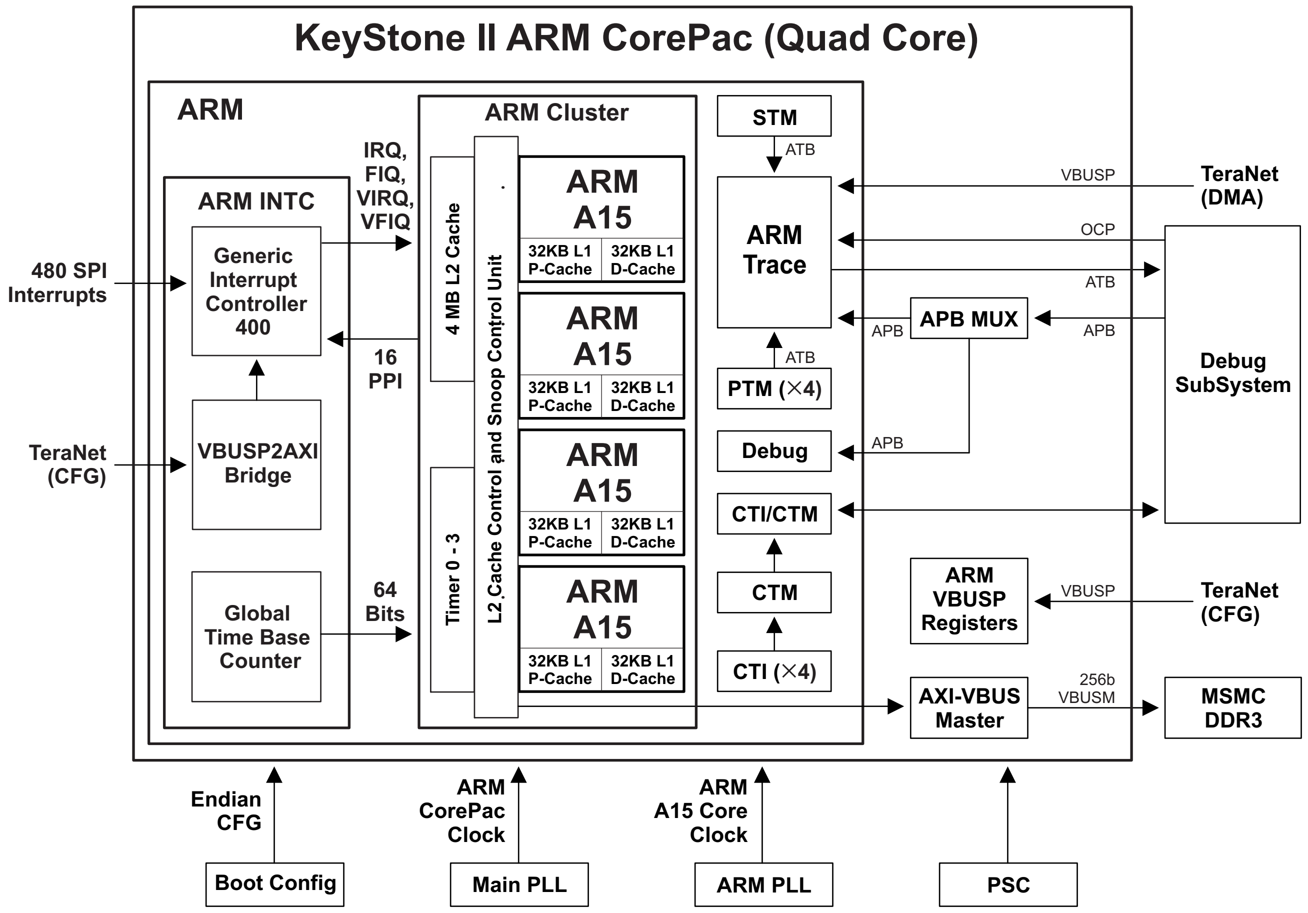
The following acronyms and abbreviations appear in this user guide.

Term	Definition
ACP	Accelerator Coherency Port
AINTC	ARM Interrupt Controller. In this document, it indicates GIC 400 and subsystem
AMBA	Advanced Microcontroller Bus Architecture - An open bus architecture developed by ARM Ltd
APB	AMBA Advanced Peripheral Bus - part of the AMBA 3 protocol family, the APB is a low-cost, simple, low-bandwidth interface
ATB	AMBA Trace Bus - A trace bus protocol released under the AMBA umbrella
AXI	Advanced eXtensible Interface (Latest AMBA interface from ARM Ltd)
CAM	Content Addressable Memory
CBA	Common Bus Architecture - a set of bus architectures developed by Texas Instruments
CTI	CoreSight Cross Trigger Interface
CTM	CoreSight Cross Trigger Matrix
CoreSight	The infrastructure for monitoring, tracing, and debugging a complete system on chip.
Cortex-A8	ARM v7 R2P3 core
Cortex-A15	ARM v7-A core
DFT	Design For Test
ETB	Embedded Trace Buffer
ICECrusherCS	TI debug/emulation logic
LPSC	ARM CorePac Local Power and Sleep Controller
MMR	Memory Mapped Register
MPU	Memory Protection Unit
OCP	Open Core Protocol
PMU	Performance Monitor Unit
PRCM	Power Reset Control Module
PSC	Power & Sleep Controller
PTM	Program Trace Macrocell
SIMD	Single Instruction Multiple Data
SECMON	Security Monitor
STM	System Trace Macro
TBR	Trace Buffer Router
TLB	Translation Lookaside Buffer

1.3 Block Diagram

Figure 1-1 shows the block diagram of the KeyStone II ARM CorePac.

Figure 1-1 KeyStone II ARM CorePac Block Diagram (Quad Core case)



The ARM CorePac contains the ARM A15 Cluster, which has up to four ARM A15 Cores, an L2 cache controller with snoop control unit, and 4 MB L2 cache RAM for all four cores. Each A15 core has its own timer, Debug module, PTM (Program Trace Macrocell), and CTI (Cross Trigger Interface); the cores share one CTM (Cross Trigger Macro).

The ARM CorePac-level CTM, CTI, APBMUX (AMBA Peripheral Bus MUX), and STM (System Trace Macrocell) are debug and trace-related modules. The AXI2VBUS Master bridge is a custom block that connects the AXI (Advanced eXtensible Interface) master interface from the ARM A15 cores to the MSMC and DDR in the SoC; it provides the AXI-VBUS protocol conversion as well as 128-bit-to-256-bit data bus width conversion with corresponding clock step-down.

All the ARM A15 core interfaces are asynchronous with the SoC. The ARM CorePac level instantiates half-bridges synchronous with the CPU clock on the APB, ATB (AMBA Trace Bus), AXI, and ACP (Accelerator Coherency Port) interfaces. The corresponding asynchronous half-bridge is instantiated and released as soft logic for

SoC-level synthesis. The ARM CorePac also includes a local PSC that interacts with the global PSC for power control, as well as config registers (VBUSP registers) and an ARM TRACE module that integrates the ARM CorePac debug logic with the SoC Debug subsystem.

The ARM CorePac has two PLL inputs: one from the SoC main PLL and one from the ARM PLL. The clock from the main PLL is used for several components in the ARM CorePac while the clock from the ARM PLL is used only for ARM Cores.

The ARM A15 core does not include an interrupt controller. Instead, a General Interrupt Controller (GIC-400) is provided by the ARM. The GIC-400 provides a scalable solution for up to 480 peripheral interrupts with 1-to-n broadcast capability. The ARM INTC (AINTC) subsystem includes the GIC-400 with a Global time base counter and a VBUSP2AXI interface for register configuration.

The ARM CorePac is configured to support 4 MB L2 cache with ECC/parity support on both L1 and L2. All the processors include the optional Neon and Vector floating-point units and are designed as independent power domains. L2 latency is the number of clock cycles it takes to access data from the memories and is a software-programmed value in the CP15 registers. Total data access time for an L1-miss followed by an L2-hit based on the RAM selections used in ARM CorePac is 19 core clock cycles.

1.4 References

This document does not have register details for the ARM Cortex-A15 and other submodules except for the VBUSP registers and AXI2VBUS master registers. This user guide assumes knowledge of TI KeyStone II architecture as well as knowledge of major IP integrated with the ARM CorePac such as the Debug subsystem, PSC, and DDR3 controller.

See the following resources for more information:

- For more detailed information about the ARM processor Cortex-A15 and submodules, visit the ARM information website:

  http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0344k/index.html
- See the *ARM Architecture Reference Manual (ARMv7-A)* for ARM instruction and coprocessor register details.
- See the *Cortex-A15 Technical Reference Manual (TRM)* for Cortex-A15 HW details.
- See the *CorLink GIC-400 Generic Interrupt Controller TRM* and *ARM Generic Interrupt Controller Architecture Specification* for ARM INTC (AINTC) submodule HW and register details.
- See the *CoreSight SoC TRM* for CTI,CTM,STM and ETB (Embedded Trace Buffer) details.
- See the *CoreSight Components TRM* for CTI,CTM,ETB HW detail and ARM TRACE module register details.
- See the *CoreSight System Trace Macrocell TRM* for STM HW and register details.

Chapter 2

Components of the ARM CorePac

This chapter describes the major components of the ARM CorePac for KeyStone II devices and how it is structured.

2.1 ARM A15 Cluster

The ARM A15 Cluster includes up to four Cortex A15 cores. The architecture highlights of the processor are:

- Full implementation of the ARMv7-A architecture instruction set
- Super scalar, variable-length, out-of-order pipeline (12 stage in-order, 3-12 stage out-of-order)
- Dynamic branch prediction with Branch Target Buffer (BTB) and Global History Buffer (GHB), a return stack, and an indirect predictor
- 32-entry fully-associative L1 Translation Lookaside Buffers (TLBs), one each for instruction fetch, data loads, and data stores
- Per CPU 512-entry four-way set-associative L2-TLB
- Fixed 32 KB L1 instruction and data caches
- Shared L2 cache of configurable size (4 MB used)
- Error Correction Code (ECC) protection for L1 data cache and L2 cache, parity protection for L1 instruction cache
- 128-bit instruction fetch
- Three-wide instruction decode
- Three-wide instruction dispatch
- Eight-wide instruction issue
- Out-of-order branch resolution
- Integrated Neon and VFP
- Program Trace Macrocell (PTM) based on the CoreSight Program Flow Trace (PFT) v1.1 architecture
- Performance Monitor Unit (PMU) based on PMUv2 architecture
- Cross Trigger Interface (CTI) and Cross Trigger Matrix (CTM) for multi-processor debugging
- ARM generic 64-bit timers for each processor
- Support for power management with multiple power domains

The ARM A15 Core has the following interfaces with other modules in the ARM CorePac:

- AXI (Advanced eXtensible Interface)
- APB (AMBA Advanced Peripheral Bus)
- ATB (AMBA Trace Bus)
- Cross Trigger

For more information, see the *ARM Architecture Reference Manual* and the *Cortex-A15 Technical Reference Manual (TRM).*

2.2 AXI2VBUS Master

The AXI2VBUS master is a custom block for interfacing the A15 Cluster AXI bus to system-level fabric through the MSMC. The MSMC is the global memory management unit that provides an interface between processor masters, such as the DSP CorePac and ARM A15 cores, to the system memory. The AXI2VBUS master serves as a bridge between the ARM protocols and the MSMC and also provides bus width and clock conversion.

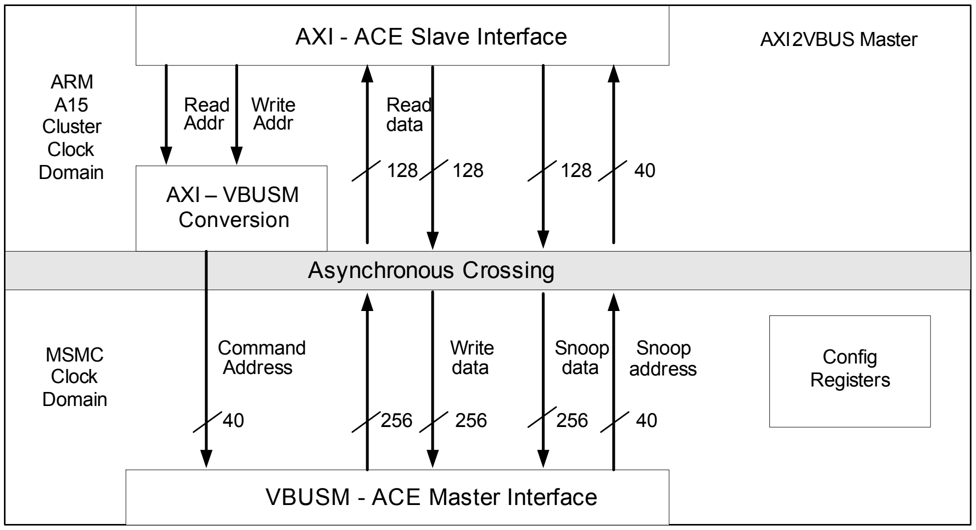
The AXI2VBUS master serves four main purposes on the ARM CorePac:

1. AXI – VBUSM protocol conversion
2. Coherency extensions translation between ACE and MSMC custom protocol
3. A 1:1 async boundary between the A15 Core and MSMC clocks while providing 128-bit:256-bit data-width conversion, leading to no loss in throughput overall
4. Mixed Endian conversion for data transmission

2.2.1 AXI2VBUS Master Block Diagram

Figure 2-1 shows a high-level conceptual view of the main features of the AXI2VBUS master. It serves as a bridge between the MSMC and the ARM A15 Cluster domains, providing bus protocol, bus width, and clock conversion.

Figure 2-1 AXI2VBUS Master Block Diagram



2.2.2 AXI2VBUS Master interfaces

The AXI2VBUS master has two primary interfaces: AXI-ACE Slave for the ARM A15 Cluster and the VBUSM + ACE Snoop Master for the MSMC/TeraNet.

2.2.2.1 AXI4-ACE Interface

The A15 Cluster uses a 40-bit address, 128-bit data for AXI-ACE master interface. The interface has separate channels for read, write, and snoop transactions. See the *AMBA-AXIACE Specification* for more information about the AXI and ACE interfaces.

2.2.2.2 VBUSM/ACE Master Interface

The MSMC uses a 40-bit address and 256-bit data VBUSM/ACE slave interface. The AXI2VBUS Master communicates additional metadata to external slaves. The metadata consists of VBUSM protocol signals and additional ACE sideband signals. The AXI2VBUS master interface also includes an ACE snoop command/data channel. The ACE Snoop Response and Snoop Data interface is shared with a secondary VBUSM (write only) command/write data interface.

2.2.3 AXI2VBUS Master Memory-Mapped Registers

The following sections describe the registers required to configure the AXI2VBUS Master. Table 2-1 lists the register offsets for these registers.

Table 2-1 AXI2VBUS Master Register Address Map

Register Name	Address Offset	Readable mode	Writable mode
AXI2VBUS_PID	0x00	All	None
AXI2VBUS_CMD_PRI	0x20	All	Emulation, Supervisor
AXI2VBUS_CPU0_END	0x30	All	Emulation, Supervisor
AXI2VBUS_CPU1_END	0x34	All	Emulation, Supervisor
AXI2VBUS_CPU2_END	0x38	All	Emulation, Supervisor
AXI2VBUS_CPU3_END	0x3c	All	Emulation, Supervisor
End of Table 2-1			

AXI2VBUS_CMD_PRI

- Non-secure device—writes by supervisor and emulation are allowed
- Secure device—writes by secure/non-secure supervisor and emulation are allowed

AXI2VBUS_CPUx_END

- Non-secure device—writes by supervisor and emulation are allowed
- Secure device—writes by secure supervisor are allowed

2.2.3.1 AXI2VBUS_PID Register

The PID register is a read-only register that identifies the version of the AXI2VBUS Master module in use. The format of the register is as per the Highlander peripheral ID convention. The version of the module is identified in the PID by the triple R.X.Y.

Figure 2-2 AXI2VBUS_PID Register

31 30	29 28	27 16	15 11	10 8	7 6	5 0
SCHEME	BU	FUNC	R	X	CUSTOM	Y
R-01	R-00	R-100100100000	R -0	R -0	R -0	R -0

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

Table 2-2 AXI2VBUS_PID Register Field Definitions (Part 1 of 2)

Bit	Field	Description
31-30	SCHEME	PID naming scheme (0x1)
29-28	BU	Business Unit - ASP (0x0)
27-16	FUNC	AXI2VBUS master function code (0x920)

Table 2-2 AXI2VBUS_PID Register Field Definitions (Part 2 of 2)

Bit	Field	Description
15-11	R	Minor Revision
10-8	X	Architecture Revision
7-6	CUSTOM	Reusable(0) vs Custom for a device (1)
5-0	Y	Configuration Revision
End of Table 2-2		

2.2.3.2 AXI2VBUS_CMD_PRI Register

This register provides the command priority information about the module. The reset value of AXI2VBUS_CMD_PRI is 0x7, the lowest VBUSM priority. Writing to this MMR only updates future transaction of VBUSM priority signal with the written value.

Figure 2-3 AXI2VBUS_CMD_PRI Register

31 3	2 0
Reserved	PRI
R-0	R/W-111

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

Table 2-3 AXI2VBUS_CMD_PRI Register Field Definitions

Bit	Field	Description
31-3	Reserved	Read returns reset value
0-2	PRI	Set Priority of the command. default is 0x7 which means the lowest VBUSM priority
End of Table 2-3		

2.2.3.3 AXI2VBUS_CPUn_END Register (0x30 ~ 0x3F)

This register provides AXI2VBUS master module endian mode setup and status for each CPU.

Figure 2-4 AXI2VBUS_CPUn_END Register (n = 0 ~ 3)

31 0
ENDIAN
R/W-t

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

Table 2-4 AXI2VBUS_CPUn_END Register Field Definitions

Bit	Field	Description
31-0	ENDIAN	CPU Endian mode configuration and status register Big endian: 0x00000000 (all zeros) Little endian: 0xFFFFFFFF (all ones)
End of Table 2-4		

2.2.4 Endian Support, Mixed-endian Conversion

The AXI2VBUS master supports both big and little endian mode on its AXI and VBUSM interfaces. This includes mixed-endian mode support, where one side is little endian and the other side is big endian.

The AXI master interface of A15 cores uses little endian mode for all data transfers. For non-MMR space accesses, AXI2VBUS master passes through data unmodified when the VBUSM side is little endian. AXI2VBUS master do byte swap for all data when the VBUSM side is big endian—packing and unpacking accordingly between 126/256-bit.

For MMR space accesses, although the AXI master interface uses the equivalent of VBUSM little endian, the A15 cores rearrange the data to the correct operating endian view of that core. In this scenario, the AXI2VBUS master does the appropriate swapping to achieve the same 32-bit register view of the MMR space access in the ARM A15 core according to the A15 core and MSMC endian combination.

The AXI2VBUS master supports endian mode on a per-core basis for an A15 Cluster (up to four cores). Because the A15 core supports runtime configuration of a core view of its data memory accesses, the AXI2VBUS master tracks each core's current endian view via internal MMRs(AXI2VBUS_CPUn_END registers). These MMRs must be programmed each time an A15 core endian state changes. It is a responsibility of user software to issue memory barrier operations and update the respective CPU endian MMR in AXI2VBUS master whenever a change of A15 core endian state is made. This maintains the appropriate endian conversion scheme.

Note—Changing the MMR endian value while there are transactions in-flight can result in unexpected endian-swapping behavior. Issuing a barrier to flush out all previous in-flight commands is necessary to guarantee correct data-endian swapping behavior.

These MMRs (AXI2VBUS_CPUn_END registers) take on the value of the A15 Cluster lendian config port during reset and require that all 32- bits be written to with either all 1s (Little Endian) or all 0s (Big Endian). The AXI2VBUS master physically represents each MMR as a single one-bit MMR and logically replicates that value to all 32 bits for MMR reads. The AXI2VBUS master looks only at one bit from the MMR word write. To make the writes to these MMRs endian agnostic, writes to these MMRs should be either all 0s or all 1s. Each AXI2VBUS_CPUn_END register is writeable only by non-cacheable accesses from its assigned A15 core.

2.2.5 Cache Line Size

The MSMC and A15 core both use a cache line size of 64 bytes.

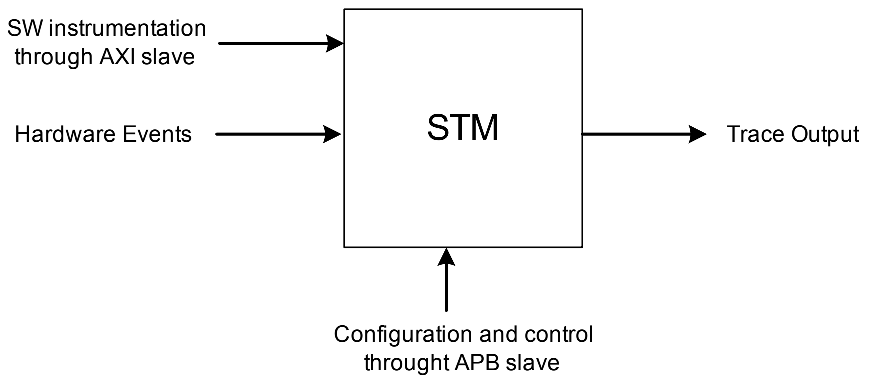
2.3 Coresight Debug Components

The ARM CorePac integrates a number of Coresight debug components around the A15 Cluster for supporting embedded debug trace as well as cross triggers. These are in addition to the PTMs inside the A15 cores and the CTM and CTI modules at the A15 core L2 level. The main debug components are described below.

2.3.1 STM

STM (System Trace Macrocell) is used to trace system activity from various sources such as hardware events and instrumented software using memory-mapped stimulus ports. An abstract model of the STM is shown in Figure 2-5.

Figure 2-5 Abstract Model of STM


SW instrumentation through AXI slave
Hardware Events
STM
Trace Output
Configuration and control throught APB slave

It is assumed that the APB slave configuration registers are accessed by on-chip debug masters and the external Debug subsystem. It is further assumed that software instrumentation trace will be the primary use mode on ARM CorePac, especially since there is no dedicated data trace capability on the A15 core. The AXI slave interface is a write-only interface accessible from any of the cores inside A15 Cluster.

On ARM CorePac, the AXI interface is accessible from any A15 core in the cluster as master, with each master occupying 16 MB of address space. Within the 16-MB address space, one master can access up to 65536 stimulus ports.

The STM also supports a DMA interface for efficient trace data transfer, but this is not used on the KeyStone II ARM CorePac. The trace output is combined with the program trace data from the A15 Core PTMs through funnels and buffers and combined with other trace streams in the system.

Note—The STM feature is disabled by default via the STM disable register. You must enable the STM to use it. When you do use it, there may be performance degradation via stalling the AXI interface. You must also turn on/turn off the STM by using barrier instruction to make sure all AXI transactions are complete.

2.3.1.1 Interfaces and Connectivity

ATB master—This is used for sending STM generated trace data to the SoC-level ATB FIFOs/funnels and buffers via an async bridge.

APB slave—The STM APB slave is combined with the A15 core APB slave through an APBMUX and exported to a SoC-level via a custom async APB bridge. The APB interface lets the STM be accessed as a memory-mapped peripheral.

Cross-trigger interface—Connects to the ARM CorePac-level CTI module. This interface is synchronous because modules share the same clock.

AXI slave—This is a write-only AXI slave, but the interface supports full Read address/data, Write address/data and Write response channels. The interface supports 32-bit address mode only.

Timestamp interface—The STM can be configured to accept 64-bit gray code value, but ARM CorePac uses the 64-bit binary configuration to maintain consistency with A15 Core PTM connections. The value needs to be synchronous to STM clock.

Programming and MMR Details are provided in the *CoreSight System Trace Macrocell TRM*.

2.3.2 Trigger Network

ARM CorePac uses the standard ARM embedded cross trigger network for cross triggers between subsystems. The primary components that make up the trigger network are CTIs and CTMs.

The CTI is the cross trigger interface that enables subsystems to cross trigger with each other. The CTI maps incoming channel events onto trigger channels. It supports an APB interface that lets the CTI be accessed as a memory-mapped peripheral on the debug bus. The CTM block is a cross trigger matrix that helps combine multiple cross triggers. The CTM block controls the distribution of triggers and enables multiple CTIs to be linked together.

CTI APB slave interface—The CTI slave is combined with the A15 Core APB and STM APB slaves via an APBMUX and exported to a SoC level via a custom async APB bridge. The APB interface lets the CTI be accessed as a memory mapped peripheral.

Additional details and register information is provided in the *CoreSight Component TRM*.

2.3.3 ARM TRACE & APBMUX

2.3.3.1 ARM TRACE

Trace requirements express the need to support the capturing of all trace sources simultaneously using on-chip resources like a trace buffer while also supporting off-chip export of a subset of those trace resources without impacting the execution of embedded applications. The ARM Trace Cell architecture was developed to support these requirements by defining a trace solution for a logical grouping of trace sources. The ARM TRACE module is customized to support simultaneous tracing of its five trace sources.

This module has the following features:

- Provides a solution for managing trace traffic of a single ARM CorePac instance
- Two configuration slave interfaces that allow programmatic access to embedded trace components via the Debug subsystem
  - APB interface supports configuration access to the Trace Replicator, Trace Funnel, and Cross Trigger Interface components
  - OCP and VBUSP interface supports configuration access to the TBR (Trace buffer Router)
- Extraction of Trace Buffer Router memory contents via a system DMA

ATB interface—The AMBA Trace bus is the ARM-defined bus format for transporting trace data. There are various components defined for use on this bus in the Arm Debug TRM such as the trace funnel, trace buffer, trace port interface unit, etc. The component added on this interface inside ARM CorePac is a standard ARM async ATB bridge for the voltage domain crossing.

ATB funnel—The five 64-bit ATB busses converge at the ATB Funnel where they are merged to create a single 64-bit ATB bus. The ATB Funnel arbitration scheme and programmer interface is described in the *CoreSight SoC Technical Reference Manual*.

ATB Replicator—The ATB Replicator takes the 64-bit ATB bus created by the ATB Funnel and makes two copies of it. The copies are exposed as ATB Master #0 and ATB Master #1 (used by the TBR). A filtering mechanism is implemented within the ATB Replicator to help constrain the amount of traffic provided over either of the two copies. This filtering mechanism and the programmer's interface for the ATB Replicator is described in the *CoreSight SoC Technical Reference Manual*.

TBR (Trace Buffer Router)—The TBR is responsible for buffering large amounts of trace data provided to it over its 64-bit ATB slave interface. The TBR supports various modes of operation including one that is DMA friendly. Detailed information of the TBR, including its programmers interface, is described in *KeyStone II Debug & Trace User's Guide*.

Cross Trigger—The TBR includes support for input and output triggers related to the operation of the trace stream. These triggers can be controlled (for incoming triggers) by any trigger source in the system and seen (for outgoing triggers) by trigger destinations. The Coresight Cross Trigger Interface (CSCTI) is responsible for merging the TBRs CTI triggers into the CoreSight triggering infrastructure. For more information on the CSCTI and its programmer interface, see the *CoreSight DK Technical Reference Manual*.

2.3.3.2 APBMUX

The AMBA Peripheral Bus (APB) is a standard ARM-defined bus for peripheral access and is used on the ARM CorePac primarily for debug peripheral access clocked off a debug PCLK separate from the master CPU clock. The A15 Cluster has an embedded debug module with a slave APB interface; the STM and CTI integrated in ARM CorePac have APB slave interfaces as well. A custom APBMUX block has been built to combine these interfaces together into a single APB slave interface for external master access. A standard ARM async APB bridge IP is also provided for the voltage domain crossing.

2.3.4 APB Memory Map

The ARM CorePac supports an 832 kB address space, within which it contains a slave region for ARM TRACE and a separate slave region for other modules in the ARM CorePac. The offsets for these regions depend on the CorePac ID; that is, the decode is different according to whether it is CorePac0, or CorePac1, and so on.

The ARM TRACE APB memory map is not listed here; see the *KeyStone Debug and Trace User's Guide* for details. The composite SoC Debug subsystem view of the external APB interface to connect to maximum four ARM CorePac is listed below.

Table 2-5 Chip-Level APB Address Decode Mapping for Multiple ARM CorePacs

Debug_SS config APB master 832 KB address space
16kB - ARM TRACE #0 (for ARM CorePac#0)
16kB - ARM TRACE #1 (for ARM CorePac#1)
16kB - ARM TRACE #2 (for ARM CorePac#2)
16kB - ARM TRACE #3 (for ARM CorePac#3)
64kB - Reserved for Expansion
128kB - ARM CorePac#0
128kB - ARM CorePac#1
128kB - ARM CorePac#2
128kB - ARM CorePac#3
192kB - Reserved for Expansion
End of Table 2-5

Table 2-6 ARM CorePac ID Based Address Decoding (Part 1 of 2)

APB Address Offset	**Target**
CorePac ID = 0 Address Map	
0x00000-0x03FFF	ARM TRACE APB
0x04000-0x1FFFF	Reserved
0x20000-0x3FFFF	ARM CorePac APB
0x40000-0xFFFFF	Reserved
CorePac ID = 1 Address Map	
0x00000-0x03FFF	Reserved
0x04000-0x07FFF	ARM TRACE APB
0x08000-0x3FFFF	Reserved
0x40000-0x5FFFF	ARM CorePac APB
0x60000-0xFFFFF	Reserved

Table 2-6 ARM CorePac ID Based Address Decoding (Part 2 of 2)

APB Address Offset	Target
CorePac ID = 2 Address Map	
0x00000-0x07FFF	Reserved
0x08000-0x0BFFF	ARM TRACE APB
0x0C000-0x5FFFF	Reserved
0x60000-0x7FFFF	ARM CorePac APB
0x80000-0xFFFFF	Reserved
CorePac ID = 3 Address Map	
0x00000-0x0BFFF	Reserved
0x0C000-0x0FFFF	ARM TRACE APB
0x10000-0x7FFFF	Reserved
0x80000-0x9FFFF	ARM CorePac APB
0xA0000-0xFFFFF	Reserved
End of Table 2-6	

Table 2-7 lists the composite APB address map for the entire APB interface in one ARM CorePac.

Table 2-7 APB interface memory map for one ARM CorePac (Quad core case)

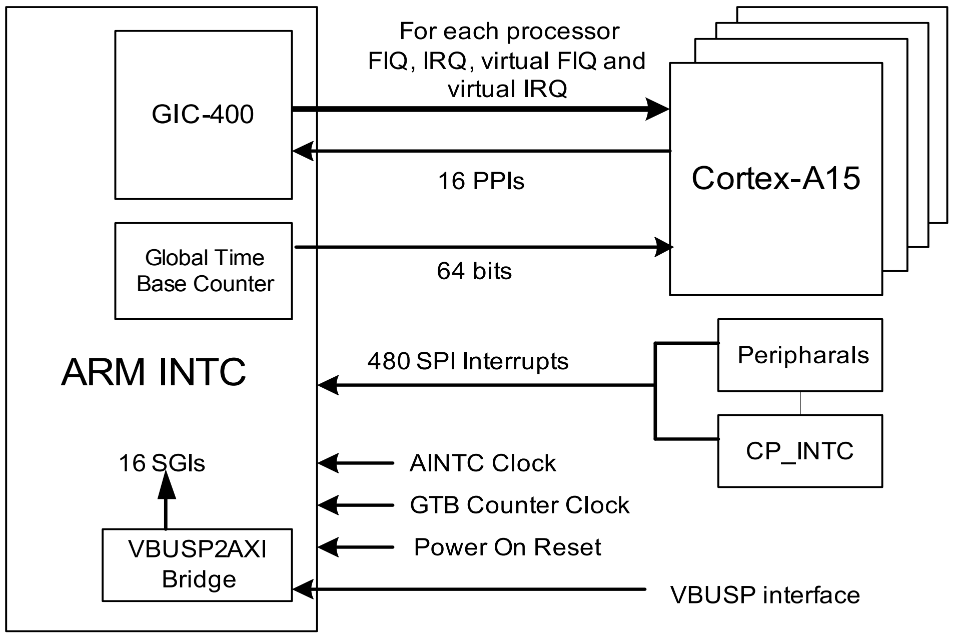
Address Range	Component
0x0E000 - 0x0EFFF	ARM CorePac level CTI
0x0F000 - 0x0FFFF	ARM CorePac STM
0x10000 - 0x10FFF	A15 Core 0 Debug
0x11000 - 0x11FFF	A15 Core 0 PMU
0x12000 - 0x12FFF	A15 Core 1Debug
0x13000 - 0x13FFF	A15 Core 1PMU
0x14000 - 0x14FFF	A15 Core 2Debug
0x15000 - 0x15FFF	A15 Core 2PMU
0x16000 - 0x16FFF	A15 Core 3Debug
0x17000 - 0x17FFF	A15 Core 3PMU
0x18000 - 0x18FFF	A15 Core 0 CTI
0x19000 - 0x19FFF	A15 Core 1CTI
0x1A000 - 0x1AFFF	A15 Core 2CTI
0x1B000 - 0x1BFFF	A15 Core 3CTI
0x1C000 - 0x1CFFF	A15 Core 0 Trace
0x1D000 - 0x1DFFF	A15 Core 1Trace
0x1E000 - 0x1EFFF	A15 Core 2Trace
0x1F000 - 0x1FFFF	A15 Core 3Trace
End of Table 2-7	

For a description of individual memory-mapped registers inside each component, see the *ARM Architecture Reference Manual (ARMv7-A)* and *Cortex-A15 Technical Reference Manual (TRM).*

2.4 ARM Interrupt Controller (AINTC)

The ARM Interrupt controller (AINTC) module includes an IP block that is provided by ARM. It is called the ARM generic interrupt controller (or GIC-400). The AINTC subsystem also includes a Global Time Base counter to implement 64-bit wide counter to get timer synchronization for ARM A15 Cores.The AINTC uses the AXI interface and the VBUSP2AXI bridge to provide read/write access to its internal memory-mapped registers. Figure 2-6 shows the block diagram of AINTC.

Figure 2-6 ARM Interrupt Controller (AINTC) Block Diagram



Detailed information about the GIC-400 is in the *CoreLink GIC-400 Generic Interrupt Controller* and *ARM Generic Interrupt Controller* architecture specifications.

2.4.1 Generic Interrupt Controller (GIC-400)

The GIC-400 is a high-performance, area-optimized controller with an AMBA Advanced eXtensible Interface (AXI) interface. It detects, manages, and distributes interrupts in the system. It also supports the separation of two groups of interrupts, typically secure and non-secure interrupts. In addition, the GIC-400 provides hardware acceleration for managing virtualized interrupts.

The GIC-400 implements the following numbers of the interrupt types.

- 16 Software Generated Interrupts (SGIs)
- Six external Private Peripheral Interrupts (PPIs)
- One internal PPI
- 480 Shared Peripheral Interrupts (SPIs)

The GIC-400 can assert the following signals to indicate pending interrupts to the CPU’s through physical and virtual interrupts. The wake-up interface is not used in AINTC.

- nFIQCPU[3:0]
- nIRQCPU[3:0]

- nVFIQCPU[3:0]
- nVIRQCPU[3:0]

The GIC-400 has a configuration register that allows software to configure each interrupt line to be level-sensitive or edge-sensitive. Because the hardware assumes all shared peripheral interrupts to be pulses, it is the software's responsibility to ensure that the GICD_ICFGR bits are set up correctly after reset. For more information about the GICD-ICFGR register, see the *ARM Generic Interrupt Controller* architecture document.

Note—The original ARM GIC Security Extension supports both secure interrupts (Group 0) and non-secure interrupts (Group 1) but all transaction made to GIC are treated as secure transaction in AINTC, so that the user has to follow the Secure mode procedure irrespective of ARM A15 Core mode.

2.4.2 VBUSP2AXI Bridge

The AINTC uses the AXI interface to provide read/write access to its internal registers. So, it is necessary to translate VBUSP transactions into AXI transactions. This protocol conversion is handled by the VBUSP2AXI bridge which is a module that is instantiated inside the AINTC. The bridge handles only the protocol conversion of VBUSP into AXI transactions. It does not perform any clock-rate adaptation or data-width conversion.

2.4.3 Global Timebase Counter

The AINTC implements a simple non-configurable 64-bit wide counter. The only time this counter gets reset is during power-on. The initialization value of the counter during power-on-reset is all zeros. After reset, the counter keeps incrementing by 1'b1 on each rising of the GTB Counter clock. Before the counter value is sent out to the timers inside the A15 core, its value is transcoded from 64-bit binary code into 64-bit graycode. This is done because the ARM CorePac has multiple voltage domains and therefore the timer value must be synchronized. Graycode encoding of the timer value simplifies the synchronization scheme greatly because inside the ARM CorePac, each bit of the timer value can be sent through a synchronizer because graycode ensures that only one-bit changes during state transition.

2.4.4 AINTC Clock

The AINTC has two clock domains. There is no phase relationship between the two clock domains which are, therefore, asynchronous to each other. However, there are no signals crossing between the two clock domains. The majority of the AINTC logic is triggered by the SoC div3 clock. The other clock coming into the AINTC module is used to trigger the global timebase counter which is exported to be used by the ARM A15 cores as a common source of time.

2.4.5 AINTC Reset

There are two resets coming into the AINTC module. The power-on reset is used only for the global timebase counter. Everything else inside the AINTC uses the normal reset. The GIC-400 uses asynchronous reset whereas the VBUSPAXI bridge and other miscellaneous logic is reset synchronously.

2.5 VBUSP Memory-Mapped Registers

The ARM CorePac supports two separate VBUSP interfaces: one on the ARM TRACE and the other one devoted to other ARM CorePac logic and ARM Local PSC(LPSC). This section shows the VBUSP memory map for ARM CorePac common logic and LPSC.

Table 2-8 lists the MMR map of the VBUSP registers. The SMA_SPARE is a simple 32-bit RW register that is unused currently. Note that all registers are reset synchronously by global warm reset; register defaults are latched when this reset goes high.

Table 2-8 VBUSP Memory Mapped Registers

Register Name	Address Offset	Description
ARMCOREPAC_PID	0x000	Peripheral Identification Register for ARM CorePac
AINTC_PID	0x004	Peripheral Identification Register for AINTC
STM_DISABLE	0x014	Disable accesses to STM in ARM CorePac
PD_CPU0_PTCMD	0x400	Power domain transition command for CPU0
PD_CPU0_PDSTAT	0x404	Power domain Status for CPU0
PD_CPU0_PDCTL	0x408	Power domain Control for CPU0
PD_CPU1_PTCMD	0x40C	Power domain transition command for CPU1
PD_CPU1_PDSTAT	0x410	Power domain Status for CPU1
PD_CPU1_PDCTL	0x414	Power domain Control for CPU1
PD_CPU2_PTCMD	0x418	Power domain transition command for CPU2
PD_CPU2_PDSTAT	0x41C	Power domain Status for CPU2
PD_CPU2_PDCTL	0x420	Power domain Control for CPU2
PD_CPU3_PTCMD	0x424	Power domain transition command for CPU3
PD_CPU3_PDSTAT	0x428	Power domain Status for CPU3
PD_CPU3_PDCTL	0x42C	Power domain Control for CPU3
End of Table 2-8		

2.5.1 ARMCOREPAC_PID Register

This register provides Peripheral Identification for the ARM CorePac.

Figure 2-7 ARMCOREPAC_PID Register

31 ... 0
TETRIS_PID
R-0x44902100

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

Table 2-9 ARMCOREPAC_PID Register Field Definitions

Bit	Field	Description
31-0	TETRIS_PID	ARM CorePAc PID Value (0x44902100)
End of Table 2-9		

2.5.2 AINTC_PID Register

This register provides Peripheral Identification for the AINTC module.

Figure 2-8 AINTC_PID Register

31 ... 0
TETRIS_INTC_PID
R-0x44911900

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

Table 2-10 AINTC_PID Register Field Definitions

Bit	Field	Description
31-0	TETRIS_INTC_PID	ARM INTC PID Value (0x44911900)
End of Table 2-10		

2.5.3 STM_DISABLE Register

This register disables or enables the STM module in the ARM CorePac.

Figure 2-9 STM_DISABLE Register

31 ... 1	0
Reserved	STM_DISABLE
R-0	R/W-1

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

Table 2-11 STM_DISABLE Register Field Definitions

Bit	Field	Description
31-1	Reserved	Read returns reset value
0	STM_DISABLE	0: STM feature enabled 1: STM feature disabled
End of Table 2-11		

2.5.4 PD_CPUn_PTCMD Register (n = 0 ~ 3)

This register is the power transition go command register. it is a pseudo-command register with no actual storage. Reads return 0. There is one register for each power domain.

Figure 2-10 PD_CPUn_PTCMD Register

31 ... 1	0
Reserved	GO_ CPU<n>
R-0	W-0

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

Table 2-12 PD_CPUn_PTCMD Register Field Definitions

Bit	Field	Description
31-1	Reserved	Read returns reset value
0	GO_CPU<n>	CPU<n> power domain GO transition. 0: Writes of 0 have no effect 1: Writes of 1 causes the ARM CorePac LPSC hardware to transit domain to PDCTL.NEXT_CPU state programmed one.
End of Table 2-12		

2.5.4.1 PD_CPUn_PDSTAT Register

This register is the power domain status register and it reflects status of the power domains. There is one register per CPU power domain. The general description is as follows.

Figure 2-11 PD_CPUn_PDSTAT Register

31 — 19	18 — 16	15 — 2	1 — 0
Reserved	Domain State	Reserved	NEXT_CPU
R-0	R-t	R -0	R-t

Legend: R = Read only; R/W = Read/Write; -*t* = actual state;

Table 2-13 PD_CPUn_PDSTAT Register Field Definitions

Bit	Field	Description
31-19	Reserved	Reserved
18-16	Domain State	Shows CPU<n> power domain actual state. 000: Powered ON 001: Powered ON and held-in-reset 010: Powered OFF. Dynamic Wakeup on interrupt. 011: Powered OFF. 100: Domain in transition NEXT state should always match current domain state unless it is in transition. The only exception is when debug prevents powering down.
15-2	Reserved	Reserved
1-0	NEXT_CPU<n>	Shows Programmed CPU<n> domain Next State. 00: Powered ON 01: Powered ON and held-in-reset 10: Powered OFF. Dynamic Wakeup on interrupt. 11: Powered OFF. These bits essentially reflect the NEXT state stored in the PDCTL shadow register inside the ARM CorePac. This field may not match the fields programmed into PDCTL until a GO transition command is issued
End of Table 2-13		

2.5.5 PD_CPUn_PDCTL Register (n = 0 ~ 3)

The is the power domain control register and it provides specific control for the ARM CorePac power domains. There is one register per CPU power domain. The general description is as follows.

Figure 2-12 PD_CPUn_PDCTL Register

31 ... 2	1 ... 0
Reserved	NEXT_ CPU<n>
R-0	R/W-t

Legend: R = Read only; R/W = Read/Write; -*t* = default value: power domain control next state

Table 2-14 PD_CPUn_PDCTL Register Field Definitions

Bit	Field	Description
31-2	Reserved	Read returns reset value
1-0	NEXT_CPU<n>	CPU<n> domain Next State. 00: Powered ON 01: Powered ON and held-in-reset 10: Powered OFF. Dynamic Wakeup on interrupt. 11: Powered OFF. If overall ARM CorePac domain is powered off, these states are ignored and the CPU domain is powered off as well.
End of Table 2-14		

2.6 Local Power and Sleep Controller (LPSC)

The ARM CorePac uses a static power domain control scheme for the overall ARM CorePac domains (from a Global PSC) and a choice of either static or dynamic control for the A15 core domains (from the Local PSC).

The A15 core power domains are managed statically or dynamically by the LPSC glue logic released as part of the ARM CorePac module. The LPSC works similar to a SoC level GPSC. A register equivalent to a GPSC PDCTL register can be programmed to set power states for an A15 core; a register equivalent to a GPSC PDSTAT register can be polled to see if the power transition actually happened. A register pair per A15 core is provided.

Four programmable power management modes are supported and fully verified: Powered ON, Powered ON and held-in-reset, Powered OFF and dynamic wakeup on interrupt, and Powered OFF. The modes are defined as follows:

- **Powered ON**: In this mode, the LPSC transitions the CPU power domain from OFF to ON when the domain specific GO bit is set in the PD_CPUn_PTCMD register. Power up does not happen if the overall ARM CorePac domain is off.
- **Powered ON and held-in-reset**: This mode is similar to the previous mode, with the difference that the CPU warm reset is not released once the domain is powered up. From this mode, the domain can be released from reset by programming to be in Powered ON mode.
- **Powered OFF**: In this mode, the domain transition from ON to OFF is done on a safe stopping boundary. It is the responsibility of user's software to set up this safe state.
- **Powered OFF and dynamic-wakeup on interrupt**: In this mode, the transition from OFF to ON does not require software intervention; the LPSC samples the interrupt lines going to the powered down core and dynamically initiates a wakeup if an interrupt occurs. It is expected that the GIC-400 is programmed to direct only the interrupts to be used as wakeup interrupts towards the A15 core to use this mode effectively. If the domain is ON when this mode is initiated, the LPSC will transition the domain ON to OFF on next, then proceed to wait on interrupt to wake up the domain.

Transitions from every state above to every other state through MMR programming is legal except for one: **Powered ON** to **Powered ON and held-in-reset**. Other than MMR based transitions, the only transition allowed is from **Powered OFF and dynamic-wakeup on interrupt** to **Powered ON** state automatically on an interrupt.

Chapter 3

Operation of the ARM CorePac

3.1 Endian Conversion

The ARM CorePac handles the case where the endianness of A15 is different from the endianess of the rest of the system. The following use cases can be used by the users; ARM and System in LE mode, ARM in LE mode and System in BE mode (Linux in LE running on ARM and rest of the system in BE mode), ARM and System in BE mode.

The AXI2VBUS master defines eight regions that will undergo a word swap instead of a byte swap for internal and external configuration region. The AXI2VBUS_CPU#_END registers are used to select which endian mode could be applied to each CPU data transaction with memory region.

The ARM A15 cores allows CP15 system control register EE bit defines to zero for LE and one for BE. Reset state for this is defined by CFGEND input. SCR.EE bit sets CPSR.E on entry to an exception vector. this CPSR.E defines the Load/Store endianness. ARM Instructions are always treated as little endian and BE implies support for byte invariant data endianness.

3.1.1 System-Level Consideration

Instruction code is always interpreted as LE in A15 cores. If the code is in BOOTROM or NOR flash, the code is compiled and burned into the ROM or NOR as little endian, so the byte 0 is LSB of instruction and stored in address 0.

If the system is in LE mode, EMIF would put address 0 in LSB of its VBUS interface and it does not make any conversion for reading and writing, but if the system is in BE mode, EMIF or BOOTROM in BE mode would put address 0 in MSB of its VBUS interface.

In this case, if both ARM and system is in LE mode, AXI2VBUS master doesn't need to do byte swap and AXI2VBUS_CPU#_END registers in the master can be set as LE (0x00000000). If ARM is in LE mode (by CFGEND pin and AXI2VBUS_CPU#_END registers) and system is in BE mode, byte swap should be done by AXI2VBUS master.

There are two corner cases which require re-swap of data by the user application when mixed endian mode is used (ARM is in LE and system is in BE)

3.1.1.1 Corner Case 1 (Shared Memory Access for Bit Field Operation)

Shared bit field memory access from ARM and system may cause trouble if the system runs in BE mode and ARM runs in LE mode, because the AXI2VBUS master will do endianness conversion for the data. For example, if the system (BE) writes 0x0000ABCD to the shared memory, it will be read by ARM (LE) as 0xCDAB0000. This means bit field 0 ~ 7 for system will be bit field 24~ 31 for ARM. The solution is swapping data before writing to the shared memory or after reading from the shared memory. For example, the ARM should write 0xCDAB0000 if 0x0000ABCD could be written to the shared memory. The same swapping could be done for reading.

3.1.1.2 Corner Case 2 (16-bit NOR Flash Access Through EMIF)

The ARM cannot correctly program 16-bit flash through EMIF when mixed endian mode is used (ARM is in LE and system is in BE) because 16-bit flash transfers 16-bit data at a time as a unit but AXI2VBUS master do byte level conversion (endian conversion).

For example, if the ARM tries to write 0xABCD to 16-bit NOR, the AXI2VBUS master will convert it to 0xCDAB and this will be delivered to flash. But 16-bit NOR cannot accept this, because the expected command is 0xABCD and not 0xCDAB (flash memory does not work correctly when it gets the wrong command). The solution for this is swapping 16-bit data before writing to flash. To write 0xABCD to flash, the user application software should swap it to 0xCDAB before writing.

3.2 Clock and Reset

3.2.1 Clock

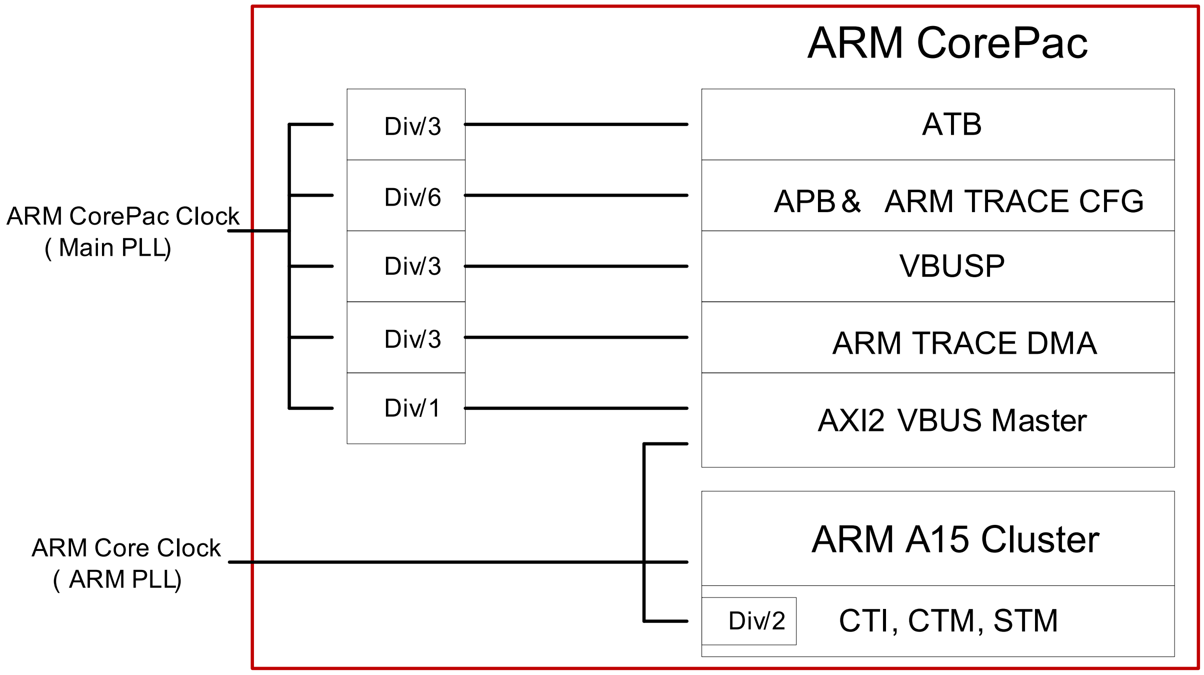
There are two primary functional clock inputs into the ARM CorePac:

- The ARM CorePac clock that comes from the SoC main PLL to drive submodules in the ARM CorePac
- The ARM core clock that comes from the ARM PLL to drive the A15 cores and CTI, CTM, and STM at the ARM CorePac level.

The ARM CorePac clock is the high-speed SoC div1 clock feeding distributed dividers in the ARM CorePac; there are separate dividers for generating clocks for ATB, VBUSP, APB, ARM TRACE, and AXI2VBUS master interfaces. The ARM LPSC is in the VBUSP clock domain.

Figure 3-1 is block diagram of the ARM CorePac clock.

Figure 3-1 ARM CorePac Clocking Diagram



3.2.2 Reset

The ARM CorePac supports a number of clock/reset pairs, one for each of the ATB, APB, VBUSP, OCP, and AXI2VBUS master interfaces. It is assumed that these resets are generated from an LPSC dedicated to controlling the ARM CorePac at the SoC PSC level.

The software reset feature of the ARM CorePac is not supported; there is no reset hookup between the global PSC and ARM CorePac. To truly support a soft reset, the ARM CorePac must go through a full powerdown software sequence and hardware disconnect sequence from the interconnect. Neither of these sequences is likely to complete in the midst of a CPU hang condition (which is the reason for a soft reset in the first place). The only recovery mechanism is therefore likely to be a system-level global warm reset.

Table 3-1 lists the mapping of SoC level reset events to their corresponding modules.

Table 3-1 ARM CorePac reset and SoC reset mapping

SoC Event	Effects on ARM CorePac
Global POR	The whole ARM CorePac is power cycled and reset
Global chip 0 warm reset	Reset except debug related logic in ARM CorePac level
Global chip 1 warm reset	Reset except debug related logic in ARM CorePac level but No power cycling
ARM CorePac power off from Global PSC	Power cycled and reset for A15 Cluster, STM, AXI2VBUS master

3.3 ARM CorePac Power

3.3.1 Overview

The power domains are controlled using a combination of a global PSC and local PSC. All the CPU power domains can be cycled ON and OFF independently, but the overall ARM CorePac domain cannot be turned off unless the underlying CPU domains are powered off. By design, the ARM CorePac is treated as an independent voltage domain from the SoC level. However, due to the high-speed/high-leakage nature of the process technology used, power domains with logic and memory powerdown provide opportunities for maximal power savings.

3.3.2 Powerup/Powerdown Sequences

The powerup/powerdown sequences that the LPSC follows are described below; a number of steps are required in software as part of the sequence that the LPSC would initiate through an interrupt. The ISR routine software steps are also described below.

3.3.2.1 Powering Down a CPU

A CPU powerdown is initiated by writing to the PD_CPU#_ PDCTL register in the LPSC to transition to either static powered off or powered off and wait for interrupt modes. In addition, interrupts must be disabled from the AINTC except for dynamic wake up interrupts if any.

1. The CPU is expected to reach WFI state following a series of steps in software, as follows:
   a. In software, block the cache allocation (by clearing the SCTLR.C or HSCTLR.C bit)
   b. Clean and invalidate all data from the L1 data cache
   c. Take the CPU out of coherency (by clearing the ACTLR SMP bit which switches the CPU from SMP mode to Asymmetric mode).
   d. Save the architectural state, if required.
   e. Execute an instruction barrier ISB instruction to commit all CP15 changes from previous steps.
   f. Execute a data barrier DSB instruction to commit all coherency operations from previous steps.
   g. Execute a WFI instruction.
2. The CPU internally gates its clocks.
3. The LPSC logic does a handshake with the SCR in front of the dedicated watchdog timer for the CPU to disable VBUS accesses to the timer. It then goes through a clock-stop handshake with the timer to idle it.
4. The Global PSC turns on isolation to the CPU domain, shuts down the memories, then powers down the CPU domain by turning off the switches.

5. After the domain is powered down, the watchdog timer reset is asserted to hold it in reset. The CPU domain resets are not asserted when powered down.

3.3.2.2 Powering Up a CPU

A CPU power up is initiated by writing to the PD_CPU#_ PWRCTL register in the LPSC or by a wake up interrupt (in dynamic power up mode).

1. The LPSC asserts the CPU domain resets, then proceeds to turn on power switches.
2. The LPSC turns on the memories in the domain.
3. The LPSC removes isolation, then de-asserts reset to allow the CPU to proceed with boot.

3.3.2.3 Powering Down an Entire ARM CorePac from SoC PSC

The powerdown of the entire ARM CorePac is initiated by programming the ARM domain OFF in the SoC PSC. Note that either a power-off or clock-off request at the SoC PSC is interpreted as a power down request to ARM CorePac.

1. The SoC PSC signals a domain power down to the LPSC.
2. The LPSC requests and waits for all CPUs to perform a power down sequence as described earlier. In parallel.
3. The LPSC requests the AXI2VBUS master to start refusing snoop requests from the MSMC.
4. When the AXI2VBUS master responds, the LPSC requests the A15 cores to idle the ACE and ACP respectively.
5. After the handshaking is done, the LPSC logic signals the overall ARM domain clock stop acknowledge to the SoC PSC to proceed with powerdown.
6. The LPSC asserts reset and isolation to the entire ARM domain, and waits 8 LPSC clock cycles. Note that isolation between the CPU and L2 domains is enabled even though both sides are turned off.
7. The LPSC shuts down the RAMs, then turns off the switches and signals the SoC PSC, which indicates ARM domain powerdown completion.

3.3.2.4 Powering Up an Entire ARM CorePac from SoC PSC

The powerup of the entire ARM CorePac is initiated by programming the ARM domain in the SoC PSC.

1. The LPSC turns on the power switches to the CPU and waits until the domain is fully powered.
2. The LPSC wakes up memories in order of RAM domains.
3. The LPSC initiates power up of all the CPU power domains.
4. The LPSC removes isolation from the ARM CorePac domain until reaching the ON state.

3.3.3 WFI and WFE Wake up events

ARM A15 cores uses both WFE and WFI instruction for clock idling, and WFI only for power domain cycling. However, the ARMv7 definitions of WFE and WFI wake-up events are modified by the Virtualized extensions in the following ways.

3.3.3.1 WFI Wake Up Events

1. A physical IRQ interrupt, regardless of the value of the CPSR.I bit
2. A physical FIQ interrupt, regardless of the value of the CPSR.F bit
3. A physical asynchronous abort, regardless of the value of the CPSR.A bit
4. A virtual IRQ interrupt, when in a Non-secure mode other than Hyp, and HCR.IMO is set, regardless of the value of the CPSR.I bit
5. A virtual FIQ interrupt, when in a Non-secure mode other than Hyp, and HCR.FMO is set, regardless of the value of the CPSR.F bit
6. A virtual asynchronous abort, when in a Non-secure mode other than Hyp, and HCR.AMO is set, regardless of the value of the CPSR.A bit
7. A debug event, when invasive debug is enabled and the debug event is permitted

3.3.3.2 WFE Wake Up Events

1. The execution of an SEV instruction on any processor in the multiprocessor system
2. A physical IRQ interrupt that is not masked by the CPSR.I bit
3. A physical FIQ interrupt that is not masked by the CPSR.F bit
4. A physical asynchronous abort that is not masked by the CPSR.A bit
5. A virtual IRQ interrupt that is not masked by the CPSR.I bit when in a Non-secure mode other than Hyp, and HCR.IMO is set
6. A virtual FIQ interrupt that is not masked by the CPSR.F bit, when in a Non-secure mode other than Hyp, and HCR.FMO is set

In addition, there are system-level implications to running multiple operating systems and virtualization / security. In a virtualized system, it is often desirable for the hyper visor to know that a Guest OS is in WFI pending an interrupt, so that another Guest OS can be run. In general it is expected that WFI will involve suspension of a Guest OS for a sufficient period of time that it is worth scheduling a different Guest OS. For this reason it is useful to be able to trap WFI to the hyper visor. When the HCR.TWI bit is set, the execution of a WFI causes a Hyper visor Entry Trap rather than an idle state; the power management scheme in ARM CorePac robustly handles this since it waits for the WFI output from the CPU to be asserted indicating true processor idle.

Similar to WFI, WFE is a mechanism for suspending operation during the polling of a variable, and as such can present an opportunity for rescheduling which virtual machine is running; an HCR.TWE bit can be set to enable a hyper visor entry trap on WFE instruction. However, as WFE is likely to be of shorter duration than WFI, this is not used to power down a CPU and instead only the internal clock gating feature of an A15 core on WFE.

3.4 ARM CorePac Program Example

This section presents low-level example code to perform basic ARM executable tasks. Low-level drivers or high-level support libraries are not available for ARM CorePac, but this document describes example code to show how to program ARM for initialization and basic operation. The assembly code in this chapter used the TI TMS470 ARM compiler and is different from GCC or ARM RVCT.

3.4.1 Entry Point Module Example

The entry point module _c_int00_supervisor performs the following tasks:

- Clears the .bss section
- Initializes the .cinit section
- Creates unique stack sections and initialize stack pointers for supervisor (SVC), IRQ and user (USR) modes of the processor
- Initializes the exception vector table and routes the interrupts to the defined table by programming the 'Vector Base Address Register' (VBAR) register
- Calls main() at the end

The processor will be in 'USR' mode when the program control reaches main().

```
_c_int00_supervisor:

    ;; clear bss memory
    ldr  r0, val_bss_start
    ldr  r1, val_bss_end
    mov  r2, #0

cloop:
    cmp r1, r0
    ble cloop_done

    str r2, [r0]
    add r0, r0, #4
    cmp r1, r0
    b   cloop

cloop_done:

    ;;  cinit setup
    ldr r0, val_cinit_start

cinit_section:
    ldr r1, [r0]   ;; r1 has the section length
    cmp r1, #0
    beq cinit_done

    add r0, r0, #4
    ldr r2, [r0] ;;  r2 has the base address
    add r0, r0, #4
    sub r3, r3, r3

cinit_loop:

    ldr r4, [r0]
    str r4, [r2]
    add r0, r0, #4
    add r2, r2, #4
    add r3, r3, #4
    cmp r3, r1
    blt cinit_loop

    b cinit_section

cinit_done:

stack_setup_start:
    ;;  Setup the stack - SVC Mode (Already in SVC Mode)
    ldr r0, svc_stack_end
    mov sp, r0
    ;;  Setup the stack - IRQ Mode
    cps #Mode_IRQ_32
    ldr r0, irq_stack_end
    mov sp, r0
stack_setup_end:

    cps   #Mode_SVC_32    ;; Go back to SVC mode

    bl _arm_init_vbar_nsec
    bl _arm_sys_init    ;; System Init

usr_stack_setup_start:
    ;;  Setup the stack - USR Mode
    cps #Mode_USR_32
    ldr r0, usr_stack_end
    mov sp, r0
usr_stack_setup_end:

b _main
```

3.4.2 Enable IRQ Example

This section shows how to enable IRQ type interrupts. This call must be made in order for any interrupt events to be received by the A15 ARM cores.

```
_arm_enable_irq:
    stmfd   sp!,{r0-r4}            ;; PUSH
    cpsie i            ;; Enable IRQ
    ;; Enable IRQ in USR mode
    mrs     r4, SPSR               ;; Take a copy of Program Status Register
    bic     r4,r4,#Irq_Bits        ;; Clear the Irq bits (0x80)
    msr     SPSR_cxsf, r4          ;; Write back the modified SPS
    ldmfd   sp!,{r0-r4}            ;; POP
    mov   pc, r14;        ;; return
end_arm_enable_irq:
```

3.4.3 Enable iCache Example

This section shows how the L1 and L2 data and instruction caches are enabled. Both L1 and L2 caches are disabled coming out of reset.

```
_arm_enable_ICache:
stmfd    sp!,{r0-r4}              ; Push Stack
mov   r1,#2
mcr   p15,#2,r1,c0,c0,#0          ; Enable Level2 unified cache.
mrc   p15, #0, r1, c1, c0, #0     ; Read Control Register
mov   r2, #1
orr   r1, r1, r2, lsl #12         ; Set I bit (instruction cache enable)
orr   r1, r1, r2, lsl #2          ; Set C bit (data cache enable)
mcr   p15, #0, r1, c1, c0, #0     ; Write Control Register
mrc   p15, #0, r1, c1, c0, #1     ; Read Auxiliary Control Register
mov   r2, #2
orr   r1, r1, r2                  ; Set L2EN bit
mcr   p15, #0, r1, c1, c0, #1     ; Write Auxiliary Control Register
ldmfd    sp!,{r0-r4}              ; Pop stack
mov   pc, r14;                   ;; return
end_arm_enable_ICache:
```

3.4.4 ARM Mode Change Example

Go to "SUPERVISOR" mode:

```
_arm_goto_mode_supervisor:
    stmfd   sp!,{r0-r4}             ;; PUSH
    mrs     r4, SPSR                ;; Take a copy of Program Status Register
    bic     r4,r4,#Mode_Bits        ;; Clear the mode bits
    orr     r4,r4,#Mode_SVC_32      ;; Set Supervisor Mode
    msr     SPSR_cxsf, r4           ;; Write back the modified SPS
    ldmfd   sp!,{r0-r4}             ;; POP
    mov     pc, r14;                ;; return
end_arm_goto_mode_supervisor:
Go to "USER" mode:
_arm_goto_mode_user:
    stmfd   sp!,{r0-r4}             ;; PUSH
    mrs     r4, SPSR                ;; Take a copy of Program Status Register
    bic     r4,r4,#Mode_Bits        ;; Clear the mode bits
    orr     r4,r4,#Mode_USR_32      ;; Set Supervisor Mode
    msr     SPSR_cxsf, r4           ;; Write back the modified SPS
    ldmfd   sp!,{r0-r4}             ;; POP
    mov     pc, r14;                ;; return
end_arm_goto_mode_user:
```

3.4.5 Enable and Read ARM Cycle Counter Example

ARM cycle counter works similar like TI CorePac TSCL. this routine enables, starts and read the ARM cycle counter.

```
_arm_enable_ccntr:
     stmfd    sp!,{r0-r4}
     mrc   p15, #0, r1, c9, c12, #1  ;; Read count enable set reg
     orr    r1,r1,#0x80000000        ;; Set CCNT enable bit
     mcr   p15, #0, r1, c9, c12, #1  ;; Write count enable set
     mrc   p15,#0,r2,c9,c12,#0       ;; Read PMNC reg
     orr   r2, r2, #1                ;; Set All counter enable bit
     mcr   p15,#0,r2,c9,c12,#0       ;; Write PMNC reg
     mrc   p15,#0,r3,c9,c12,#0       ;; Read PMNC reg

     ;; Enable Performonce monitor reads in USR mode
     mrc   p15, #0, r2, c9, c14, #0  ;; Write USEREN Register
     orr   r2, r2, #1                ;; Set USEREN Bit
     mcr   p15, #0, r2, c9, c14, #0  ;; Write USEREN Register

     ldmfd    sp!,{r0-r4}
     mov   pc, r14;                 ;; return
end_arm_enable_ccntr:
_arm_read_ccntr:
     mrc   p15, #0, r0, c9, c13, #0 ; Read CCNT Register
     mov   pc, r14;                 ;; return
end_arm_read_ccntr:
```

3.4.6 SWI Code and Function Example

This section describes Interrupt related code and functions.

```
INT_Swi:
        STMFD   sp!, {r4-r14}
        LDR     r5, [LR, #-4]       ; Retrieve SWI parameter
        AND     r5, r5, #0xFF       ; Keep the argument only
        STMFD   sp!, {r5}           ; Save R5
        cmp     r5, #0              ; Check SWI Id
        SMIEQ   #0
        LDMFD   sp!, {r5}           ; Restore R5
        MOV     R0, R5              ; Copy ID to R0
        cmp     r5, #0              ; Check SWI Id
        BLNE    _svc_intr_isr       ; Call handler
        B       ExitSwi
SetSupervisor
MRS     r7, SPSR                          ; Take a copy of Program Status Register
BIC     r7,r7,#Mode_Bits                  ; Clear the mode bits
ORR     r7,r7,#Mode_SVC_32                ; Set Supervisor Mode
MSR     SPSR_cxsf, r7                     ; Write back the modified SPSR
B       ExitSwi
```

```
SetUser
MRS     r7, SPSR                            ; Get copy of Program Status Register
BIC     r7, r7, #Mode_Bits                  ; Clear mode bits
ORR     r7, r7, #Mode_USR_32                ; Set User Mode
MSR     SPSR_cxsf, r7                       ; Write back modified SPSR
B       ExitSwi

EnableIRQ
;MOV     r5, #Mode_IRQ_32
MOV     r6, #IBit
;Set FIQ or IRQ mode before reset I/F bit to prevent interruption
MRS     r4, CPSR                            ; Read current PSR
BIC     r4,r4,#Mode_Bits                    ; Remove all mode bits
ORR     r4,r4,r5                            ; Set desired mode
MSR     CPSR_c, r4                    ; Set current mode to disable FIQ or IRQ catch
                                           ;Stack manipulation
MOV     SP, r0                              ; Stack address
MOV     r1,r1,lsl #2                        ; Convertion from word size to byte size
ADD     r0,r1,SP
MOV     SP, r0

BIC     r4,r4,#Mode_Bits                    ; Remove all mode bits
ORR     r4,r4,#Mode_SVC_32
MSR     CPSR_c, r4                          ; Reset supervisor mode
MRS     r4,SPSR                             ; Read stored status reg
BIC     r4,r4,r6                            ; Enable exception
MSR     SPSR_cxsf,r4                        ; Set stored status reg

SetIRQBit
MOV     r5, #IBit
MOV     r6, #SET_BIT
B       SetClearFIQIRQ

ClearIRQBit
MOV     r5, #IBit
MOV     r6, #CLEAR_BIT
B       SetClearFIQIRQ

SetClearFIQIRQ
MRS     r7, SPSR                            ; Read saved PSR
BIC     r7, r7, r5                          ; Clear FIQ or IRQ bit of saved PSR
CMP     r6, #CLEAR_BIT                      ; If equal to #CLEAR_BIT
BEQ     STORE_SPSR                          ; Do not set any bit of saved PSR
ORR     r7, r7, r5                          ; Set FIQ or IRQ bit of saved PSR

ExitSwi
        LDMFD   sp!, {r4-r14}
        CPSIE   I                                    ; Enable IRQ
        MOVS    PC, r14                              ; Return from SWI
```

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		