Versal ACAP

Technical Reference Manual

AM011 (v1.0) July 16, 2020

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
07/16/2020 Version 1.0	
Initial release.	N/A

Table of Contents

Introduction

This section includes these chapters:

Introduction to Versal ACAP

Versal™ adaptive compute acceleration platforms (ACAPs) combine Scalar Engines, Adaptable Engines, and Intelligent Engines with leading-edge memory and interfacing technologies to deliver powerful heterogeneous acceleration for any application. Most importantly, Versal ACAP hardware and software are targeted for programming and optimization by data scientists and software and hardware developers. Versal ACAPs are enabled by a host of tools, software, libraries, IP, middleware, and frameworks to enable all industry-standard design flows.

Built on the TSMC 7 nm FinFET process technology, the Versal portfolio is the first platform to combine software programmability and domain-specific hardware acceleration with the adaptability necessary to meet today's rapid pace of innovation. The portfolio includes six series of devices uniquely architected to deliver scalability and AI inference capabilities for a host of applications across different markets—from cloud—to networking—to wireless communications—to edge computing and endpoints.

The Versal architecture combines different engine types with a wealth of connectivity and communication capability and a network on chip (NoC) to enable seamless memory-mapped access to the full height and width of the device. Intelligent Engines are SIMD VLIW AI Engines for adaptive inference and advanced signal processing compute, and DSP Engines for fixed point, floating point, and complex MAC operations. Adaptable Engines are a combination of programmable logic blocks and memory, architected for high-compute density. Scalar Engines, including Arm® Cortex™-A72 and Cortex-R5F processors, allow for intensive compute tasks.

The Versal AI Core series delivers breakthrough AI inference acceleration with AI Engines that deliver over 100x greater compute performance than current server-class of CPUs. This series is designed for a breadth of applications, including cloud for dynamic workloads and network for massive bandwidth, all while delivering advanced safety and security features. AI and data scientists, as well as software and hardware developers, can all take advantage of the high-compute density to accelerate the performance of any application.

The Versal Prime series is the foundation and the mid-range of the Versal platform, serving the broadest range of uses across multiple markets. These applications include 100G to 200G networking equipment, network and storage acceleration in the Data Center, communications test equipment, broadcast, and aerospace & defense. The series integrates mainstream 58G transceivers and optimized I/O and DDR connectivity, achieving low-latency acceleration and performance across diverse workloads.

The Versal Premium series provides breakthrough heterogeneous integration, very high-performance compute, connectivity, and security in an adaptable platform with a minimized power and area footprint. The series is designed to exceed the demands of high-bandwidth, compute-intensive applications in wired communications, data center, test & measurement, and other applications. Versal Premium series ACAPs include 112G PAM4 transceivers and integrated blocks for 600G Ethernet, 600G Interlaken, PCI Express® Gen5, and high-speed cryptography.

The Versal architecture documentation suite is available at: https://www.xilinx.com/versal.

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

- **System and Solution Planning:** Identifying the components, performance, I/O, and data transfer requirements at a system level. Includes application mapping for the solution to PS, PL, and AI Engine.

  The technical reference manual (TRM) describes the overall hardware architecture of the Versal™ ACAP and provides details on the blocks in the platform management controller (PMC) and in the processing system (PS).

  - High-level chip description: Section II: Hardware Architecture
  - PS architecture: Processing System Overview
  - PMC architecture: Platform Management Controller
  - AMBA® Interconnect: Section VIII: Interconnect
  - I/O connectivity architecture (buffers and transceivers): I/O Connectivity
  - Clock, reset, and power architectures and controls: Section XIV: Clocks, Resets, and Power

  There a several device families with different options. The device-specific options are listed in the *Versal Architecture and Product Data Sheet: Overview* (DS950).

- **Embedded Software Development:** Creating the software platform from the hardware platform and developing the application code using the embedded CPU. Also covers XRT and Graph APIs.

  Embedded software can run on one or both of the Arm® Cortex™ scalar engines in the PS:

  - Real-time Processing Unit (dual-core Cortex-R5F)
  - Application Processing Unit (dual-core Cortex-A72)

  The PMC and PS functional units require device drivers as part of the embedded software stack. Several TRM reference sections primarily focus on content for device driver development. Major peripherals are listed in the following sections:

- Section VII: Platform Processor, Configuration, and Security Units
- Section XII: I/O Peripheral Controllers
- Section XIII: Flash Memory Controllers

Additional TRM sections and chapters describe the interconnect, timers, counters, clocks, resets, and power.

- **Board System Design:** Designing a PCB through schematics and board layout. Also involves power, thermal, and signal integrity considerations.

  The TRM includes some important information to help with board design planning and development:

  - Boot device interfaces: Boot Modes
  - Pin planning for I/O peripherals: Multiplexed I/O
  - Power controls: Power Diagram
  - JTAG interface: JTAG and Boundary-Scan

XILINX

Chapter 3

TRM Organization

The TRM provides a high-level architecture overview of the Versal™ ACAP. After this overview, the TRM provides details on the platform management controller (PMC) boot process, address maps, signals, and functionality of each block in the PMC and processing system (PS). These sections can be used as guides to the TRM content:

- TRM Outline
- PMC Hardware Perspective
- PS Hardware Perspective
- Embedded Software Perspective

TRM Outline

The TRM is divided into sections that provide Versal ACAP hardware architecture information on the PS and the PMC. The TRM also provides references to companion documentation that complement the TRM and provide detailed information outside of the PS and the PMC. The key TRM sections are described in this section.

Device and Document Overview

Section I: Introduction provides an introduction to the Versal ACAP.

The TRM technical content begins with Section II: Hardware Architecture. This is a hardware-centric section that covers the entire Versal ACAP. This section includes links to other parts of the TRM and to companion documents that include extensive technical information about the Versal ACAP.

Platform Boot, Control and Status Functionality

The reset response, boot flow, and run-time services are provided by the PMC. The start-up activities of the PMC are described in Section III: Platform Boot, Control, and Status. The chapters in this section describe how the Versal device comes up after a reset and how to manage the platform during normal device operation.

Global Address Maps and Signals

The TRM includes two reference sections for address maps, control register summaries, and tables that list the Versal ACAP's signals, interfaces, and pins. These sections include device-wide content.

- Section IV: Address Maps and Programming Interfaces
- Section V: Signals, Interfaces, and Pins

PMC and PS Functional Units

The remainder of the TRM includes multiple technical reference sections that illustrate the detailed architectures and describe the functional models of each block in the PMC and PS.

- Section VI: Engines
- Section VII: Platform Processor, Configuration, and Security Units
- Section VIII: Interconnect
- Section IX: Interrupts and Errors
- Section X: Timers, Counters, and RTC
- Section XI: Memory
- Section XII: I/O Peripheral Controllers
- Section XIII: Flash Memory Controllers
- Section XIV: Clocks, Resets, and Power
- Section XV: Test and Debug

The *Versal ACAP System Software Developers Guide* (UG1304) describes the boot sequences after the PMC has begun to fetch the first part of the boot image from the boot device.

PMC Hardware Perspective

The TRM includes several PMC-related content areas:

- Section II, Hardware Architecture
  - Chapter: PMC hardware functionality and architecture are described in the Platform Management Controller chapter.
- Section III, PMC Processes
  - Section: The progression of activity from reset to device boot to platform management is described in the dedicated Section III: Platform Boot, Control, and Status section.

- Section VII, PMC Functional Units
  - Section: The PMC-centric functional units are located in the Section VII: Platform Processor, Configuration, and Security Units section.
- Section XII, I/O Peripheral Controllers
  - Chapters: Many of the Section XII: I/O Peripheral Controllers can be a boot device. They are listed in Boot Modes.
- Section XIII Flash Memory Controllers
  - All of the Section XIII: Flash Memory Controllers can be used as a boot device.
- System-level Functionality Sections
  - Section XV: Test and Debug.
  - Section XIV: Clocks, Resets, and Power.

PLM Software

The platform loader and manager (PLM) software is downloaded by the RCU BootROM code. This software can be generated by the Vivado® tools to configure and manage the system. The PLM runs on the MicroBlaze™-based PMC platform processing unit (PPU). The TRM provides programming models for the functionality in the PMC.

PSM Firmware

The PSM controls power management features for the PS, which includes the blocks in the LPD and FPD subsystems. The processing system manager (PSM) firmware is downloaded by the PLM. The TRM provides programming models for the power control features in the PS, which includes power islands, memory chip enables, isolation, and APU sleep/wake events.

PS Hardware Perspective

The processing system encompasses the real-time processing unit (RPU) and application processing unit (APU). These are Arm® Cortex™ MPCores located in the low-power domains (LPDs) and full-power domains (FPDs), respectively. These MPCores are tied together with direct AXI interfaces between their main interconnect switches. The LPD subsystem also has AXI interfaces with the platform management controller (PMC). All processing cores have access to the network on chip (NoC). The LPD and FPD have several direct AXI interfaces with the PL.

The PS-related content includes all sections in the TRM except for the Section III: Platform Boot, Control, and Status section.

RPU Software

The RPU is based on a dual-core Arm Cortex-R5F MPCore processor. The TRM describes the architecture and the programming model for the controllers and other functional units. References include descriptions of the memory-mapped control and status registers. The LPD address map includes the RPU and is shown in LPD Address Map.

APU Software

The application processing unit (APU) is based on a dual-core Arm Cortex-A72 MPCore processor. The TRM describes the architecture and the programming model for the controllers and other functional units. References include descriptions of the memory-mapped control and status registers. The FPD address map includes the APU and is shown in FPD Address Map.

Embedded Software Perspective

The TRM is primarily written for embedded software and device driver development. The TRM, together with the register reference manual, describe the details of what can be configured, controlled, and monitored in the PS and PMC.

The PMC and PS include approximately 100 register sets or modules. Most functional units include one register set. Some functional units have multiple register sets (e.g., CCI, USB). Other register sets are associated with multiple functional units, which includes the system-level control registers (SLCRs). All register sets are accessed by software using 32-bit read and write transactions. There are two types of hardware register modules:

- PMC and PS (and some other): 32-bit APB read/write interface (these are included in the *Versal ACAP Register Reference Manual* (AM012))
- Non-PMC/PS 32-bit NPI read/write interface with burst (these are included in several different documents)

The Section II: Hardware Architecture section of the TRM covers the whole chip from a high-level hardware perspective. The chapters in this section include links to other documents that describe functionality and provide software programming details.

Versal Device Variations

A core set of functionality is shared by all Versal™ devices. Also, certain devices targeted towards specific applications can contain optional functions. Both core and optional functionality are listed in this section.

The TRM describes the functionality of production silicon.

All Devices

- Platform management controller (PMC) is standard in all devices
- Processing system (PS) is standard in all devices
- Network on chip (NoC) interconnect (two or more horizontal and vertical channels)
- DDR memory controller (one or more)
- Programmable logic (PL) (various sizes and compositions)
- PL I/O interfacing (various counts)
- GTY transceivers (various counts)

Optional Functionality

The following items are not in every device.

Note: The TRM explains functionality with the assumption that every block is present in the system. The device variations are defined in the *Versal Architecture and Product Data Sheet: Overview* (DS950).

- AI Engine arrays
- PL integrated hardware (e.g., multirate Ethernet MAC)
- CCIX PCIe® module (CPM)
- 4 MB accelerator RAM (XRAM)
- GTM transceivers (with various counts)

Device Matrix

The following table is provided as a convenience. For details, additional counts of each option, and latest information, see the *Versal Architecture and Product Data Sheet: Overview* (DS950).

Table 1: **Device Options Matrix**

Integrated Option	Series Devices		
	AI Core	Prime	Premium
AI Engine	Yes	~	~
MRMAC (100G)	1 to 4	1 to 6	2 to 6
Ethernet (600G)	~	~	1 to 8
Interlaken (600G)	~	~	0 to 3
HSC Crypto (400G)	~	~	1 to 5
PCIe® CPM	~	Yes	Yes
4 MB accelerator RAM	Some devices	~	~
GTM transceivers	~	0 to 40	10 to 70

Comparison to Previous Generation Xilinx Devices List

The migration of functionality from a previous generation Xilinx® device is captured on a per block basis. There are several "Comparison to Previous Generation Xilinx Devices" topics throughout the TRM, which are summarized below.

Note: For additional device comparison information, see *Versal ACAP Design Guide* (UG1273).

- Platform Management Controller (PMC)
- Real-time Processing Unit (RPU)
- Application Processing Unit (APU)
- LPD DMA
- Platform Processing Unit (PPU)
- Interconnect
- System Master IDs
- Inter-processor Interrupts (IPI)
- System Watchdog Timer
- On-chip Memory
- Accelerator RAM (XRAM)
- CAN FD
- Gigabit Ethernet MAC (GEM)
- General-purpose I/O (GPIO)
- I2C
- SPI
- UART SBSA
- USB 2.0
- Octal SPI
- Quad SPI

- SD/eMMC
- Resets

IP Versions

The following table lists the IP versions used in the Versal™ ACAP.

Table 2: **Versal ACAP IP Versions**

Functional Unit	Location	Vendor	Version
Application Processing Unit (APU)			
Processor cores, Cortex™-A72	FPD	Arm®	r0p3-00rel0
Floating point unit, VFPv4	FPD	Arm	
NEON execution unit, v8-A architecture	FPD	Arm	
Cryptography extension	FPD	Arm	r0p2-00rel0
GIC interrupt controller (GIC-500)	FPD	Arm	r1p1-00rel0
Real-time Processing Unit (RPU)			
Processor cores, Cortex-R5F	LPD	Arm	AT570-r1p3-00rel0
GIC interrupt controller (PL-390)	LPD	Arm	r0p0-00rel2
LPD DMA			
Descriptor-driven DMA	LPD	Xilinx®	
Platform Processors, Configuration, and Security Units			
AES	PMC	Athena	ro-2017-12-12
RSA/ECDSA	PMC	IP cores	5X-409603203 r2.0_12_20_2016
True random number generator, TRNG	PMC	IP cores	MP32 core r1.5
Interconnect			
System memory management unit (SMMU-500)	FPD	Arm	TCU is r2p4 TBU is r2p1
Cache Coherent Interconnect (CCI-500)	FPD	Arm	PL422-r1p0-00rel0
AXI interconnect switches (NIC-400)	PMC, LPD, FPD	Arm	r0p2
Timers and Counters			
System counter	LPD	Arm	
Triple timer counters (TTC)	LPD	Cadence	T-CS-PE-0005-100
System watchdog timer (SWDT)	LPD, FPD	Xilinx	0.8
I/O Peripherals			
CAN FD controller	LPD	Xilinx	v2.0
GEM Ethernet MAC controller, GXL and RGMII	LPD	Cadence	r1p12
I2C controller	PMC, LPD	Cadence	dcw0701_R114_f0100_final
SPI controller	LPD	Cadence	r112

Table 2: **Versal ACAP IP Versions** *(cont'd)*

Functional Unit	Location	Vendor	Version
UART SBSA controller	LPD	Arm	r1p5-00rel1
USB 2.0 controller	LPD	Synopsys	USB3 DRD 3.30a
Flash Memory Controllers			
OSPI flash memory controller	PMC	Cadence	DNV3100_R003_F004
QSPI flash memory controller	PMC		
SD/eMMC controller	PMC	Arasan	1p48_140929
Test and Debug			
CoreSight™ debug (SoC-400)	~	Arm	r3p2-00rel1
CoreSight embedded logic analyzer (ELA-500)	FPD, CPM	Arm	r2p2-00rel0
CoreSight LAK-500 A/I	FPD	Arm	r1p0-00rel0
CoreSight STM-500	FPD	Arm	r0p1-00rel1
Aurora	LPD	Xilinx	1.0

XILINX

Section II

Hardware Architecture

This section includes these chapters:

Device Overview

The Versal™ ACAP architecture is a complex device with a rich set of integrated hardware and many user programmable design options for many system-level solutions. It incorporates programmable logic (PL) and a processing system (PS) that need to be brought up and configured through a coherent flow. The PL and PS sections of the device each have many components that can electively be used as needed and, if included, they are configured and initialized to accommodate different functional and power requirements demanded by the platform solution.

Additionally, the system is monitored during its run time to detect errors and provide the necessary means to address the errors as a part of the security, reliability, and safety requirements. The configuration, bring-up, and general platform management tasks include reset, clocking, power management, and system monitoring. This is achieved by the platform management controller (PMC) that exists on every device. The PMC also provides a unified interface for the cohesive debug and trace capture on the entire device including the PS, PL, and other components that interact with them.

The integrated hardware is listed in the following table with their power domain. The integrated hardware is in all devices except where noted as optional. The options, size, and number of integrated units are defined in the product data sheet *Versal Architecture and Product Data Sheet: Overview* (DS950).

Table 3: **Device Integrated Hardware and Power Domains**

Integrated Hardware		Power Domain	Options	Notes
Name	Acronym			
Platform management controller	PMC	PMC		
Processing system	PS	~		
Real-time processing unit	RPU	LPD		
Application processing unit	APU	FPD		
SoC components				
Network on chip	NoC	SPD		Scales with die size
DDR memory controller	DDRMC		Number	One to four controllers
Programmable logic	PL	PL	Size	
AI Engine (tiled)	AIE	PL	Yes	
4 MB accelerator RAM	XRAM	PL	Yes	Mutually exclusive options
Cache Coherent Interconnect for Accelerators (CCIX) PCIe® Module	CPM	LPD	CPM4	

Table 3: **Device Integrated Hardware and Power Domains** *(cont'd)*

Integrated Hardware		Power Domain	Options	Notes
Name	Acronym			
Multirate Ethernet MAC	MRMAC	PL		

The power domains are color-coded in Figure 1: System-level Interconnect Architecture.

The TRM provides a high-level device perspective with details for the PMC and PS functionality. The functionality of the other integrated hardware is referenced in the Integrated Hardware section.

Platform Management Controller

The system starts up and is controlled by the PMC. The ROM code unit (RCU) boots the hardware and loads the initial platform loader and manager (PLM) software into the PPU processor. The PMC is in its own power domain. The boot sequences and platform control functions are described in Section III: Platform Boot, Control, and Status.

The integrated hardware is configured with programmable device image (PDI) files. The PDIs are composed of configuration data object (CDO) files and other elements that are processed by the PLM software. This includes configuring the PS, NoC, DDR, and others. These files are described in the *Versal ACAP System Software Developers Guide* (UG1304).

Processing System (RPU and APU)

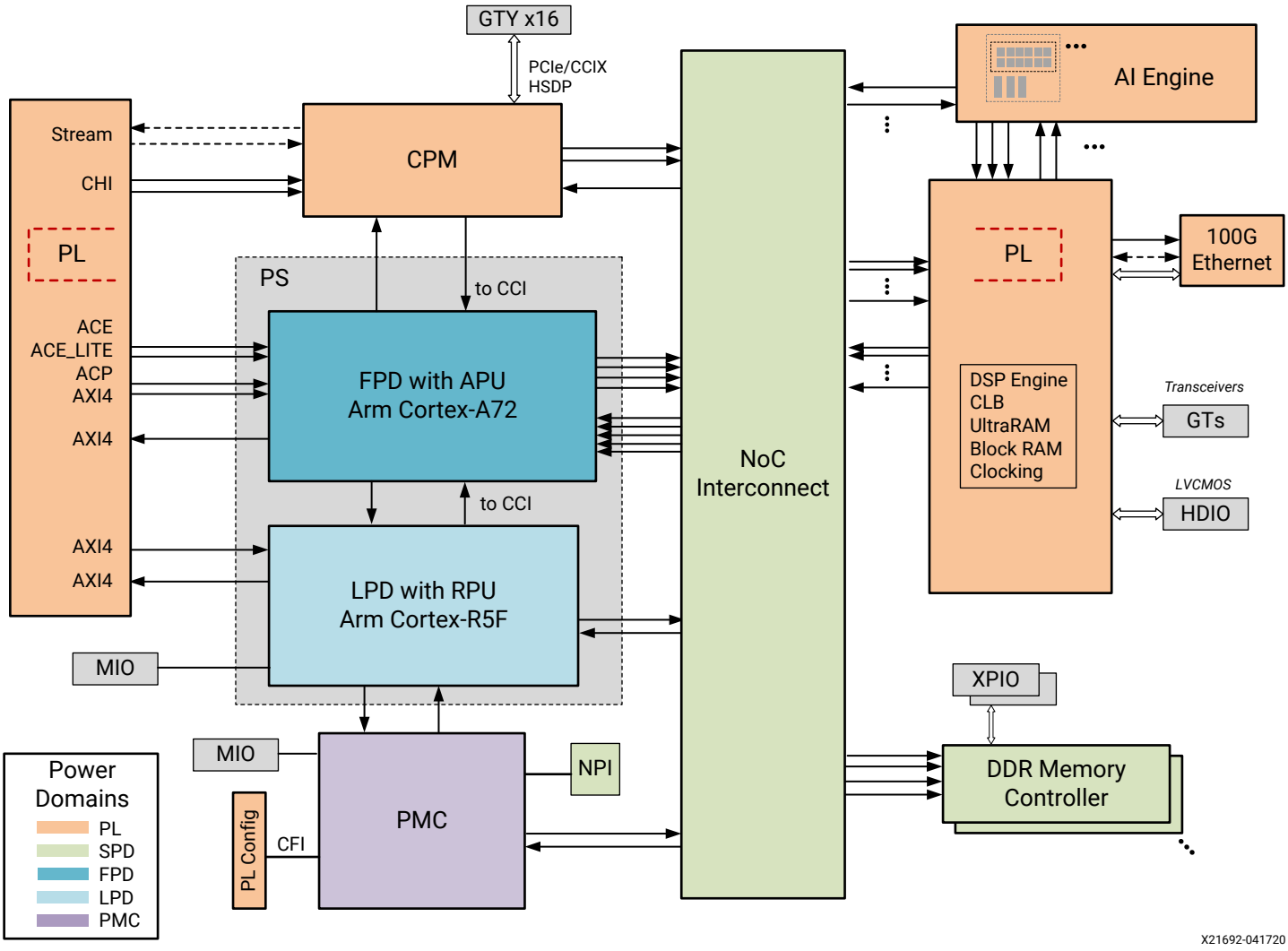
The PS includes two Arm® Cortex™-R5F RPUs and two Arm® Cortex™-A72 APUs. These provide programmers with two different operating environments.

The RPU is in the low-power domain (LPD) and the APU is in the full-power domain (FPD).

System-Level Interconnect Diagram

The system-level interconnect is featured in the following figure.

Figure 1: **System-level Interconnect Architecture**



Integrated Hardware

The Versal ACAP device has several subsystems. The details of the PMC and PS are included in the TRM. The details of the other subsystems are included in the referenced documents listed in this section.

Platform Management Controller

The PMC operations are divided into four phases beginning with hardware resets that start or restart the ROM code unit (RCU). After reset, the RCU executes the BootROM to configure the system to access the boot device and process the boot header. The platform loader and manager (PLM) is loaded from the boot device and into the platform processing unit (PPU). When this is completed, the PLM takes control of the system for further configuration and, optionally, loads system software for the RPU, APU, and processing system manager (PSM) controller.

During normal run time, the PLM works with other parts of the PMC and the PSM to monitor and respond to system activities and events. The PMC is in all devices and is required in all operating modes.

Processing System

The PS includes real-time processing unit (RPU) and application processing unit (APU). The RPU and APU can work together or independently. The PS includes the low-power and full-power domains (LPD and FPD).

The RPU is based on the Arm Cortex-R5F multicore processor with floating point unit. The RPU provides deterministic execution times for real-time processing needs by accessing its tightly coupled memories (TCM), on-chip memory (OCM), and, when available (not shown in the System Perspective figure), the 4 MB accelerator RAM (XRAM). The RPU processor can be configured in a dual-processor performance mode or in a lock-step safety mode.

The APU is based on the Arm Cortex-A72 multicore, superscalar processor and is based in the FPD with the system memory management unit (SMMU) and the Cache Coherent Interconnect (CCI). The SMMU and CCI enable the APU and other processors to be assigned system memory address space, as needed, for the operating system kernels. The SMMU translates addresses from the system masters. When a page table look-up fails, an interrupt fault is generated and the transaction is terminated. The CCI enables transaction requests from system masters to be two-way or I/O coherent with the APU L2 cache.

The power modes of the PS are controlled by the PSM in the LPD.

The Arm documentation is listed in Arm References. Programming and usage information about the RPU and APU are located in the *Versal ACAP System Software Developers Guide* (UG1304).

PCIe Cache Coherent Interconnect for Accelerators

The Cache Coherent Interconnect for Accelerators (CCIX) PCIe® Module (CPM) enables PL-instantiated masters to share the CPM L2-cache with external system masters that are attached to the PCIe® controller using the CCIX protocol.

For more information, see the *Versal ACAP CPM CCIX Architecture Manual* (AM016).

Programmable Logic

The PL provides several important functions. The array includes DSP engines, adaptable building blocks, and RAM arrays. These are configured together to create almost any type of hardware functionality. This provides scalable, adaptable functionality with connections to NoC, high-speed I/O, integrated hardware, and the PS.

Debug functions can be instantiated in the PL that include connections to the Arm CoreSight™ debug hardware located in the PS and PMC to collect data.

AI Engine

The AI Engine includes both engine and array interface tiles. AI Engine tile contain a high-performance vector based single instruction multiple data (SIMD) processor with integrated memory and interconnect ports for streaming, configuration, and debug. The array interface tile connects the AI Engine to the NoC and the PL.

The AI Engine is integrated into the Versal ACAP AI Core Series. For an introduction, see *Xilinx AI Engine and Their Applications* (WP506). The AI Engine hardware descriptions are in the *Versal ACAP AI Engine Architecture Manual* (AM009).

Network on Chip

The network on chip (NoC) interconnect spans the entire device to enable most any master to potentially reach most any slave. The global address maps are based on the NoC interconnect.

The configurable NoC is an AXI4-based network to route high-bandwidth, real-time, and/or low-latency connections. The NoC extends in both horizontal and vertical directions to the edges of the device. The multichannel structures provide several options for routing and isolating traffic. The NoC is a full blocking crossbar between memory controllers, programmable logic, processing system, AI Engines, and platform management controller. Examples of NoC connections include:

- DDR memory controller access for all
- PL to PL connections
- Memory mapped access to the AI Engine array
- Connecting between PS and PL

In devices built using stacked silicon interconnect technology, the vertical NoC columns connect between adjacent super logic regions (SLRs), which allows device configuration data to travel between master and slave SLRs.

The NoC functionality is described in the *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* (PG313).

DDR Memory

The integrated DDR memory controller (DDRMC) supports both the DDR4 and LPDDR4 memory interfaces. It can be configured with a 32-bit or 64-bit DRAM interface with or without ECC. Some devices include multiple DDR memory controllers. The controller has four NoC interface ports to handle multiple streams of traffic and supports quality of service (QoS) classes to ensure appropriate prioritization of the memory requests inside the controller.

Each DDR controller also includes a Xilinx memory protection unit (XMPU) to only allow authorized accesses by specific masters with proper security and read/write attributes.

For more information on the integrated DDRMC, see the *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* (PG313).

Embedded Memories

There are many RAM memory arrays embedded in the Versal device. Most are protected by parity or ECC to improve safety.

The 256 KB on-chip memory (OCM) is accessible from the LPD OCM AXI switch. The 4 MB accelerator RAM (XRAM), when present in the device, includes 1 port from the OCM switch and 3 ports from the PL. Each port includes a Xilinx memory protection units (XMPUs).

The PMC includes the 384 KB PPU and 128 KB PMC RAMs. The RPU includes 256 KB of tightly coupled memory (TCM) in six banks and are configured in two groups for high-performance dual processor mode, and grouped together for high-safety, lock-step mode. All of these memories are ECC protected.

The PL includes block RAM and UltraRAM scattered throughout the array. These include ECC bits for 64 byte data segments.

The last group of RAMs are scattered in various blocks for buffers, FIFOs, and caches.

- 256 KB OCM
- 4 MB XRAM
- 384 KB PPU RAM
- 128 KB PMC RAM
- 4 KB block RAM in PL
- 32 KB UltraRAM in PL
- Miscellaneous buffers, FIFOs, queues, and caches

The embedded memories are summarized with additional documentation references in the Section XI: Memory section of the TRM.

AMBA Interconnect

The PS processors and the PMC each have local AXI main switches, which have master and slave units connected to the NoC. The PS processors and the PMC also have AXI interfaces that interconnect directly with each other. PMC and PS peripherals are controlled by AXI and APB programming interfaces.

CPM Interconnect

The CPM interconnect provides memory coherency for processors in the PL and externally attached processors on the PCIe bus interfaces. The CPM interconnect includes a 1 MB CPM L2-cache.

Programming Interfaces

The interconnect includes the NPI and APB register programming interfaces. These interconnects provide 32-bit read/write access to the registers. The NPI interface supports burst transactions to reduce the register programming time. Device configuration is also done using configuration frames; these are embedded in files and written to the configuration frame unit (CFU) in the PMC. These are explained in Programming Interfaces chapter.

I/O Connectivity

The Versal ACAP includes many different types of I/O pins. Each pin has a dedicated I/O buffer with characteristics that are often programmable. The functionality of a pin can be dedicated to a specific function or have a flexible assignment. See I/O Connectivity.

Integrated Peripherals

The Versal™ device can include the following integrated peripherals. The existence of a peripheral and the number of cores in a device are described in the *Versal Architecture and Product Data Sheet: Overview* (DS950).

- Multirate Ethernet MAC
- 600G Channelized Multirate Ethernet
- 600G Interlaken with FEC
- 400G High-Speed Crypto Engine

Multirate Ethernet MAC

The multirate Ethernet MAC (MRMAC) provides high-performance, low latency Ethernet ports supporting a wide range of customization and statistics gathering. Supported configurations are 1 x 100GE, 2 x 50GE, 1 x 40GE, 4 x 25GE, and 4 x 10GE.

The MRMAC supports the following FECs defined and required by IEEE standards:

- Clause 91 RS(528, 514) KR4 FEC, for 25/50/100GE NRZ support
- Clause 91 RS(544, 514) KP4 FEC for 50/100GE PAM4 support
- Clause 74 FEC, for 10/25/40/50GE low-latency support

The MRMAC has a rich set of bypass modes to enable access to FEC-only mode (for custom protocols) and FEC+PCS (for protocol testers).

The MRMAC also supports a new high-precision timestamping feature to enable sub-nanosecond accuracy on IEEE Std 1588 timestamps. This provides hardware support for new IEEE Std 1588-based time-sensitive networks (TSN), as well as the next generation Ethernet-based wireless fronthaul protocol (eCPRI).

The MRMAC controller is described in the *Versal Devices Integrated 100G Multirate Ethernet MAC (MRMAC) LogiCORE IP Product Guide* (PG314).

600G Channelized Multirate Ethernet

The 600G channelized multirate Ethernet subsystem (DCMAC) provides up to 600G of Ethernet bandwidth that can be configured for various rates including 1x400GE, 3x200GE, and 6x100GE. The DCMAC handles all protocol-related functions of an Ethernet MAC, PCS, and FEC, including handshaking, synchronizing, and error checking. It also provides a segmented AXI4-Stream interface for packet data and an AXI4-Lite interface for statistics and management.

The DCMAC can be configured to include forward error correction (FEC) capability, supporting Clause 91 RS(528, 514) KR4 FEC, Clause 91 RS(544, 514) KP4 FEC, Clause 119 RS(544, 514) KP4 FEC, and Clause 134 RS(544, 514) FEC.

The DCMAC flexible interface (FLEXIF) supports several operating modes including OTN, FlexE, and PCS modes.

600G Interlaken with FEC

The integrated 600G Interlaken (ILKN) core with FEC supports channelized interfaces operating up to 600 Gb/s with built-in flow control. Each 600G Interlaken core can be configured as follows:

- 24 lanes (maximum) with 12.5G and 28.21G transceivers

- 12 lanes (maximum) with 56.42G transceivers

The flexible AXI4-Stream user interface is configurable in width from 2048b to 512b. Pairs of lanes share 100G RS(544, 514) FEC and can support FEC-only mode.

400G High-Speed Crypto Engine

The 400G high-speed cryptography (HSC) engine implements an AES-GCM-256/128 engine that provides up to 400 Gb/s of bulk encryption capability on up to 40 channels that can be connected to the DCMAC. Each HSC engine supports both MACSec and IPSec at up to 400 Gb/s configurable as 1x400G, 2x200G, or 4x100G channels with up to 128 source addresses (SA) per 100G.

Grouped Functionality

There are several situations where functionality is grouped together for various purposes. This section highlights some of these groupings.

Processing System

The PS includes the Arm Cortex-A72 application processor unit (APU) and the Cortex-R5F real-time processing unit (RPU). Both of these are MPCore units with flexible configurations for software applications. Their usage is described in the *Versal ACAP System Software Developers Guide* (UG1304).

The PS is also known as the combination of the LPD and FPD power domains.

Control, Interfaces, and Processing System

The control, interface, and processing system (CIPS) IP core is a large group that consists of the PMC, PS, and CPM. These are configured using the Vivado® Design Suite CIPS wizard tool flow.

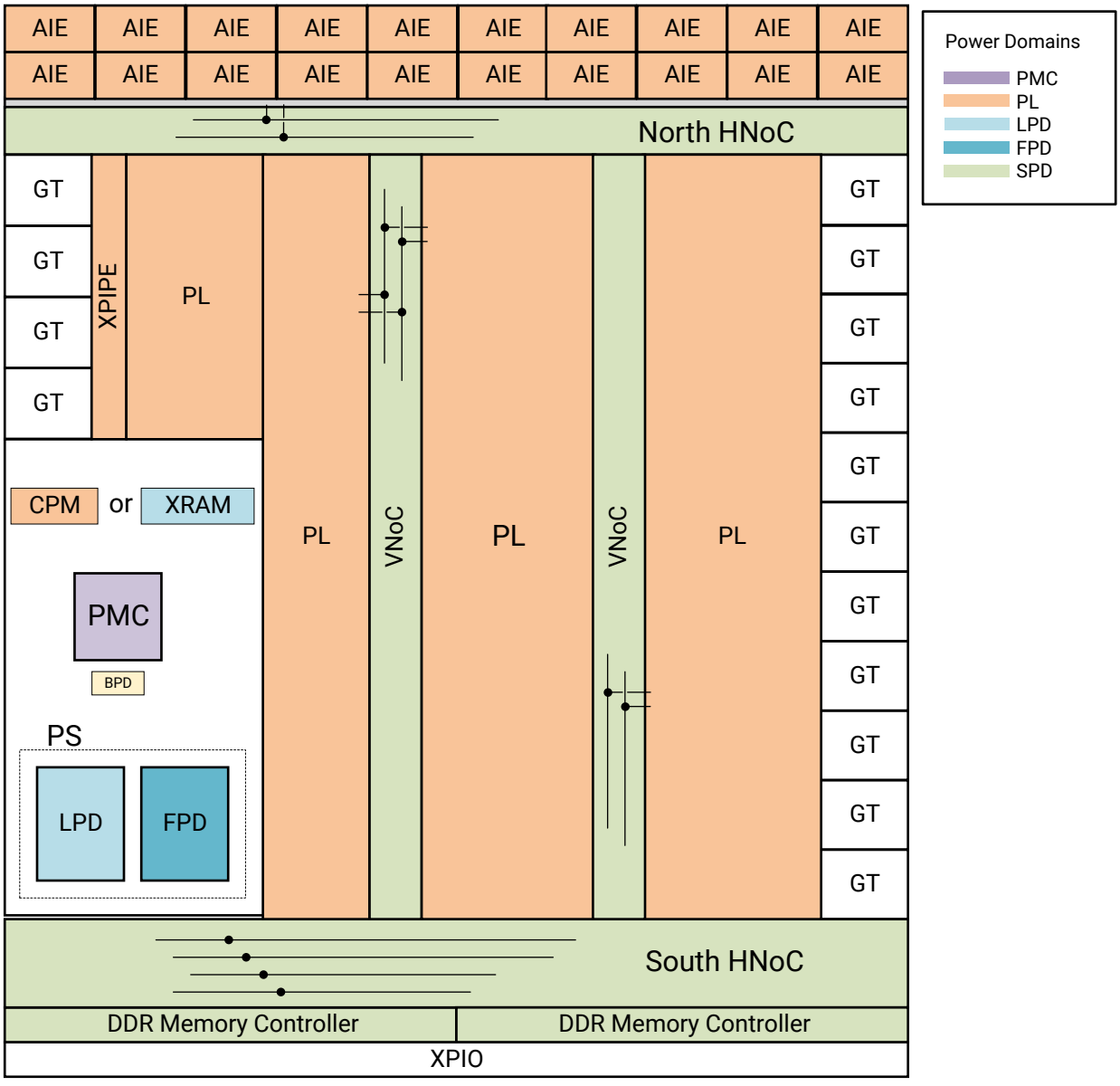
Physical Layout

In the physical layout, the NoC interconnect and the PL predominate. The horizontal and vertical NoC structures form a grid in the PL with gigabit I/O transceivers placed along the left and right edges of the perimeter. The entire PMC and PS subsystems are located in the lower left corner of the device. The DDR memory controllers are located along the southern edge of the device by the XPIO banks. The AI Engine array, when present, is along the northern end of the device with access to the NoC and the PL.

***Note*:** The Versal™ ACAP has a large number of devices with many scalable PL layouts and I/O structures. The following figure is only a representation and does not reflect a specific device. There are additional HNoC and VNoC structures.

***Note*:** Not all features are included on a given device. For more information, see the Versal ACAP data sheets.

Figure 2: **Physical Layout Example Representation**



X21771-060720

XILINX

Chapter 8

Processing System Overview

The TRM provides the details for the PMC and PS. The PS provides both real-time and application multicore processors as a major computational and control center. The AI Engine and adaptable components in the programmable logic (PL) can greatly expand the device performance. These are described in other documents. See Integrated Hardware for a description of all parts of the device.

All PMC and PS masters connect to the NoC interconnect for access to the DDR memory controller and other parts of the device.

The PMC and PS are included in the control, interfaces, and processing system (CIPS) IP, which is configured by the CIPS wizard in the Vivado® tools.

The PS includes two multi-core processors:

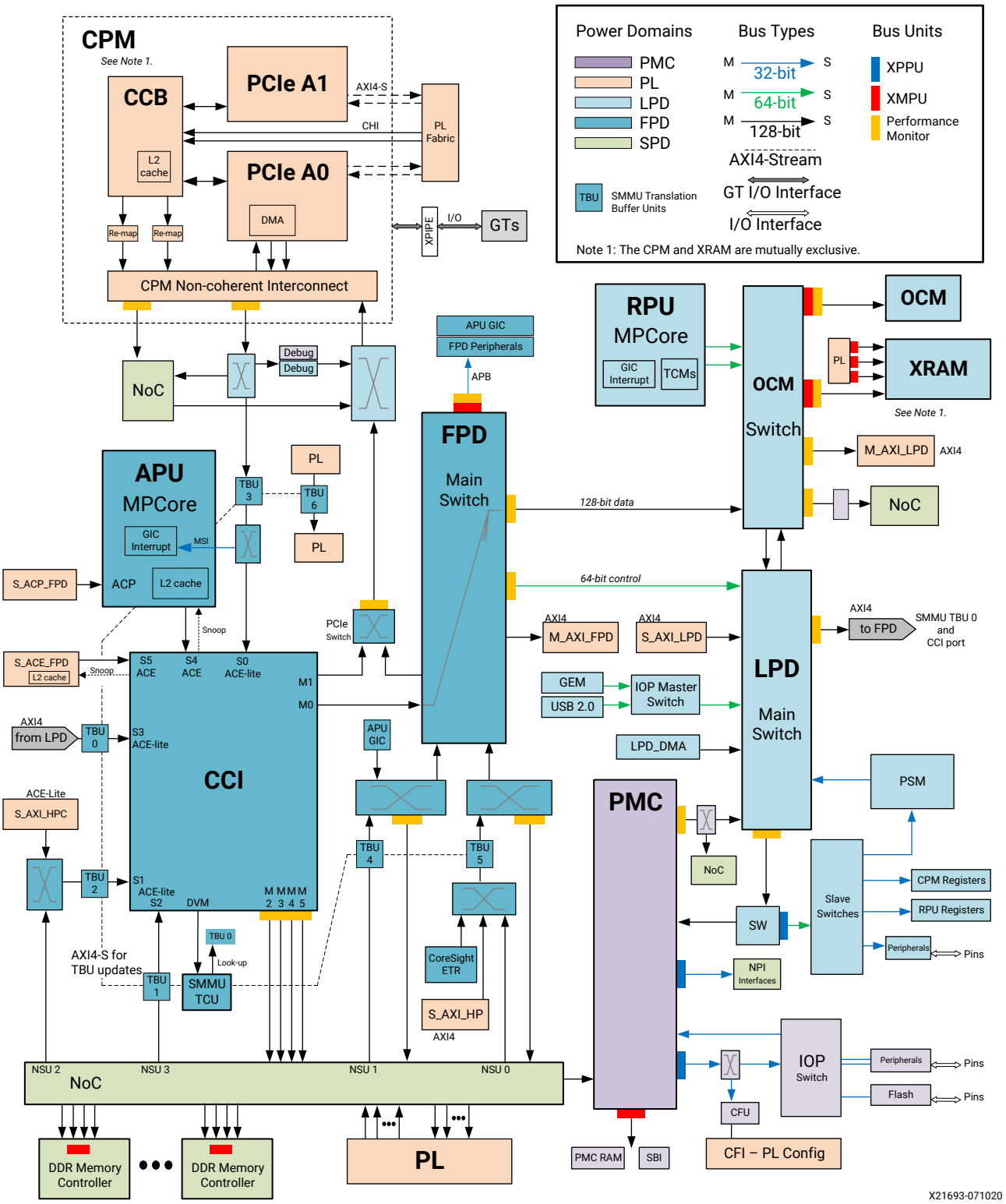
- APU: Arm-based Cortex™-A72 processor in the FPD
- RPU: Arm-based Cortex-R5F processor in the LPD

PS Interconnect Diagram

The following figure shows the system interconnect details for the LPD and FPD. The PMC and optional CPM are also included. Additional block diagrams include:

- LPD Interconnect
- APU Interconnect
- For CPM, see *Versal ACAP CPM CCIX Architecture Manual* (AM016)

Figure 3: **PS Interconnect Diagram**



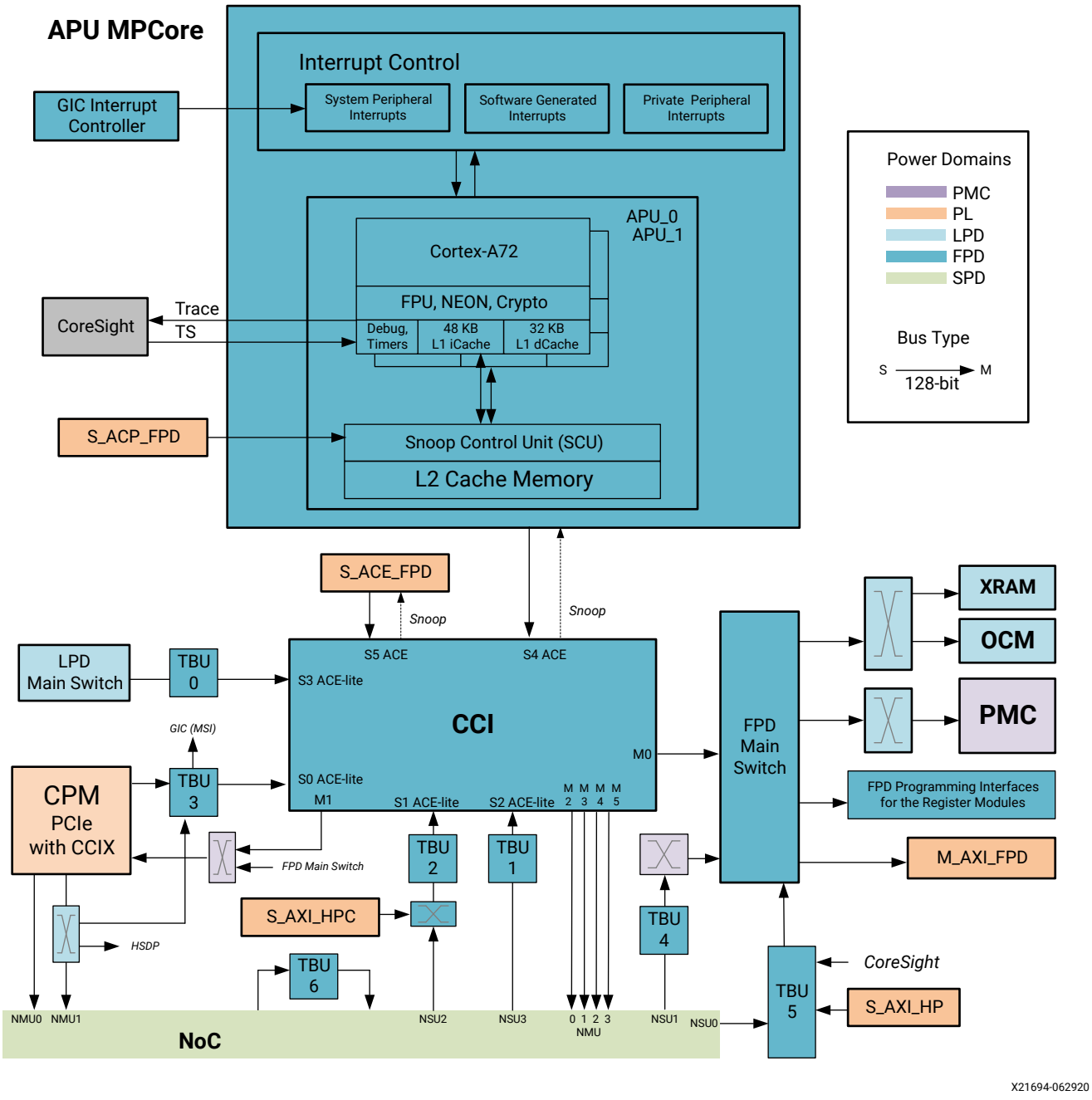
Full-power Domain

The PS full-power domain (FPD) includes the APU, Cache Coherent Interconnect (CCI) with access to the APU L2-cache, and connections to the CCIX PCIe® module (CPM).

APU Interconnect

The APU system interconnect is shown in the following figure. This figure is a subset of the FPD interconnect. The full FPD interconnect with additional details is shown in the figure.

Figure 4: **APU Interconnect Block Diagram**



X21694-062920

APU Processor Features

The APU is a dual-core processor that is based on the Arm® superscalar, out-of-order execution Arm Cortex-A72 core.

The 64-bit Cortex-A72 cores are based on Arm-v8A architecture that supports hardware virtualization. Each A72 core includes:

- 48 KB of L1 instruction cache with parity protection
- 32 KB of L1 data cache with ECC protection
- NEON SIMD pipeline
- Floating point unit (FPU): single and double precision
- Embedded trace microcell (ETM) to support real-time debug and trace. The ETM communicates with the Arm CoreSight™ debug system.

The APU is located in the FPD. The APU is clocked independently from the FPD blocks and can be reset independently or with the FPD power domain.

APU Interrupt Controller

To manage system interrupts, the APU includes the GIC interrupt controller, which is based on the Arm GIC-500 generic interrupt controller and is compatible with the Arm GIC v3 architecture.

System Memory Management Unit

The system memory management unit (SMMU) includes these functions:

- Address translation
- Transaction security state control
- Memory protection using page table look-ups

These functions are performed with a combination of the seven translation buffer units (TBU 0 to 6). Four of these are in the path of incoming AXI interfaces to the CCI. The translation and protection tables that are cached in the TBU are updated by the SMMU translation control unit (TCU).

The functions of the SMMU are performed in the TBUs and include:

- Translates 48-bit virtual address from processors and other masters into a 44-bit physical address for DDR memory and other memory-mapped destinations
- Memory protection from unauthorized or errant accesses

Cache Coherent Interconnect

The Cache Coherent Interconnect (CCI) includes ACE ports to provide full APU L2-cache coherency to a PL master. The two ACE ports can snoop the caches of the two attached processors.

Other system masters connect to the ACE-Lite slave ports on the CCI to optionally provide I/O coherency of their transactions with the APU L2 cache (including the RPU but excluding the LPD DMA unit).

See Cache Coherent Interconnect chapter for more information.

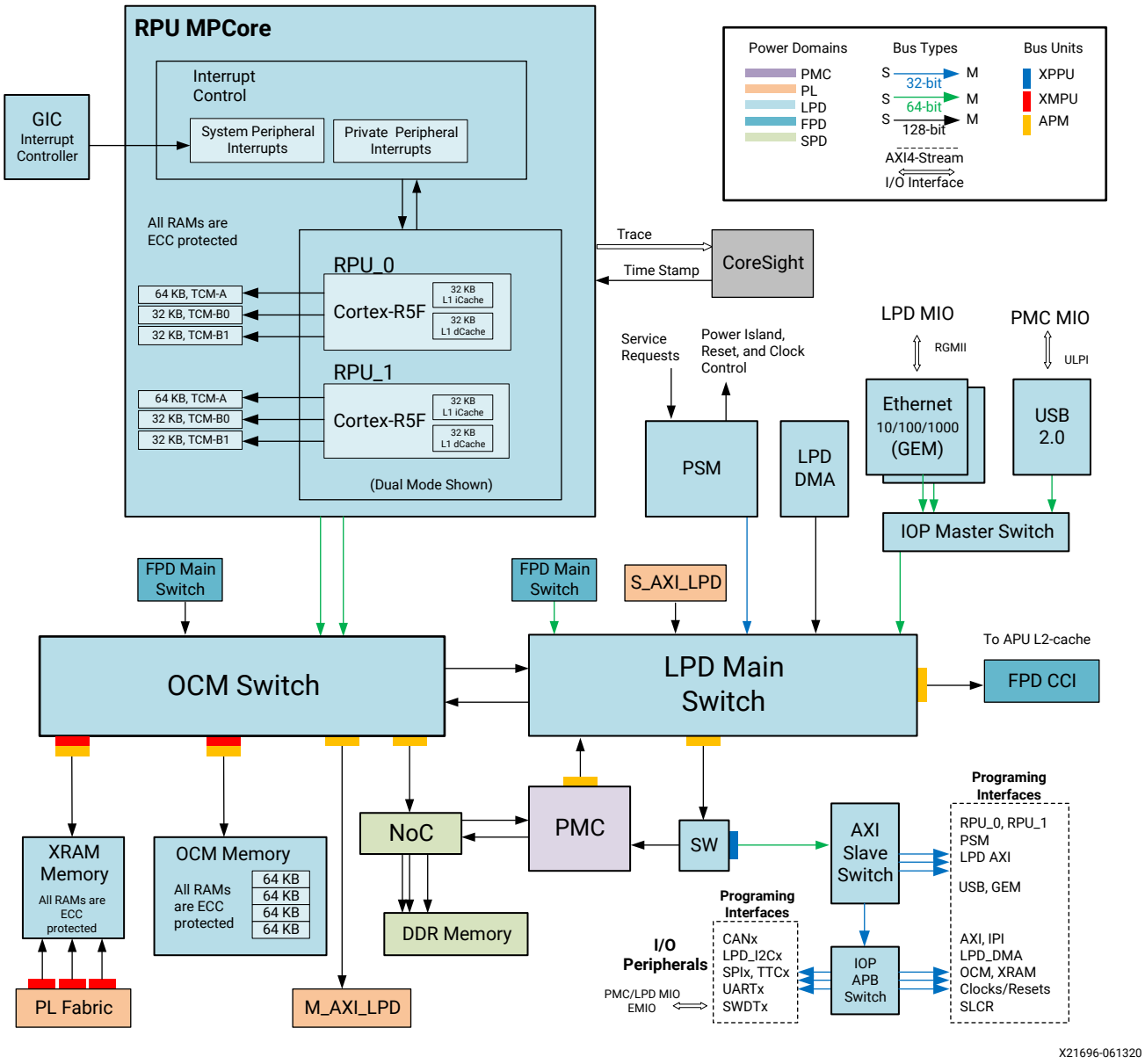
Low-power Domain

The PS low-power domain (LPD) includes the Cortex-R5F MPCore processors with their tightly coupled memories (TCM), OCM memory, I/O peripherals (IOP), and the PSM controller for PS power control. The RPU also has a direct interconnect to the accelerator RAM (XRAM, if present) that can be partitioned and shared with logic in the PL.

LPD Interconnect

The LPD includes the RPU MPCore, OCM, PSM controller, and the I/O peripherals. The following figure shows the LPD interconnect.

Figure 5: **LPD Interconnect Block Diagram**



RPU Processor

The real-time processing unit (RPU) is a dual-core processor. The RPU is based on the Arm Cortex-R5F MPCore processor with its out-of-order execution CPU that is coupled with a single/double precision floating point unit (FPU). The processor also includes an interrupt controller to receive system interrupts.

The RPU can be used for applications requiring functional safety and provides deterministic code execution times while accessing its tightly coupled memories (TCM). The RPU can be configured in a dual CPU performance mode, or in a safety lock-step mode.

The RPU MPCore features include:

- Two 32-bit Cortex-R5F cores based on Arm v7-R architecture with FPU
- TCM memory with single cycle read access and ECC protection
- Generic interrupt controller (GIC) based on the Arm GIC-390
- Interfaces to:
  - OCM
  - Accelerator RAM (when present)
  - I/O peripherals
  - AXI interfaces connected directly to the PL and APU

Each Cortex-R5F core includes:

- 32 KB L1 instruction cache with ECC
- 32 KB L1 data cache with ECC
- FPU: single and double precision
- Embedded trace microcell (ETM) to support real-time debug and trace; ETM communicates with the Arm CoreSight™ debug system

Each RPU processor can be individually configured for inter-processor interrupts (IPI). The RPU processors have a common power island. The TCM are divided into four banks, each with its own power island.

The RPU is documented in Real-time Processing Unit.

Tightly Coupled Memory

The tightly coupled memory (TCM) provides a deterministic, low-latency memory space for the RPU. There are multiple memory banks. The TCM banks are protected by ECC.

The distribution of TCM memory depends on the processor mode:

- Dual-processor, performance mode: each processor has 128 KB of TCM memory
- Lock-step, safety mode: TCMs are combined for a total of 256 KB of memory

The TCMs are described in Tightly Coupled Memories.

OCM Switch

The OCM switch is optimized to service RPU and APU requests directed to the OCM and accelerator RAM (XRAM).

Main Switch Masters

The LPD main switch has several masters.

Peripheral Masters

The LPD includes several peripherals attached to the LPD main switch as masters.

- Gigabit Ethernet MAC (GEM)
- USB 2.0
- 8-channel DMA unit

The LPD includes the processing system manager (PSM) processor. The PSM appears as a master on the LPD main switch.

Additional Masters

- PSM
- FPD main switch
- PL interface: S_AXI_LPD
- OCM switch, and PMC

Register Programming Interfaces

The programming interfaces include AXI and APB programming interfaces.

Platform Management Controller

The Versal™ ACAP includes a centralized platform management controller (PMC) that is required by all designs. The PMC has a power domain that is independent from both the PL and the PS. An overview of the PMC blocks, I/O, and interconnect are described in this chapter.

For details on boot and configuration or available PMC services, see Section III: Platform Boot, Control, and Status. For details on PMC units, see Section VII: Platform Processor, Configuration, and Security Units.

Primary Functions

The PMC primary functions include:

- Initialization of the device after a system reset or power-on reset (POR)
- Boot and configuration from a supported boot device
- Configure the adaptable engines using the configuration frame interface (CFI)
- Performs security core functions that supports encryption and decryption, authentication, and key management
- Provides test and debug infrastructure to support boundary-scan and Arm® CoreSight™ trace and debug
- Monitors system activity and responds to security and functional safety events
- Releases the PS from reset and provides system power and error management services

Features Supporting System Start-up to Life Cycle Management

The PMC features that support system start-up to life cycle management include:

- Controllers and monitors
  - ROM code unit (RCU) triple modular redundant MicroBlaze™ processor and dedicated RCU BootROM for initial device boot and tamper monitoring
  - Platform processing unit (PPU) triple modular redundant MicroBlaze processor and dedicated 384 KB PPU RAM for boot loader and platform management software.
  - System monitor (SYSMON) with temperature and power supply monitoring

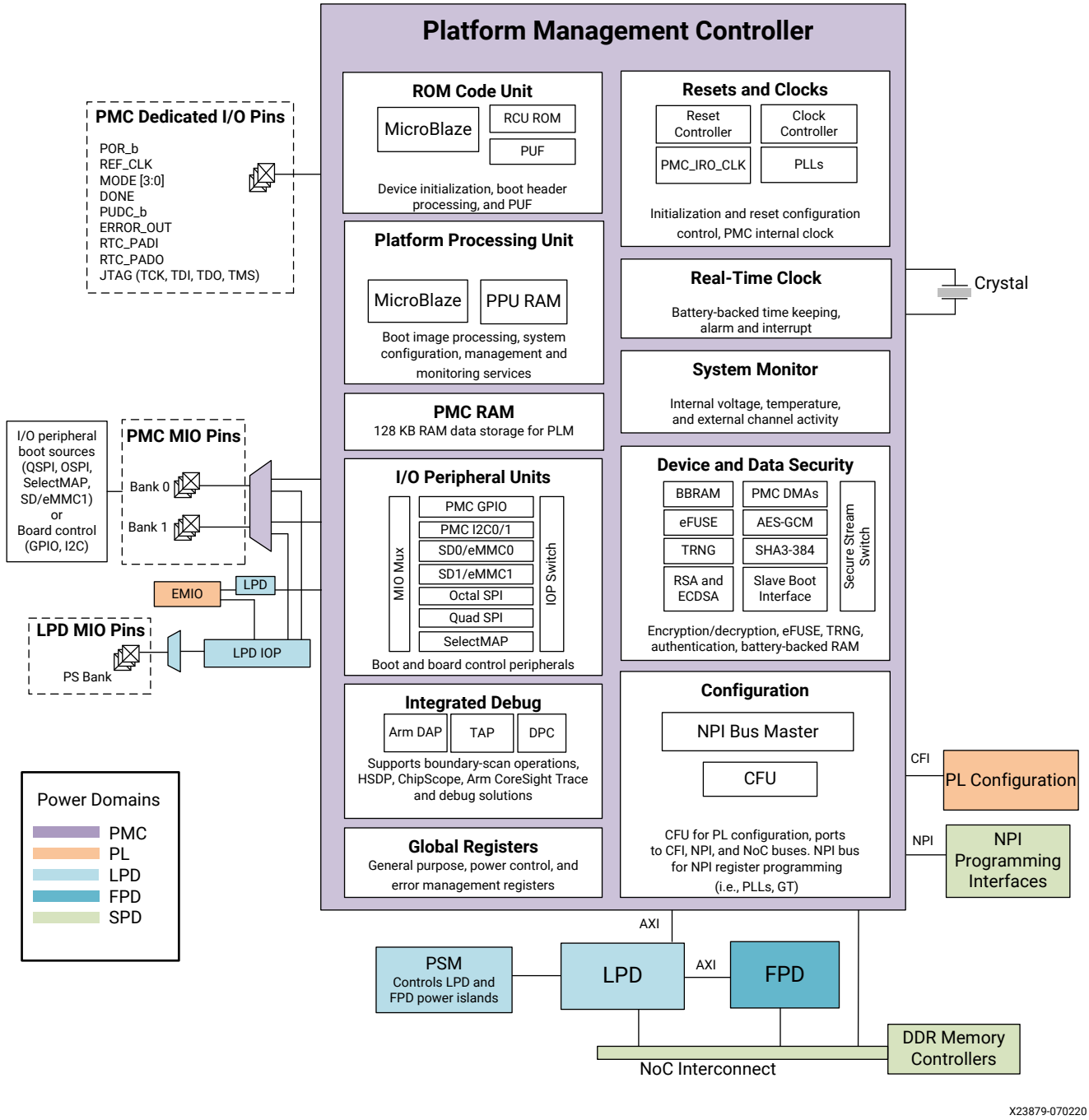
- I/O features
  - One bank of PMC dedicated I/O pins for mode, ref_clk, JTAG, RTC, reset, and status
  - Two banks of PMC MIO for flash controllers, I2C, GPIO, and SelectMAP
  - Peripheral controllers in the PMC:
    - Quad SPI controller
    - Octal SPI controller
    - Two SD/eMMC controllers
    - I2C controller
    - GPIO controller
- Memory and register features
  - 128 KB PMC RAM, used by software for PLM data processing
  - Global registers capture general-purpose, power, and error conditions
  - Two PMC DMAs transfer data within the system for configuration and processing
- Clock and time keeping features
  - Clock and reset PMC controllers handles initial steps on reset
  - Two PMC phase-locked loops (PLLs), the PMC PLL (PPLL) and NoC PLL (NPLL), generate the clock for flash interfaces and controllers. The NPLL also generates the clock for the NoC.
  - PMC delay-locked loop (DLL) for SD controller
  - Internal ring oscillator clock (PMC_IRO_CLK) provides the clock to the PMC
  - Real-time clock (RTC) time keeping
- Test and debug features
  - Debug packet controller (DPC) processes data packets for the high-speed debug port (HSDP) solution
  - Single TAP controller supports boundary-scan and Xilinx JTAG operations for configuration
  - Single DAP controller supports Arm CoreSight™ trace and debug
- Security features and accelerators
  - Xilinx memory protection unit (PMC_XMPU) for slave boot interface and PMC RAM
  - Xilinx peripheral protection units (PMC_XPPU) for I/O peripheral register modules, I/O peripheral memory space, and CFU
  - Xilinx peripheral protection units (PMC_NPI_XPPU) for NPI programming interface

- Physical unclonable function (PUF) generates two device unique signatures per die. One signature is used for the key encryption key (KEK) and one signature is used as an identification value.
- Battery-backed RAM (BBRAM) supports security key storage
- eFUSE non-volatile memory supports security key storage
- DNA unique identifier provides product traceability
- Xilinx hardware cryptographic accelerators
  - True random number generator (TRNG) generates cryptographically strong random numbers
  - RSA and elliptic curve digital signature algorithm (ECDSA) public-key cryptography enables authentication
  - AES-GCM for symmetric key cryptography enables encryption and decryption
  - SHA3-384 secure hash is used with the asymmetric algorithms to authenticate the programming device image

Block Diagram

The PMC functional block diagram identifies the PMC blocks and underlying units associated with each primary function.

Figure 6: **PMC Functional Block Diagram**



X23879-070220

Functionality

The PMC consists of control units and functional groups classified as blocks. The RCU and the PPU provide the central control and manage the PMC blocks.

ROM Code Unit (RCU)

The RCU includes a MicroBlaze™ triple modular redundant system that exclusively executes the BootROM. The RCU is the first processing unit out of reset during boot and performs the device initialization, boot interface validation, and the loading of the platform loader and manager (PLM) into the PPU RAM. The RCU releases the PPU from reset and is responsible for PUF management.

Platform Processing Unit (PPU)

The PPU includes a MicroBlaze triple modular redundant system that executes the platform loader and manager (PLM) software loaded into the PPU RAM by the RCU during hardware boot. The PLM is responsible for device boot and configuration and post-boot services. See Platform Processing Unit for more information.

Interconnect

The interconnect includes the main switch, the IOP switch, and other switches for the APB, NPI, and CFI programming interfaces. The RCU and PPU are masters on the main switch. The architecture allows PMC masters to access peripherals in the LPD. It can allow processors in other power domains to access PMC peripherals.

PMC RAM

The PMC RAM is a 128 KB RAM that is used by the PLM. This is in addition to the 384 KB PPU RAM in the PPU.

I/O Peripherals

The PMC I/O peripheral (IOP) block is a collection of peripheral controllers for initial boot and board control. The PMC IOP controllers on the PMC power domain include SD/eMMC, quad SPI, octal SPI, PMC I2C, and two GPIOs. See the Section XII: I/O Peripheral Controllers and the Section XIII: Flash Memory Controllers for more information.

Integrated Debug

The PMC integrated debug block includes the TAP controller, Arm DAP controller, and debug packet controller (DPC). This block supports basic device JTAG operations, ChipScope debug solution, Arm® CoreSight™ trace and debug, and the high-speed debug solution. See Section XV: Test and Debug for more information.

Run-time Service Request Registers

There are several sets of run-time service request registers. These are written to by system software to interrupt the platform loader and manager (PLM) running in the PPU processor. See PMC Service Requests for more information.

System Error Accumulator

The system error accumulator enables system errors to generate an event or be ignored. Events include asserting the ERROR_OUT pin, issuing a reset, or asserting an interrupt. See System Error Accumulators for more information.

System Interrupts

The PMC is a GIC proxy for system interrupts. The global register module includes interrupt status and mask registers for the 150+ system interrupts. See System Interrupts for more information.

Inter-processor Interrupts

The PMC PPU can use the inter-processor interrupt (IPI) mechanism to send and receive interrupts from other processors. Each interrupt can include a short, 32 byte message. See Inter-Processor Interrupts for more information.

Reset and Clocks

The reset and clocks block includes power-on reset and PMC clock sources (PMC reference clock, PMC PLLs, internal ring oscillator (IRO)), and manages the clock hierarchy. At start-up, the reset controller ensures that the PMC (VCC_PMC, VCCAUX_PMC, VCCO_503) voltage rails are within their minimum operating range. The clock controller provides programming registers for the PMC and NoC PLLs (PPLL, NPLL) and the clock generators for the reference clocks routed to the blocks. The PMC clock controller has a similar architecture and programming model as the PS and CPM. See Section XIV: Clocks, Resets, and Power for more information.

Real-Time Clock

The real-time clock (RTC) operates on the PMC auxiliary power domain when the device is on, or operates on the battery power domain when the device is off. The RTC maintains an accurate time base for system and application software when the device is off. The RTC has an alarm setting and can generate periodic interrupts to the PMC and other processors within the device. The alarm feature can be used for user-level system services. See Section X: Timers, Counters, and RTC for more information.

System Monitor

The System Monitor (SYSMON) resides in the PMC and monitors operating conditions on the device. The SYSMON can access internal sensors for monitoring internal power supplies and temperature. MIO or high-density I/O (HDIO) pins can be used by the SYSMON for measuring voltage levels external to the device. The results captured by the SYSMON are stored in a register map that is accessible through platform management controller resources. See the *Versal ACAP System Monitor Architecture Manual* (AM006) for more information.

Device and Data Security

The PMC device and data security block supports secure boot and security management. This block includes the Xilinx hardware cryptographic accelerators, secure stream switch (SSS), the PMC DMAs, BBRAM controller, eFUSE controller, and the slave boot interface (SBI). See Security Management and Section VII: Platform Processor, Configuration, and Security Units for more information.

Configuration

The configuration block consists of the configuration frame unit (CFU), configuration frame interface (CFI) port, and AXI4 ports to the NoC and NPI. The CFU is a bridge between the PMC main switch and the CFI, and handles the transfer of configuration data to the programmable logic configuration RAM (CRAM). See Section VII: Platform Processor, Configuration, and Security Units for more information.

I/O Signals

The PMC top-level I/O connections facilitate system management. Each Versal™ ACAP has 67 pins associated with the PMC power domain. To support core device functions and status, 15 of these pins are dedicated I/O.

The remaining 52 pins are PMC multiplexed I/O (MIO) that support the flash peripherals used to boot the device and I/O peripherals used to provide board control functions. The PMC MIO pins are split across bank 0 (Bank_500) and bank 1 (Bank_501). Each MIO bank contains 26 I/Os.

The PMC SDIO flash controllers and I/O peripherals can use the PL HDIO instead of the PMC MIO. When the PMC peripherals use the PL HDIO they are called extended MIO (EMIO). EMIO require the PMC, LPD, and PL power domains to be powered because the PMC EMIO signals route through the LPD.

For more information on the PMC I/O, see Section V: Signals, Interfaces, and Pins.

Figure 7: **PMC I/O**

PMC Interconnect Diagram

The PMC interconnect connects the master and slave AXI interfaces within the PMC and provides connectivity to the slave ports from the master ports. The PMC interconnect includes the following switches and interfaces:

- PMC main switch is an AXI switch that provides the connections to the control units (ROM code unit (RCU) and platform processing unit (PPU)). The PMC main switch also provides access to primary bus interfaces (NPI, NoC, CFI) and PMC modules (including CFU, global registers, PMC RAM, System Monitor, real-time clock, reset, and clock).
- PMC I/O peripheral (IOP) switch is an AXI switch that links flash controllers and I/O peripherals for boot and board control to the PMC main switch and system interconnect.
- Secure stream switch (SSS) is the only PMC switch that is not AXI based. The SSS is used to link to security accelerators, slave boot interface (SBI), and PMC DMAs.
- SBI enables the JTAG or SelectMAP interface.
- Configuration frame interface (CFI) provides a dedicated high-bandwidth 128-bit bus to PL for configuration and readback.
- AXI 128-bit interface link to the processing (PS) and network on chip (NoC), with conversion of the NoC protocol handled outside of the PMC.
- AXI 32-bit master bus controller port that links to the NoC programming interface (NPI).

The PMC interconnect also provides Xilinx protection units for the following:

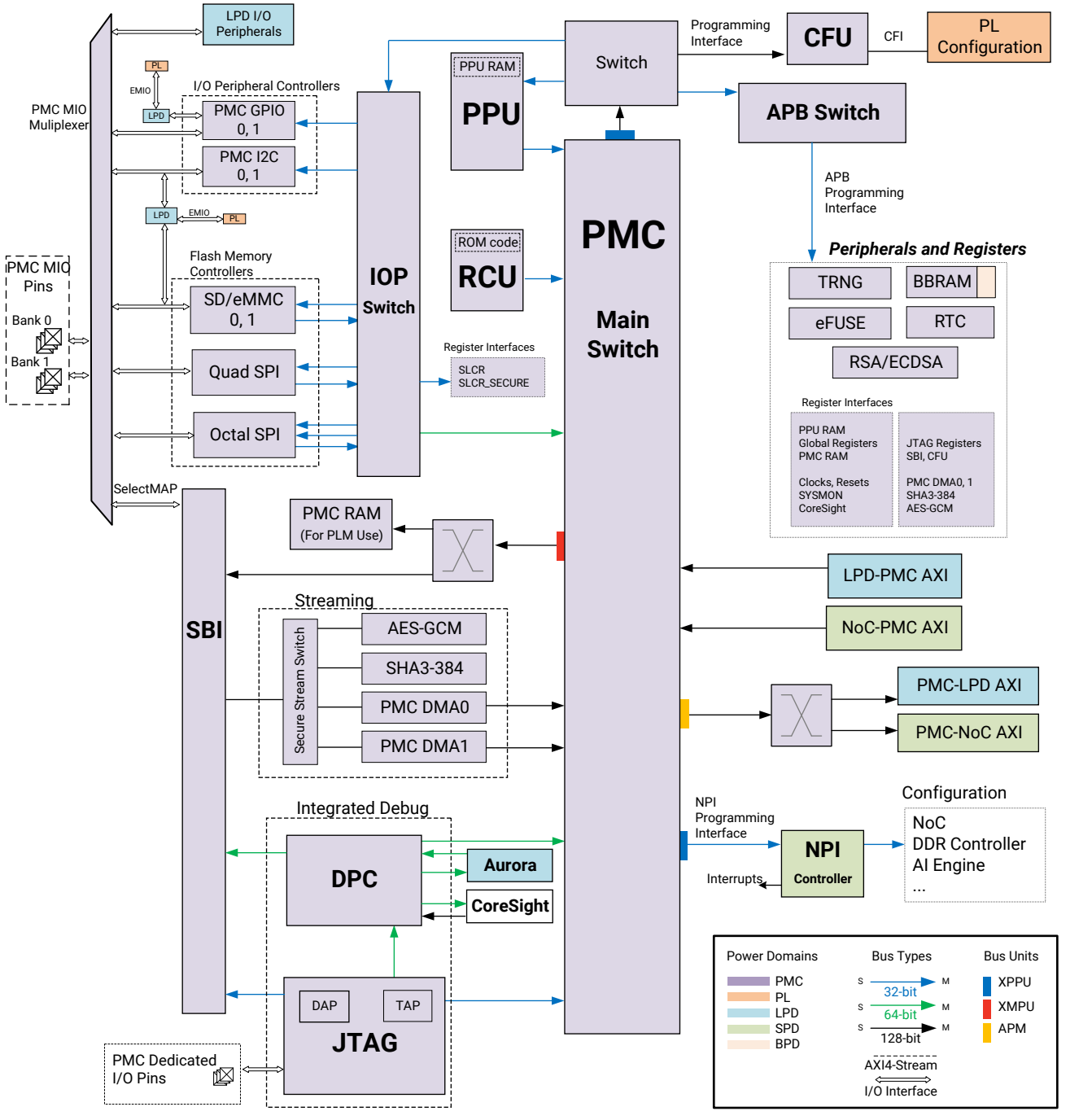
- Xilinx memory protection unit (XMPU) for the PMC_RAM.

- Network on chip peripheral interface Xilinx protection unit (NPI_XPPU) dedicated for the NPI.
- Xilinx peripheral protection unit (XPPU) used for the other PMC peripheral interfaces.

For more information on XPPU and XMPU, see Memory Protection.

The following figure shows the connectivity of the control units (RCU and PPU) and the PMC blocks, as well as which programming interfaces are protected.

Figure 8: **PMC Interconnect**



Comparison to Previous Generation Xilinx Devices

The Versal ACAP new platform management controller (PMC) centralized integration provides support for basic boot and configuration, Design Function eXchange (DFX), power management, and reliability and safety functions from a single controller. The PMC bus architecture enables significantly faster configuration and readback performance when compared with previous architectures. The following table summarizes the boot mode differences between architectures.

Table 4: **Boot Mode Comparison**

Mode	Virtex UltraScale+ or Kintex UltraScale+ FPGA	Zynq UltraScale+ MPSoC or Zynq UltraScale+ RFSoC	Versal ACAP
JTAG	Yes	Yes	Yes
OSPI	No	No	Yes
QSPI32	Yes	Yes	Yes
QSPI24	Yes	Yes	Yes
SelectMAP	Yes	No	Yes[1]
eMMC1 (4.51)	No	Yes	Yes
SD1 (3.0)	No	Yes	Yes
SD1 (2.0)	No	Yes	Yes
SD0 (3.0)	No	No	Yes
SD0 (2.0)	No	Yes	No
PJTAG_0	No	No	No
PJTAG_1	No	Yes	No
Serial	Yes	No	No
BPI	Yes	No	No[2]
NAND	No	Yes	No[2]
USB (2.0)	No	Yes	No

Notes:

1. SelectMAP mode provides hardware flow control using a BUSY signal.
2. Octal SPI and eMMC1 modes supersede the BPI and NAND modes used in previous architectures. Octal SPI and eMMC1 modes provide similar performance while reducing pin count.

Additional key differences from previous generations are listed:

- PMC has its own dedicated power domain. Unlike the Zynq UltraScale+ MPSoCPS CSU and PMU, the Versal PMC RCU and PPU are decoupled from the PS or PL power domains.
- Configuration frame interface (CFI) bus is dedicated to accessing the configuration frames and provides configuration and readback performance improvements. In conjunction with the network on chip (NoC), it replaces the internal configuration access ports (ICAP, PCAP, and MCAP) used in previous generations.

- NoC programming interface (NPI) provides register access for remote peripherals such as gigabit transceivers and DDR memory.
- Octal SPI boot mode supports compatible octal SPI flash memory with DDR mode providing a high-speed and low pin-count solution.
- SelectMAP boot mode loads configuration data and requires hardware flow control using a BUSY signal.
- Single TAP located in the platform management controller.
- Single DNA identification accessible via JTAG or in the AXI register set. Versal ACAP does not have a PL DNA or a corresponding PL DNA_PORT primitive.
- Internal configuration clock provides higher performance than prior generation.
- Debug packet controller (DPC) supports the high-speed debug port (HSDP) for processing packets from interfaces such as the Aurora in the PS or PCI Express.
- Integrated system monitor in the platform management controller.
- Enhanced encryption and decryption for increased resistance to differential power attacks (DPA).
- Two PUF outputs that are exclusively managed by the RCU, a unique readable device ID, and a unique device key encryption key (KEK) for encrypt/decrypt.
- Enhanced secure debug (RSA/ECDSA) authenticated access via JTAG).
- True random number generator (TRNG), additional AES user keys, and ECDSA authentication added for security applications.
- Connections from the gigabit transceivers, through the integrated blocks for PCI Express in the CPM and through LPD into the PMC configuration.
- The legacy quad SPI (LQSPI) controller mode is not supported in the Versal ACAP.
- Execute-in-place (XIP) is not supported by Versal device boot modes.
- JTAG accessible internal private scan registers (with USER1-4 commands) are accessed with the PS9 primitive through the control, interface, and processing system (CIPS) IP. The Versal ACAP does not have a BSCANE2 primitive.
- The Xilinx soft error mitigation (XilSEM) library is a pre-configured and pre-verified solution to detect and optionally correct soft errors in the configuration memory of Versal ACAPs.

I/O Peripherals

The I/O peripheral controllers (IOP) are accessible via the local advanced peripheral bus (APB) interconnect switches or the local AMBA high-performance bus (AHB) interconnect switch. In most cases, their I/O signals are routed through the PS-LPD and PMC multiplexed I/O (MIO), or, by default, to the extended MIO (EMIO) to the PL. For more information, see MIO-EMIO Interface Options. Some peripherals can serve as a primary boot device. See Boot Modes for more information.

Low-Speed Peripherals

The low-speed peripheral controllers include:

- SPI Controller (two in LPD)
- I2C Controller (two LPD_I2Cx, one PMC_I2C, and one special purpose SYSMON_I2C)
- PMC GPIO Controller
  - Two banks to PMC MIO (52 channels, total)
  - Two banks to PL EMIO interface (64 channels, total)
- LPD GPIO Controller
  - One bank to the LPD MIO (26 channels)
  - One bank to PL EMIO interface (32 channels)
- UART SBSA Controller (two in LPD)
- CAN FD Controller (two in LPD)

For more information, see Section XII: I/O Peripheral Controllers.

High-Speed Peripherals

The high-speed peripheral controllers are located in the LPD and include:

- Gigabit Ethernet MAC (two)
  - RGMII via the PMC MIO or LPD MIO
  - MII/GMII via the PL EMIO
- USB 2.0 Controller (one; can be device, host, OTG, or DRD)

- ULPI is routed via the PMC MIO (not LPD MIO or EMIO)

For more information, see Section XII: I/O Peripheral Controllers.

Flash Memory Controllers

The flash memory controllers are located in the PMC and include:

- Quad SPI Controller
- Octal SPI Controller
- SD/eMMC Controller (0 and 1)

Flash memory controllers can serve as primary boot devices. For SD and eMMC boot, controller 0 is used.

Programmable Logic

The PL is a scalable structure that includes adaptable engines and intelligent engines that can be used to construct accelerators, processors, or almost any other complex functionality.

PL Configuration

The connections and configuration of the PL elements are captured in the Vivado® design suite and the Vitis™ unified software platform tool chain using a programmable device image (PDI). The PDI contains PL configuration frames (CFRAME), which are sent by the PLM to the configuration frame unit (CFU) for processing. The CFU interfaces to the PL via the configuration frame interface (CFI). The PL can be configured during the boot process and can be re-configured during normal system operation. The PL configuration can be read-back for debug and functional safety applications. The CFU is described in Configuration Frame Unit and the CFI is described in Configuration Frame Interface.

Building Blocks

The PL includes building blocks and provides several types of connections to many parts of the device including several subsystems and I/O. The PL has AXI interfaces to the PS, CPM, AI Engine, and the integrated controllers. The PL also has port interface signals and parameter configuration inputs to the PS, PMC, and other parts of the system.

The PL building blocks include the DSP Engine, configurable logic block (CLB), Block RAM, and UltraRAM integrated components. These components are surrounded by clocking structures and wiring pathways. The PL makes connection between PS, CPM, PMC, NoC, AI Engine, GTs, XPIO, high-density I/O (HDIO) buffers, and components instantiated within the PL.

The PL building blocks include:

- DSP Engine (intelligent)
- CLB (adaptable)
- Block RAM and UltraRAM (adaptable)

The PL also contains clocking structures and PLLs for the fabric and I/O.

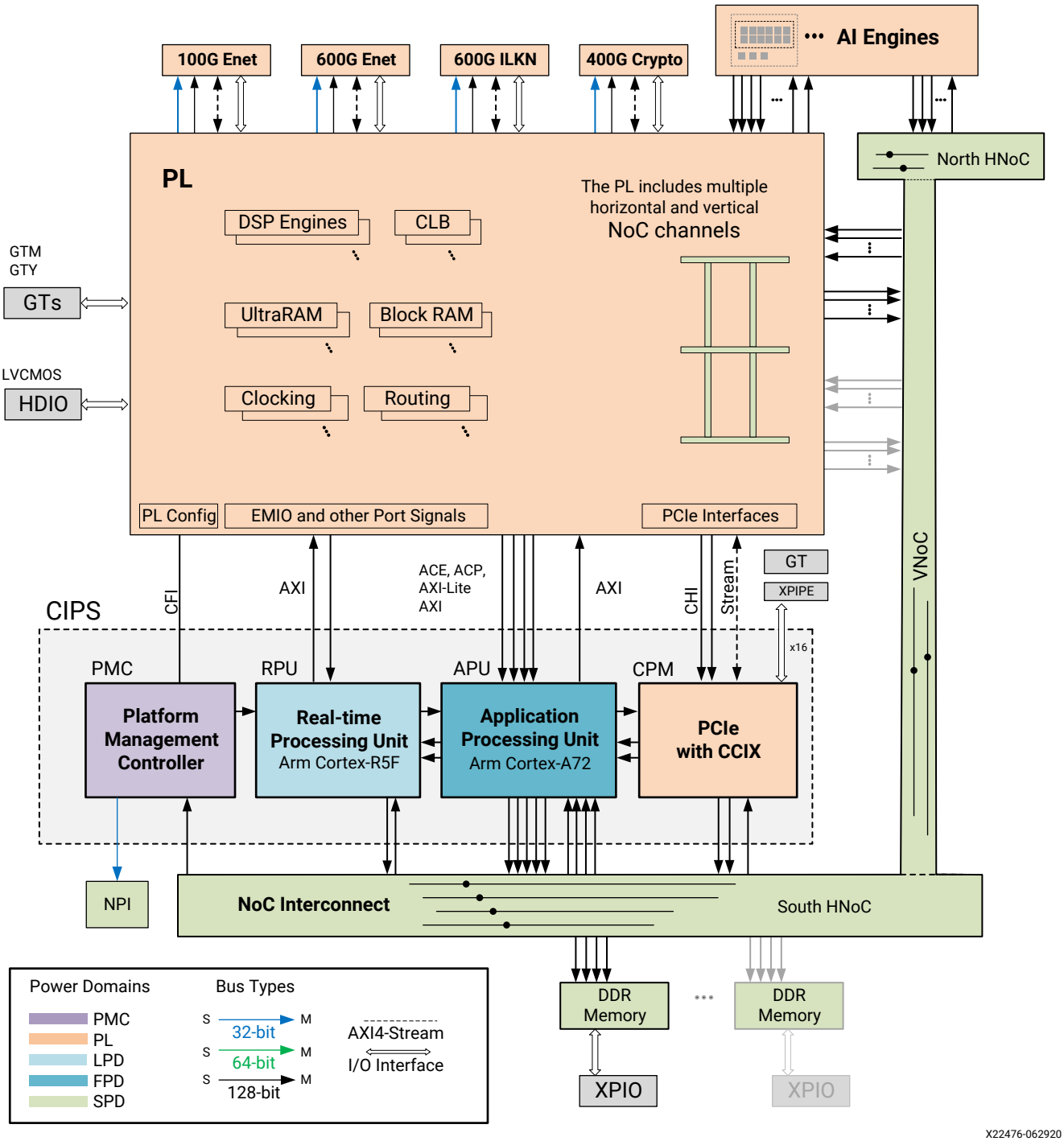
The Vivado tools provide a large library of complex functional components (microprocessors, peripherals, filters, etc.), that can be instantiated and connected to create a design. Additionally, a hardware description language can be used to describe specific functions in the design. The Vivado tools then translate the design into the building blocks of the PL. The PL can be partially or fully programmed during the boot start-up and as a service operation when the system is operating.

PL System Perspective

The PL building blocks and clock structures provide the foundation for instantiating functionality. The PL is provided with port interface signals attached to nearly every part of the device.

The high-level PL perspective of the system is shown in the following figure.

Figure 9: **PL System Perspective**



In addition to the PL interconnect interfaces shown in the figure, the PL port interface signals include the system interrupts, errors, events, and other signals from all parts of the device.

Adaptable Engines in PL

The PL Adaptable Engines are building blocks to instantiate functional units in the PL and they include:

- Digital signal processing engine (DSP)
- Configurable logic blocks (CLB): logic and LUT
- Block RAM: 4 KB data with ECC (36 Kb)
- UltraRAM: 32 KB data with ECC (288 Kb)

Digital Signal Processing Engine

Versal devices have many dedicated low-power DSPs combining high speed with small size while retaining system design flexibility. The DSP resources enhance the speed and efficiency of many applications beyond digital signal processing such as wide dynamic bus shifters, memory address generators, wide bus multiplexers, and memory-mapped I/O registers. The DSP engine is defined using the Xilinx DSP58 primitive.

For more information, see *Versal ACAP DSP Engine Architecture Manual* (AM004).

Configurable Logic Block

The configurable logic block (CLB) includes logic and look-up tables (LUTs) that can be configured into many different combinations and connected to other components in the PL to create special purpose functions, processing units, and other entities.

Every CLB contains 32 LUTs and 64 flip-flops. The LUTs can be configured as either one 6-input LUT with one output, or as two 5-input LUTs with separate outputs but common inputs. Each LUT output can optionally be registered in a flip-flop.

In addition to the LUTs and flip-flops, the CLB contains arithmetic carry logic and multiplexers to create wider logic functions. Within each CLB, 16 LUTs can be configured as 64-bit RAM, 32-bit shift registers (SRL32), or two SRL16s.

Within every CLB are dedicated interconnect paths for connecting LUTs without having to exit and re-enter a CLB, drastically reducing the use of global routing resources. In addition, new CLB features such as cascade multiplexers allow flexible carry logic structures to be created.

For more information, see the *Versal ACAP Configurable Logic Block Architecture Manual* (AM005).

Block RAM

The dual-port block RAMs have 4 KB of data storage capacity that is protected by error correction coding (ECC) for a total of 36 Kb per block RAM. The RAM can be configured as either one 36 Kb RAM, or two completely independent 18 Kb RAMs. The RAMs can be configured to operate in simple dual port mode (one read-only port and one write-only port) or true dual port mode (both ports have read and write interfaces). Each port can be configured independently as 4K×9, 2K×18, 1K×36, or 512×72. The 512×72 mode requires simple dual port mode.

The block RAM is described in the *Versal ACAP Memory Resources Architecture Manual* (AM007).

UltraRAM

The dual-port block RAMs have 32 KB of data storage capacity that is protected by error correction coding (ECC) for a total of 288 Kb per block RAM. Each port can be configured independently as 32K×9, 16K×18, 8K×36, or 4K×72.

- Cascade-able for building larger memories: dedicated column routing wires to connect adjacent units
- ECC on both ports with single bit error detection and correction, and double bit error detection
- Sleep power saving features

The UltraRAM is described in the *Versal ACAP Memory Resources Architecture Manual* (AM007).

I/O Connectivity

The Versal™ ACAP includes many different types of I/O pins. Each pin has a dedicated I/O buffer with characteristics that are often programmable. The functionality of a pin can be dedicated to a specific function or have a flexible assignment.

The I/O functionality and buffers for the PMC, PS, and other subsystems are summarized in the following table.

Table 5: **PMC, PS, and Other I/O Buffer Groups**

Functionality Pin Group		Pin Count	I/O Buffer Group		Notes
Description	Name		Name	Voltage Bank	
PMC dedicated	PMC DIO	15	PSIO	VCCO_503 Bank_503	POR, Ref clock, JTAG, boot mode. See PMC Dedicated Pins.
PMC analog	PMC DIO_A	6	Analog	VCCO_500 Bank_500	
PMC MIO	PMC Bank 0 PMC Bank 1	52	PSIO	VCCO_500 Bank_500 VCCO_501 Bank_501	Programmable functionality: Multiplexed I/O. Programmable I/O buffer: MIO PSIO Buffer Configuration I/O Peripherals.
LPD MIO	LPD Bank	26	PSIO	VCCO_502 Bank_502	
PS GTs	PS XPipe	16			Attached to PCIe and HSDP Host Debug Ports.
PL XPIO	XPIO bank	54x			Normally used by DDRMC. Available to PL fabric.
PL CMOS	HDIO	x	HDIO		Multiple banks of HDIO buffers connect PL to device pins.
PL GTM	GTM	x	GTM		
PL GTY	GTY	x	GTY		*Versal ACAP GTY Transceivers Architecture Manual* (AM002).

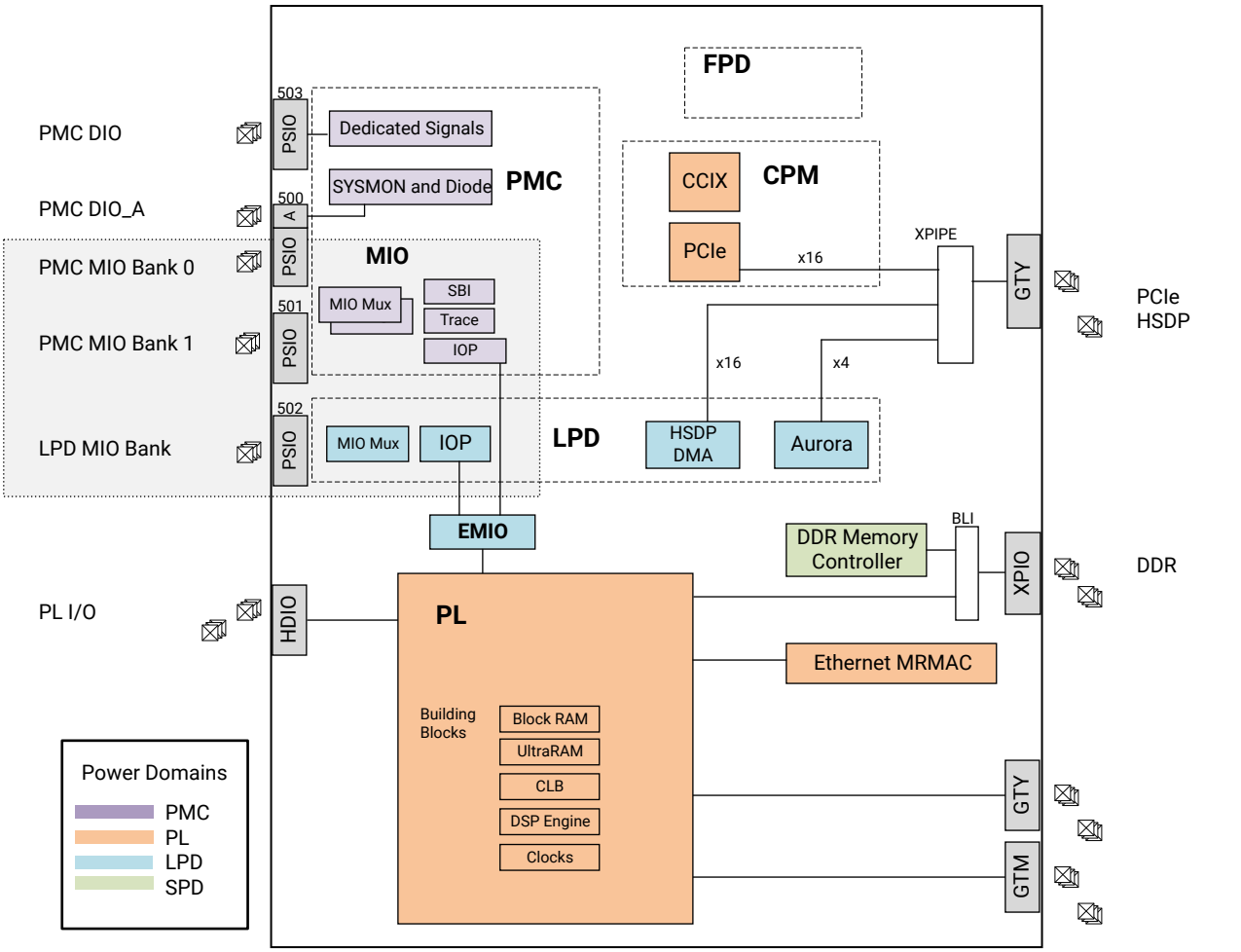
Related Information

Signals, Interfaces, and Pins

Device-Level Diagram

The device-level I/O connectivity diagram is shown in the following figure.

Figure 10: **I/O Connectivity Diagram**



X22413-063020

I/O Interface Summary

The I/O interfaces are divide in several parts. The PMC and PS I/O signals are described in the TRM, see PMC Dedicated Pins and Multiplexed I/O.

The other device signals are described in their respective architecture manuals.

I/O Buffer Features

The features for the I/O buffers are summarized in this section.

PMC and PS PSIO Buffers

The PMC and PS PSIO buffers features include:

- 1.8 to 3.3V signaling based on a per bank basis
- Output drive strength and slew rate
- Inputs independently programmed
  - Weak pull-up, weak pull-down, or weak keeper
  - Hysteresis

The output signaling voltage level is determined by the power pins for the bank.

PL HDIO Buffers

The HDIO buffer is optimized for high-performance signaling, and it also includes lower frequency, LVCMOS circuits.

- Output control: drive strength and slew rate
- Settings on a per-bank basis
  - I/O voltage swing:
    - LVCMOS: 3.3V, 2.5V, and 1.8V
    - HSTL: 1.8V
    - SSTL: 1.8V
- Inputs independently programmed
  - Weak pull-up, weak pull-down, or weak keeper
  - Hysteresis
- HD IOL logic resources support low-speed interfaces with SDR and DDR logic
- IODELAY feature with cascadable output delay
- Coarse data alignment
- On-die termination
- Common internal VREF on per bank

- Receive a differential signal at low-speed
- Transmit a pseudo differential signal
- External termination for LVDS and LVPECL inputs

Operating Modes

- LVCMOS, HSTL, SSTL single-ended signals
- Transmit single-ended signals with pseudo-differential mode
- Receive single-ended and differential signals; differential receiver for low-speed clock inputs

System View

The HDIO is arranged in banks of 22 buffers each to connect the PL to the device pins. The PL includes multiple banks of HDIO buffers. The number of HDIO banks varies depending on the device and package size. Examples include the following:

- Bank 306: PL with 22 pins
- Bank 406: PL with 22 pins

Programming Model

The I/O characteristics of the HDIO buffers are controlled by the parameters that are configured by the Vivado® design suite wizard.

PS XPIO Buffers

The I/Os in the XPIO supports both high-performance and low-speed interfaces. Each XPIO can use the XPHY to align, serialize, and deserialize a data stream. Each XPIO bank has nine nibbles of six cells each for a total of 54 pins.

The XPIO input and output buffers support a wide range of single-ended and differential I/O standards along with resources to support a high level of signal quality. The XPIO provides:

- 1.0V, 1.1V, 1.2V, 1.35V, and 1.5V bank voltage standards
- XPHY logic resources to align and serialize/de-serialize high-speed data streams
- IOL resources to provide simplified lower-bandwidth SDR and DDR logic
- Internally generated VREF support shared across nibble boundaries
- Calibrated output drive
- Calibrated internal termination
- Internal differential termination and bias offset
- Transmitter pre-emphasis and receiver equalization

- Native MIPI D-PHY interfacing
- Serialization/deserialization ratios of 1:8, 1:4, and 1:2

System Performance

There are inherent performance features in the system and several performance related configuration options. The TRM describes the inherent performance features and the functionality that can be used to obtain an optimal configuration.

- Inherent NoC interconnect design features with configurable, multichannel structures
- Multiple interconnect traffic types to control quality of service (QoS)
  - Isochronous for video and other time-sensitive transactions
  - Low latency for communications and other applications
  - Best effort, bulk traffic for large data sets without critical timing needs
- Intelligent DDR memory controller scheduler
- Hardware acceleration in PL instantiated functions

Performance Tuning

Performance tuning builds on the inherent features. This includes properly routing NoC traffic, optimizing the use of the DDR memory controller, and using the QoS traffic types. Performance tuning is not covered in the TRM.

Interconnect Features

The interconnect has dedicated 128-bit AXI channel connections between the subsystems. These include low-latency datapaths and high-throughput datapaths with buffering. There are also noteworthy datapaths.

The interconnect optimizes the performance of the RPU and APU. The interconnect port connections are shown in PS Interconnect Diagram and listed in this section.

Low-latency Datapaths

- APU to NoC: CCI connections to the NoC
- RPU to NoC: AXI master on OCM switch
- RPU to OCM: AXI master on OCM switch

- RPU to its TCMs: two cycle access with deterministic execution

High-throughput Datapaths

Popular high-throughput datapaths:

- APU to NoC with four CCI master ports
- RPU to NoC with main switch master port
- LPD DMA to FPD main switch

Noteworthy Datapaths

- APU to CCI to FPD main switch to OCM switch to OCM and XRAM (if available)

Transaction Quality of Service

Each transaction includes a quality of service (QoS) traffic attribute.

- Low-latency
- Isochronous
- Bulk transfer

The QoS attribute is recognized by the AMBA® switches and DDR memory controller. System performance can be obtained by setting the QoS attributes appropriately. Each master can generate one or more QoS values. The traffic types are detailed in Quality of Service.

XILINX

Section III

Platform Boot, Control, and Status

This section includes these chapters:

Overview

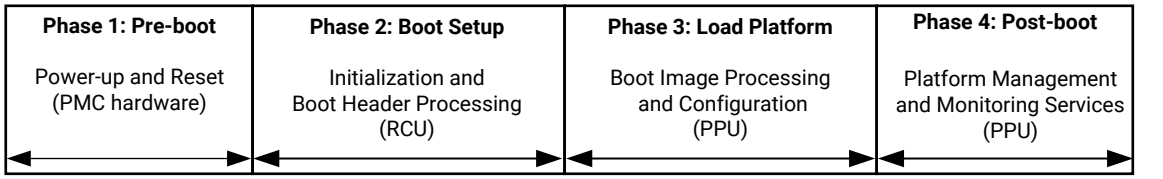
The platform management controller (PMC) supports platform management during boot, configuration, and run-time. The boot flows, boot modes, and example platform management services are described in this section.

For an introduction to the PMC architecture, see Platform Management Controller. For information on the PMC units, see Section VII: Platform Processor, Configuration, and Security Units.

System Start-up Phases

The Versal™ ACAP has four key system start-up phases from boot through life-cycle management.

Figure 11: **System Startup Phases**



Non-Secure Boot Flow

In the non-secure boot flow chapter, the system start-up phases required to boot a programmable device image into the Versal ACAP are discussed. The PMC primary functional control units and their responsibilities are described.

Secure Boot Flow

The secure boot flow enables programmable device images to use decryption and authentication to protect and secure user designs and the IP stored in Versal ACAPs. The hardware root of trust and encrypt-only secure boot methods are described.

Boot Modes

For design flexibility, the Versal ACAP supports multiple boot-mode options. This chapter discusses selection criteria trade-offs between the boot modes and provides details about each boot mode interface.

Boot Image - Programmable Device Image

The programmable device image format for the Versal ACAP is highlighted in this chapter and the boot header read by the PMC RCU is provided.

Platform Management

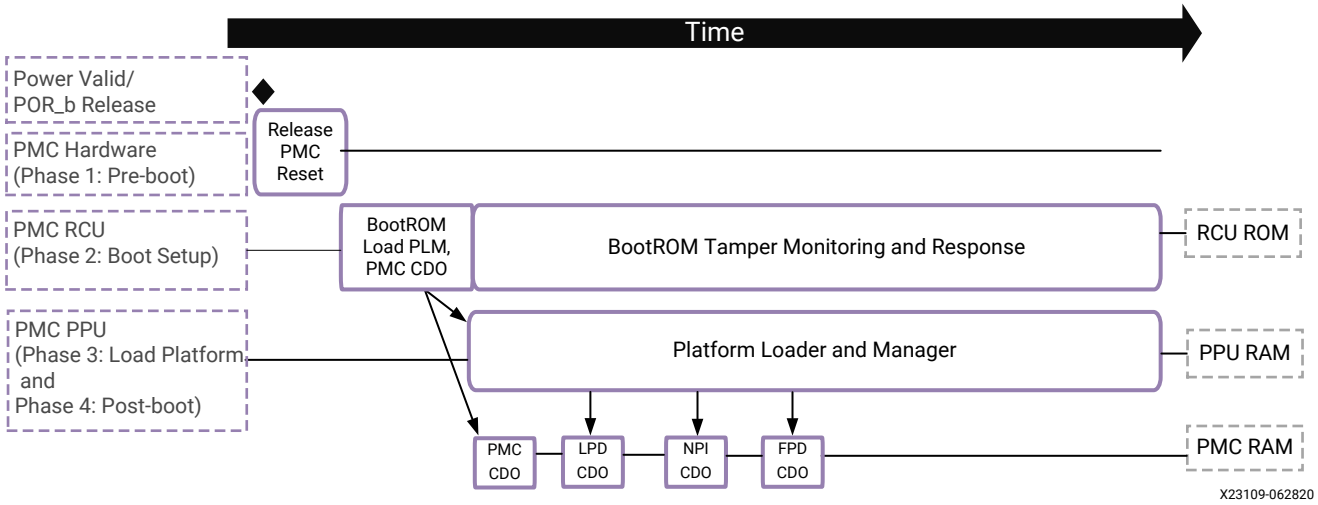
The Versal ACAP supports several run-time services including the following, which are introduced in this chapter.

- Functional safety management: assists with managing random faults. ECC protected RAMs, RCU and PPU controller triple modular redundancy, XPPU, and XMPU are a few of the features provided.
- Design Function eXchange (DFX): enables a board host connection to be maintained while multiple application functions are loaded and reloaded into the device. The Versal architecture supports using a shell and workload setup for dynamic reconfiguration. The shell (hardened infrastructure with required PMC and interfaces to host) remains static, but different workloads (user designs) can be plugged into the shell to support multiple applications.
- Power management: applications that must limit or optimize power consumption use power modes such as sleep to meet requirements. The power modes and some options for power management are discussed.
- Security management
    - Secure key storage and management: options for key storage and key management.
    - Tamper monitoring and response: features that can be used in developing mitigation techniques to resist tamper attacks.
    - User access to Xilinx hardware cryptographic accelerators: Versal ACAP has multiple cryptographic functions that can be accessed post-boot.
- Soft error mitigation: soft error mitigation techniques and feature support in Versal ACAPs.

Non-Secure Boot Flow

For system start-up, a Versal™ device must successfully initialize, boot, and configure from a supported boot source. Both non-secure and secure boot flows are supported. This chapter details the non-secure boot flow. For secure boot flow details, see the Secure Boot Flow chapter. The following figure illustrates the non-secure boot flow high-level phases.

Figure 12: **High-Level Non-Secure Boot Flow**

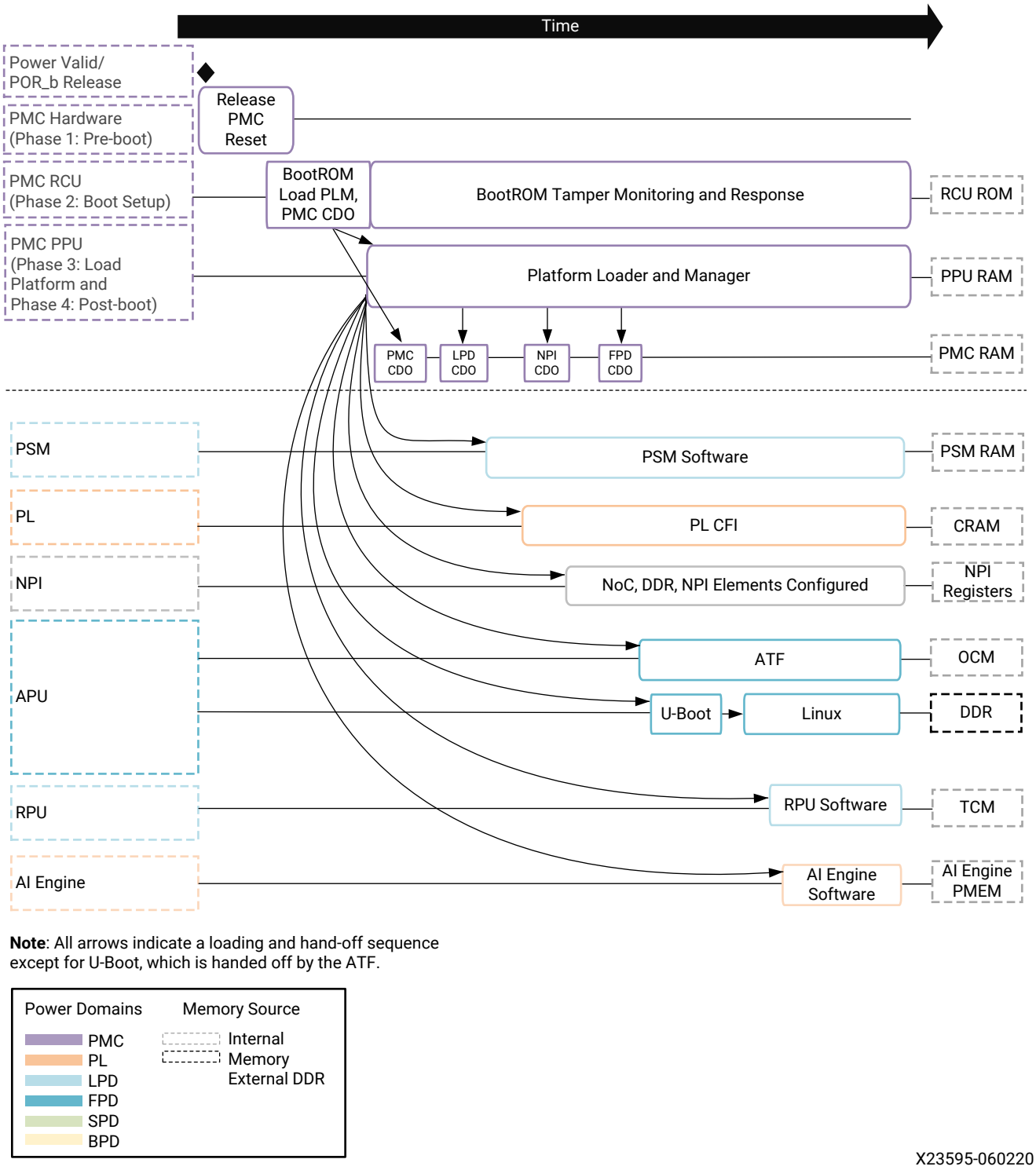


The three platform management controller (PMC) functional blocks that control the non-secure boot process are:

- PMC hardware dedicated state machines
- PMC ROM code unit (RCU)
- PMC platform processing unit (PPU)

The following figure shows the PMC functional blocks primary responsibilities and their memory source at each phase in the non-secure boot flow. The figure provides one example in which the major partition components (except for Linux) are loaded by the platform loader and manager. Linux is loaded by U-Boot.

Figure 13: **Example Standard Boot Flow Processing Engines and Memory Sources**



Note: All arrows indicate a loading and hand-off sequence except for U-Boot, which is handed off by the ATF.

X23595-060220

There are many different application partition requirements, and the Versal device provides the flexibility to address them. For example, some application protocols might require the RPU partition to be loaded first, or the software to be loaded from U-Boot, while other applications might not require the RPU or AI Engine partitions at all. Each phase in the example non-secure boot flow figure is described below.

Phase 1 (Pre-boot)

In phase 1, the non-secure and secure boot flows execute the same sequence of steps. The PMC hardware must detect that the power is valid (VCCAUX_PMC, VCC_PMC, and VCCO_503) and that the external POR_b pin is released to initiate a boot sequence. Dependent on the boot mode selected and application other power supplies will be required.

After power is applied to the device, the dedicated PMC hardware state machines perform a series of mandatory tasks to prepare the system for the PMC RCU release. The tasks include capturing the value of the boot mode pins into a PMC register for the RCU to read. The test interfaces (e.g., JTAG) initialize to a known secure state. This is followed by scan clear, where the registers in the PMC are zeroized and readback to confirm scan clear was successful. Next, the dedicated hardware hashes the PMC immutable BootROM using the SHA-3/384 engine and compares the calculated cryptographic hash against a golden copy stored in the device. If the hashes match, the integrity of the RCU ROM is validated and the PMC is released from reset If the hash comparison fails an error is flagged. The default action is to log and continue until the PLM can determine what action to take.

Phase 2 (Boot Setup)

In phase 2, the PMC RCU non-secure and secure boot flow steps begin to diverge. See Secure Boot Flow for details on the additional security checks available. In the default non-secure boot flow, the PMC RCU performs basic integrity checks. The RCU initializes PMC blocks such as the System Monitor and the PMC PLLs. Checks for voltage and the PLL lock are performed.

After the initial security and integrity checks pass, the RCU reads the boot mode register value to determine the boot mode configuration required. If a slave boot mode is detected, the RCU enables the SelectMAP or JTAG interface path and then hands the control to the user to load the programmable device image.

When a master boot mode is detected, the RCU initializes the corresponding boot interface and searches for a valid boot header within a programmable device image (PDI). To validate a boot header, the RCU looks for the identification string XLNX. When a valid identification string is found in the boot header, the checksum for the boot header is checked. If the checksum is valid, the rest of the programmable device image boot header and platform loader and manager (PLM) are loaded into the PPU RAM.

If a valid boot header is not found, the image search is initiated for master boot modes. The search works differently depending on the type of master boot mode selected. For OSPI and QSPI boot modes, the programmable device images can be located every 32 KB in the boot memory device, which allows for more than one image to be stored in the flash memory device. If an image header is invalid, the BootROM increments the image header address register by 32 KB and tries again. For SD and eMMC boot modes, the 8191 FAT files can be searched for the identification string.

The RCU checks and validates the image signature, and then copies the platform loader and manager into the PPU RAM. The RCU releases the PPU from reset to begin phase 3 (load platform) and the RCU enters a sleep state, wake on interrupt for service routines throughout Phase 3 and Phase 4.

Phase 3 (Load Platform)

In phase 3, the PMC PPU executes the PLM from the PPU RAM. The PLM reads the programmable device image from the boot source and the PLM configures the components of the system including the NoC initialization, DDR memory initialization, programmable logic, and processing system, and then completes the device boot.

If a boot header is valid, but the PLM determines the boot image is corrupt, the PLM can recover by writing the location of another boot header into the MultiBoot register, and issuing an internal system reset (not an external POR_b reset). After the system reset, the boot header is fetched from the address location equal to the value of the MultiBoot register multiplied by 32 KB. When the fallback boot header is invalid, the RCU continues normally with its boot image search function if the boot device supports image search.

Phase 4 (Post-boot)

After the non-secure boot flow is complete, the PLM is active and numerous services can be run in this phase. Services include power management, partial reconfiguration, system error management, safety monitoring, security monitoring, and soft-error mitigation.

For more information on the Versal ACAP boot process see the *Versal ACAP System Software Developers Guide* (UG1304).

Secure Boot Flow

This chapter describes the Versal™ ACAP secure boot features. The Versal device supports two secure boot modes: Hardware Root of Trust (HWRoT) and Encrypt-only (EO). The HWRoT mode uses asymmetric authentication with optional encryption to provide for both authenticity and confidentiality of the boot and configuration files. The EO mode does not use asymmetric authentication, and instead encrypts all portions of the boot and configuration files excluding the boot header. Confidentiality, integrity, and authentication are provided with the use of AES-GCM/256.

***Note*:** Because authentication in EO mode is only provided by the encryption process, the boot header is not authenticated and cannot be relied upon for security critical information. As such, security critical information contained in the boot header is ignored in lieu of information stored in eFUSEs. See Encrypt-only Secure Boot for more details.

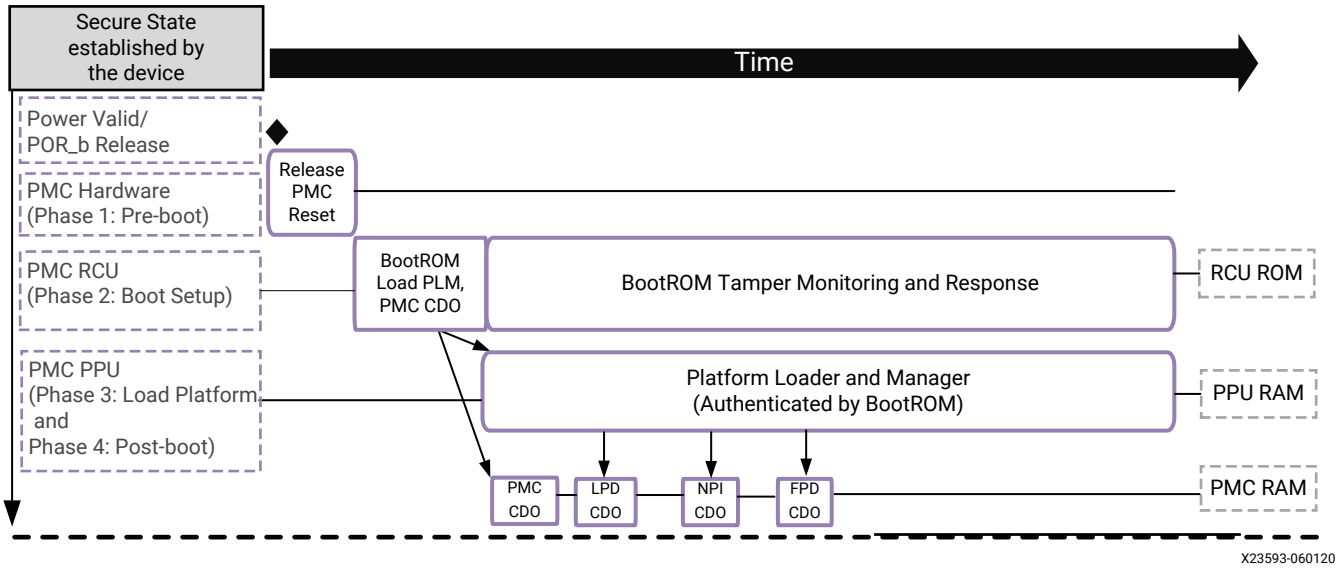
***Note*:** The Versal device allows for two methods to protect its secret symmetric keys from differential power analysis (DPA): protocol and built-in leakage reduction. Each method can be used individually or together to create enhanced protection.

The functional blocks in a secure boot process are:

- Dedicated hardware state machines in the PMC
- PMC ROM code unit (RCU)
- PMC Platform processing unit (PPU)

The high-level boot flow summary is shown in the following figure.

Figure 14: **High-Level Secure Boot Flow Summary**



After the power is applied to the device, the dedicated hardware state machines perform a series of mandatory tasks. First, all test interfaces (e.g., JTAG) initialize to a known secure state. Second, all registers in the PMC are zeroized (reset + verification of reset state). Before execution of the PMC BootROM, the dedicated hardware hashes the immutable BootROM code using the SHA-3/384 engine and compares the calculated cryptographic hash against a golden copy stored in the device. If the hashes match, the integrity of the BootROM is validated, and the PMC RCU is released from reset. If the hash comparison fails an error is flagged. The default action is to log and continue until the PLM can determine what action to take. However, eFUSEs can be programmed to halt the secure boot process immediately and go into a secure lockdown state when an error occurs.

Once released, the PMC RCU becomes the center of the secure boot process. It is responsible for all mandatory and optional security operations, as well as the secure loading of the PLM. A list of all security checks at this stage are listed in the following table. Optional checks are enabled by programming eFUSEs.

Table 6: **Security Checks**

Security Operation	Description	Optional?
Zeroize PMC RAM	The PMC RAM has zeros written to it and read back to confirm the write was successful	No
User-defined environmental monitoring	Temperature and voltage are monitored to ensure operation within user-defined limits	Yes
Secure debug	JTAG can be enabled through a valid cryptographically strong authentication method	Yes
Known answer tests	Known answer tests are performed on the cryptographic engines used for loading the PLM prior to them being used	Yes
NoC configuration (SSI technology only)	Configuration of the NoC on SSI technology devices	No

The RCU also enforces the secure boot modes (HWRoT or EO), if enabled, and is responsible for governing that transition of security state by prohibiting the transition from secure to non-secure or non-secure to secure without a full power-on reset (POR).

After all checks pass, the RCU securely loads the PLM (authenticated and, if desired, encrypted). Once loaded, the PLM can check the error messages from inside the device to determine what security actions, if any, are necessary.

Hardware Root of Trust Secure Boot

The Versal device HWRoT boot mode is built upon the use of RSA-4096 or ECDSA P-384 asymmetric authentication algorithms using SHA-3/384, and allows the use of both primary and secondary public keys for signature verification (PPK and SPK, respectively). The following table lists the characteristics of each public key type.

Table 7: **Public Key Types**

Public Key	Number	Location	Revocable
Primary (PPK)	3	External memory with hash in eFUSEs	Yes
Secondary (SPK)	256	Boot image	Yes

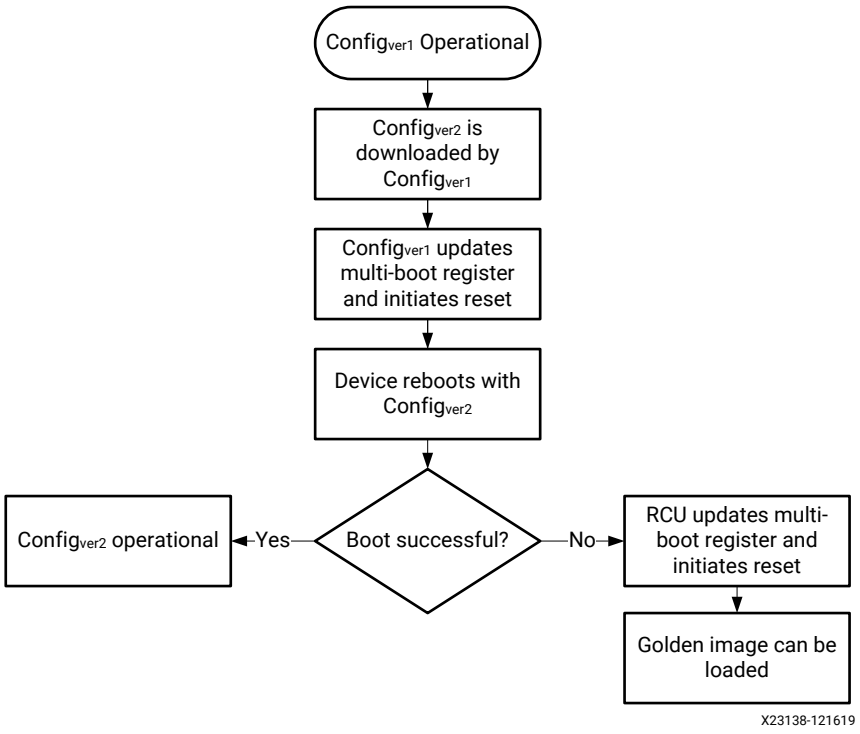
The Versal device allows for the use of three PPKs, each of which is revocable. To reduce the number of fuses required, the full public key is stored in external memory (e.g., flash) while 256 bits of a SHA-3/384 hash of each key is securely stored inside the device using eFUSEs. During the secure boot process, the RCU first validates the integrity of the full public key stored externally by hashing it (SHA-3/384) and taking 256 bits of that hash and comparing against the value stored in eFUSEs.

There are also 256 SPKs available, each of which are also revocable. The SPK is delivered inside the authenticated boot image, and is consequently protected by the PPK, which is the primary purpose of the PPK. The SPK is intended to authenticate everything else.

Configuration Update

Configuration update is a critical capability to enable such features such as an over-the-air (OTA) update. The Versal device supports the ability to update the configuration in-system. The following figure shows a high-level flow diagram of a configuration update performed by the RCU.

Figure 15: **Configuration Update**



In this notional system, it is assumed that revision of the design ($Config_{rev1}$) and a golden image are both stored in external memory. Some applications choose to use a golden image as a backup (or fallback). It is typically not full-featured but provides basic diagnostic and communication capabilities in the event of a failed boot of the primary image. This notional system is built upon to demonstrate key revocation.

The initial design, $Config_{rev1}$, is notified when an update is desired. $Config_{rev1}$ then downloads $Config_{rev2}$, writes to the multiboot register, and initiates a reset. Upon successful completion, $Config_{rev2}$ is booted and becomes operational. In the event of a failed boot, the RCU automatically increments the multiboot register and initiates a reset. As the golden image is stored at a higher address in external memory, it is ultimately loaded and communication can be re-established (assuming this in an OTA case).

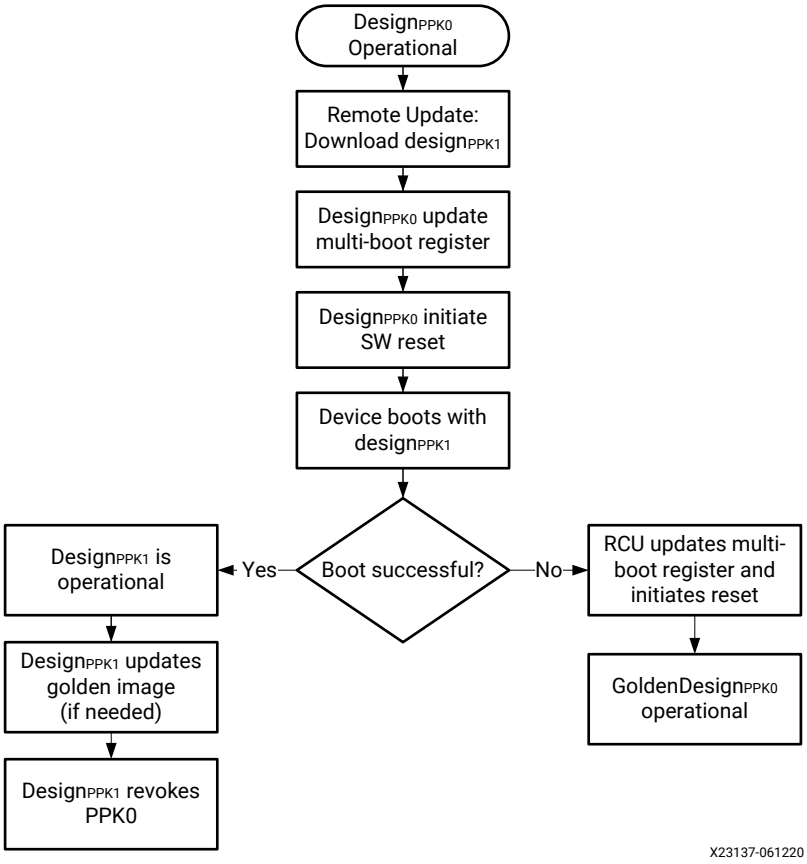
Configuration Update with Key Revocation

Now that a baseline configuration update use case has been described, it is necessary to look at that use case when booting using the HWRoT mode. Key revocation is an integral part of any public key system. When keys are changed (as is a good key management practice), or if a private key has been compromised, the ability to revoke keys is needed to provide rollback protections. This section describes the process of revoking both PPKs and SPKs, as well as the use of revocation as a tamper penalty.

PPK Revocation

There are three PPKs in the Versal device. Each PPK has a set of revocation bits implemented with eFUSEs. Programming these fuses invalidates (revokes) that PPK permanently preventing its use. The following figure demonstrates the OTA use case with the revocation of a PPK (PPK0 in this instance).

Figure 16: **Configuration Update with PPK Revocation**



In the notional system, it is assumed that a revision of the design ($Design_{PPK0}$) and a golden image ($GoldenDesign_{PPK0}$) are both stored in external memory and both are signed by PSK0 (authenticated by PPK0).

***Note*:** The subtext of the design/image name represents the public version of the key used to sign the image.

Again, this is a representative system used to describe the process of updating a system when it is necessary to revoke a PPK. It is not a requirement.

The initial design, $Design_{PPK0}$, is notified when an update is desired. In many cases, the design itself is responsible for supporting the remote update. $Design_{PPK0}$ increments the multiboot register and then initiates a reset. $Design_{PPK1}$ is then booted, and if successful, begins operation. $Design_{PPK1}$ should then update, if necessary, the golden image and then program the eFUSEs to revoke PPK0. In the event of a failed boot, the RCU increments the multiboot register and initiates a reset. As the golden image is stored at a higher address in external memory, it is ultimately loaded, and communication can be re-established (again, assuming this is an OTA case).

SPK Revocation

The revocation of an SPK follows a very similar process as described in PPK Revocation. However, the difference is that the SPK and its corresponding revocation ID are part of the boot image (authenticated using the PPK). The revocation of an SPK is done by modifying the 256-bit SPK revocation ID field in the eFUSEs (representing 256 possible revocations). This revocation ID acts as a pointer to a revocation list. If the device boots with an old SPK and ID, the RCU or PLM flags this as invalid and prevents the device from booting with that image/partition.

Revocation as a Tamper Penalty

Key revocation not only allows for good key management practices (periodic key changes) but also can serve as a tamper penalty. This dual role can be a very valuable addition to a secure system. In the event of a tamper event, the system can revoke the PPK or SPK currently being used and initiate a reset. This revocation invalidates the current boot image and prevents the system from booting, which halts operation and protects the system from additional threats. The system would then have to be taken back to the depot and flashed with an image signed by a different (valid) key. This method represents a temporary penalty. However, some systems might desire a more drastic response. In this case, the system that detects the tamper event can revoke all PPKs. This revocation essentially “bricks” the part as there is no longer a valid key with which to boot (all have been revoked). This is a permanent penalty and is typically used only in the most secure systems as there is no method to recover the use of the device.

Encrypt-only Secure Boot

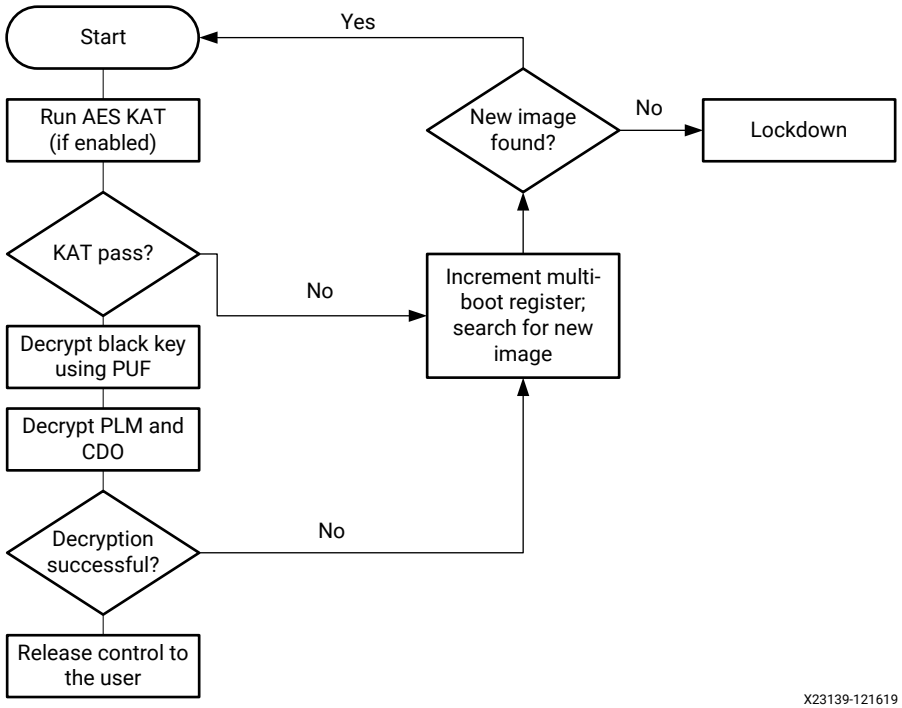
The Versal device encrypt-only secure boot mode is enabled through the programming of eFUSEs. Similar to the HWRoT mode, this mode provides confidentiality, integrity, and authentication of the device configuration files. However, unlike the HWRoT mode, the EO mode provides integrity and authentication using the counter mode of symmetric AES (AES-GCM).

In this boot mode, all configuration images are encrypted (excluding the boot header). Given this exclusion, the boot header is subject to modification. Consequently, all security critical information contained in the boot header is ignored but replicated in eFUSEs. Modification of the boot header itself, while possible, achieves nothing as the eFUSEs are used for security critical decisions, not the boot header itself.

RECOMMENDED: *Xilinx strongly recommends the purchase of devices with built-in side channel leakage reduction functionality enabled. Protocol DPA countermeasures cannot be used in the encrypt-only boot mode due to the lack of asymmetric authentication. However, the built-in side channel leakage reduction capabilities can provide DPA resistance.*

The following figure shows a high-level view of the EO boot flow.

Figure 17: **Encrypt-only Boot Flow**



The RCU detects that the EO secure boot mode is enabled and then automatically decrypts the PLM and CDO. To maximize security, the key used by the RCU at this point is limited to the black key. This key is stored encrypted by the PUF key encryption key (KEK). No other key source is allowed for the PLM and CDO. The IV used at this point is programmed by the user into internal eFUSEs. As this is located inside the device, it is protected from modification.

Once the PLM and CDO have been decrypted and authenticated (via the AES GCM tag) and stored in internal memory, the RCU releases the reset to the specified processing unit. At this stage, control is handed over to the user application and it is responsible for maintaining the established security. All remaining partitions can use the following key sources on a partition-by-partition basis:

- Black eFUSE key
- User keys (volatile or non-volatile)

The IVs used for the remaining partitions are securely delivered within the configuration image. The process to load these partitions is user-configurable.

Configuration Update

As it is with the HWRoT mode, configuration update is critical for enabling features such as over-the-air (OTA) update. The configuration update flow for EO boot mode is no different than that of the HWRoT mode described in Figure 15: Configuration Update. However, this flow does have an additional setup step if it is necessary to update the PLM or CDO. This step involves programming additional eFUSEs in the IV space. This step prevents a system from reusing the same AES key/IV pair for different data. Such reuse is a violation of the AES standard. All remaining partitions requiring an update are done so by creating new configuration images encrypted using a different key/IV pair and loaded via the PLM no differently than was previously done.

Configuration Update with Partition Revocation

Key revocation, as described in the HWRoT secure boot mode, is not available in the EO secure boot mode. However, it is still important to support the revocation of individual partitions if an update is required and for protection against a rollback attack. In EO secure boot mode, rollback protection is achieved via the use of the revocation ID (stored in eFUSEs) associated with each partition. Revocations of keys might not be supported. However, it is possible to revoke the partition using the key desired to be revoked and replacing that partition by one encrypted with a different key/IV pair and a new (valid) revocation ID.

BootROM Error Codes

Any error that occurs during the BootROM execution and monitoring is recorded with a BootROM error code. The BootROM first and last error codes are 12 bits each and can be accessed from ERROR_STATUS [153:124] register read from the JTAG interface. See ERROR_STATUS Register for more information

The Versal ACAP BootROM error codes are listed in the following table.

Table 8: **BootROM Error Codes**

Error Code	Description
0x203	Secure boot not allowed in non-secure boot modes
0x204	Invalid boot mode read from BOOT_MODE_USER register
0x205	Image search error
0x206	Image/width in QSPI24 boot mode not detected
0x207	Image/width in QSPI32 boot mode not detected
0x209	eMMC FAT file system boot initialization error
0x20A	OSPI initialization error
0x213	SD0_LS FAT file system boot initialization error
0x214	SD1 FAT file system boot initialization error
0x215	SD1_LS FAT file system boot initialization error
0x21D	OSPI is not indicating idle during read operation
0x21F	OSPI command execution error during read operation
0x220	OSPI DMA read timeout error
0x300	Boot header does not have a XLNX signature
0x301	Invalid JTAG boot mode used for authenticated image
0x302	Invalid JTAG boot mode used for authenticated image
0x303	eFUSE and boot header authentication enabled
0x305	Boot image integrity check when authentication/encryption enabled error
0x307	SD/eMMC read error
0x308	Decryption with eFUSE only set in eFUSE and key source is not from eFUSE
0x309	Boot header source offset is overlapping with boot header
0x30A	Data partition or total data partition length is crossing the permissible limit of 112 KB
0x30B	Data partition or total data partition length is crossing the permissible limit of 384 KB
0x30C	Boot header image sync word (0xAA995566) does not match in SD/eMMC boot mode
0x30D	Image search not supported for slave boot modes

Table 8: **BootROM Error Codes** *(cont'd)*

Error Code	Description
0x30E	No image found in QSPI flash after searching the supported address range
0x30F	No image found in OSPI flash after searching the supported address range
0x310	No image found in SD/eMMC flash after searching the supported address range
0x311	Device read failed during the certificate read
0x312	Boot not allowed because all PPK revoked through eFUSE
0x313	Boot not allowed because all SPK revoked through eFUSE
0x314	Invalid PPK
0x315	Boot not allowed because chosen PPK is revoked through eFUSE
0x316	Invalid SPK
0x317	Boot not allowed because chosen SPK is revoked through eFUSE
0x318	PPK hash does not match any of the eFUSE locations
0x319	PLM length error
0x31A	Boot header authentication error
0x31D	PUF helper from boot header is not allowed when decryption through eFUSE only is set
0x31E	DPA counter measure enabled in boot header and disabled through eFUSE mismatch
0x31F	PMC firmware length is not 4-byte aligned
0x320	Key source changed from the previous to current image and authentication is not enabled
0x322	Data partition length is not 4-byte aligned
0x323	Source offset of PLM in image is not 4-byte aligned
0x324	Data partition load address in PMC RAM is not 16-byte aligned
0x325	Total data partition length is not 4-byte aligned
0x326	Total PLM length is not 4-byte aligned
0x327	Voltage glitch detected
0x328	PPK has a non-zero hash in the eFUSE
0x329	Error occurred reading the authentication certificate from flash
0x32A	Timeout occurred during SHA3 calculation for the authentication header
0x32B	Timeout occurred during SHA3 calculation for SPK
0x32C	Hash of SPK timeout error
0x32D	Timeout occurred during SHA3 calculation using DMA
0x32E	Hash of BH timeout error
0x32F	Timeout error occurred while calculating SHA3 using DMA
0x330	Hash of PPK timeout error
0x336	OSPI device not showing idle status after completion of read operation
0x337	OSPI idle check error before triggering DMA operation
0x338	OSPI idle check error after the DMA operation
0x339	Image header copy operation error after the boot header authentication to PPU RAM
0x33A	RSA operation timeout error
0x33B	RSA operation status error
0x33C	Encrypted signature invalid

Table 8: **BootROM Error Codes** *(cont'd)*

Error Code	Description
0x33D	M' hash does not match
0x33E	Invalid ECDSA key
0x33F	ECDSA signature verification error
0x340	QSPI DMA read operation timeout error
0x341	QSPI DMA read operation timeout error
0x342	QSPI DMA read operation timeout error
0x343	Data not received from host before timeout
0x344	SelectMAP abort sequence detected
0x345	Boot header PLM length is greater than the total PLM length
0x346	Total PLM length is less than the authentication certificate size
0x347	Data partition load address is not PMC RAM limits
0x348	Requested data partition cannot fit in PMC RAM
0x349	Total data partition length is less than data partition length
0x34A	Total data partition or data partition length mismatch
0x34B	Source offset of PLM in flash is beyond search limit
0x34C	Total data partition length does not match data partition length when authentication/ encryption/integrity is enabled
0x34D	JTAG boot timeout error
0x34E	DMA timeout error during SHA3 KAT operation
0x34F	Calculation timeout error during SHA3 KAT operation
0x350	KAT error during SHA3 operation
0x351	Key validation failed during KAT operation
0x352	KAT Error during ECDSA operation
0x353	KAT Error during RSA operation
0x354	HASH mismatch during RSA KAT operation
0x400	Invalid address for register initialization
0x401	Device read error after register initialization
0x402	Boot header does not match original after register initialization
0x403	Register initialization disabled through eFUSE
0x504	Boot image integrity error
0x505	Block size to be decrypted is not 128-bit aligned
0x506	Timeout error occurred before AES engine key load completed
0x507	Timeout error occurred during AES operation completed
0x508	DMA done not asserted after pushing the IV to AES engine with in timeout
0x509	DMA done not asserted after pushing the data to AES engine with in timeout
0x50A	DMA done not asserted after pushing the secure header to AES engine with in timeout
0x50B	DMA done not asserted after pushing the GCM tag to AES engine with in timeout
0x50C	DMA done not asserted after pushing the KEK to AES engine with in timeout
0x50D	Decrypted length does not match total image length specified in the boot header

Table 8: **BootROM Error Codes** *(cont'd)*

Error Code	Description
0x50E	Total decrypted length is greater than image size specified in the boot header
0x50F	GCM tag does not match for PLM decryption operation
0x510	GCM tag does not match for data partition decryption operation
0x511	Invalid key source
0x512	Invalid PUF command
0x513	Voltage glitch detected
0x514	Voltage glitch detected
0x515	PLM copy error occurred during boot image integrity check
0x516	PLM copy error occurred during boot image integrity check
0x517	PLM copy error occurred because authentication is enabled
0x518	PLM copy error occurred because decryption only or non-secure boot set
0x519	Data partition copy error occurred during boot image integrity check
0x51A	Data partition copy error occurred because authentication enabled
0x51B	Data partition copy error occurred because decryption only/non-secure boot set
0x51C	Timeout error occurred during SHA3
0x51E	Timeout error occurred during SHA3
0x51F	Timeout error occurred during SHA3
0x520	Timeout error occurred during SHA3
0x521	Timeout error occurred during SHA3
0x522	Timeout error occurred during SHA3
0x523	Timeout error occurred during SHA3
0x524	Timeout error occurred during SHA3
0x529	AES engine key or KUP key clearing error
0x52A	PUF key clear error
0x52B	Key load KAT error
0x52C	IV load KAT error
0x52D	Data load KAT error
0x52E	GCM tag load KAT error
0x52F	AES timeout KAT error
0x530	KAT GCM tag does not match
0x531	KAT decrypted data does not match original data
0x532	Key load error for counter measure enabled KAT
0x533	DMA timeout error for counter measure enabled KAT
0x534	AES timeout error for counter measure enabled KAT
0x53C	KEK load to AES engine error
0x53D	KEK IV load error
0x53E	Red key load from decrypted KEK error
0x53F	DPA counter measure KAT criteria does not match
0x540	DPA counter measure KAT criteria does not match

Table 8: **BootROM Error Codes** *(cont'd)*

Error Code	Description
0x541	DPA counter measure KAT criteria does not match
0x542	DPA counter measure KAT criteria does not match
0x543	DPA counter measure KAT criteria does not match
0x600	Voltage glitch detected
0x700	Error occurred with PUF disable
0x701	Error occurred with PUF regeneration disable
0x707	Timeout occurred before PUF word ready asserted
0x708	Timeout occurred before PUF key ready asserted
0x709	Read word not asserted by PUF during regeneration
0x70A	Timeout for PUF occurred before the key was ready
0x70B	Key not converged during regeneration
0x70C	PUF regeneration error
0x70D	PUF regeneration error
0x70E	Helper data in eFUSE is not valid so regeneration is not possible
0x710	Error occurred during PUF zeroization
0x711	PUF interrupt command is invalid
0x712	PUF interrupt NOOP command is not supported
0x722	DMA operation not completed before SHA3 calculation time
0x723	Timeout error occurred before SHA3 operation completed
0x730	Tamper event detected
0x731	BBRAM zeroization failed during tamper processing
0x747	PMC MBIST timeout error
0x748	Error occurred during PMC MBIST
0x749	Error occurred during PMC scan clear
0x74A	NoC scan clear error occurred during secure lock down
0x75A	PL scan clear timeout error
0x75B	Error occurred during PL scan clear
0x75C	VCCINT not detected
0x75D	Isolation error occurred between PMC and PL
0x75E	Error occurred during PL house-cleaning
0x75F	VCCINT not detected during PL house-cleaning
0x760	Isolation error between PMC and PL CFRAME occurred during PL house-cleaning
0x763	Key zeroization error occurred during secure lock down
0x800	SYSMON error

Boot Modes

The Versal™ ACAP boot modes are designed for maximum flexibility. This chapter provides a primary boot mode summary, selection considerations, and interface details. The primary boot modes in the PMC are:

- JTAG Boot Mode
- Quad SPI Boot Mode
- SD Boot Modes
- eMMC1 Boot Mode
- Octal SPI Boot Mode
- SelectMAP Boot Mode

Each boot mode uses a set of I/O pins and has a voltage requirement that can affect post-boot peripheral use of shared MIO on a bank. The best overall boot mode solution for an application considers the overall system requirements, performance, cost, and complexity.

The boot modes are categorized into master or slave boot modes. The master boot modes automatically load the programmable device image from a memory source (SD, eMMC, quad SPI, or octal SPI). The master boot modes provide a basic solution with easy setup. The slave boot modes require an external processor or controller to load the programmable device image with a command set (JTAG or SelectMAP). An advantage of using a slave boot mode is that the device image can reside almost anywhere in the host system or over a network connection. The slave boot modes are multipurpose interfaces that can also be used for system debug and readback.

RECOMMENDED: *Regardless of the boot mode selected, if the secure boot flow is not used, then JTAG connectivity on the board is recommended for the application. JTAG connectivity is a valuable debugging and bring-up interface.*

For systems that require a low-cost solution, QSPI boot modes are ideal with a variety of second-source vendors. For applications that require faster boot times due to power-on latency constraints, the boot modes with wide bus widths are inherently faster. For the master boot modes, the QSPI dual-parallel 8-bit or OSPI 8-bit is an optimal choice for a faster boot time. For slave-boot modes, the SelectMAP 32-bit mode provides the fastest option. For applications with large storage capacity requirements, the SD and eMMC1 boot modes support larger boot memory devices.

The Versal™Versal ACAP MIO-at-a-glance table should be reviewed to ensure that the requirements for boot and post-boot peripherals are satisfied. This chapter focuses on the primary boot mode options, however, the Versal ACAP is capable of starting with a primary boot mode and then switching to a secondary boot option (i.e., QSPI primary boot, followed by eMMC0 as a secondary boot option to provide larger density and flexibility). See the *Versal ACAP System Software Developers Guide* (UG1304) for more information on secondary boot options. The following table lists the available primary boot modes. Boot modes that are secure boot capable support both encrypt-only and hardware root of trust boot.

Table 9: **Primary Boot Modes**

Mode	MODE[3:0] Pins	PMC I/O Pins	Secure Boot Capable	Data Bus Width	Direction	Description
eMMC1 (4.51)	0110	MIO[12:3,0]	Yes	1-bit, 4-bit, 8-bit	Master	eMMC interface supports eMMC 4.51 at 1.8V
JTAG	0000	Dedicated I/O	No	1-bit	Slave	Dedicated JTAG interface
OSPI	1000	MIO[11:0]	Yes	8-bit	Master	Octal SPI interface supports single and dual-stacked flash devices
QSPI24	0001	MIO[12:0]	Yes	1-bit, 2-bit, 4-bit (single or dual-stacked) 8-bit (dual-parallel)	Master	Quad SPI interface supports the 24-bit (3-byte) flash addresses[1]
QSPI32	0010	MIO[12:0]	Yes	1-bit, 2-bit, 4-bit (single or dual-stacked) 8-bit (dual-parallel)	Master	Quad SPI interface supports the 32-bit (4-byte) flash addresses[1]
SD0 (3.0)	0011	MIO[49:37]	Yes	4-bit	Master	SD interface supports SD 3.0 with a required SD 3.0 compliant external level shifter
SD1 (2.0)	0101	MIO[51:50,33:28, 26]	Yes	4-bit	Master	SD interface supports SD 2.0
SD1 (3.0)	1110	MIO[51:50, 36:26]	Yes	4-bit	Master	SD interface supports SD 3.0 with a required SD 3.0 compliant external level shifter
SelectMAP	1010	MIO[51:28, 25:14]	Yes	8-bit, 16-bit, 32-bit	Slave	SelectMAP bidirectional parallel data bus interface

Notes:

1. For Quad SPI single flash or dual-stacked flash setups, only a subset of the MIO interface pins listed are required and the MIO interface pins can be used for other peripherals. See the boot interface diagrams for more information.

The ROM code unit (RCU) has a search limit to locate the device image boot header for every boot mode. The following table lists the boot image search limits for each mode.

Table 10: **Master Boot Mode Search Limit**

Boot Mode	Search Offset Limit
OSPI (single, dual-stacked)	8 Gb
QSPI24 (dual-parallel)	256 Mb
QSPI24 (single, dual-stacked)	128 Mb
QSPI32 (dual-parallel)	8 Gb
QSPI32 (single, dual-stacked)	4 Gb

Table 10: **Master Boot Mode Search Limit** *(cont'd)*

Boot Mode	Search Offset Limit
SD0 (3.0), SD1 (2.0), SD1 (3.0), or eMMC1	8191 FAT files (default)

***Note*:** When using OSPI or QSPI dual-stacked mode, the BootROM can only access the lower QSPI or OSPI addressable flash memory space for boot. After boot, the PLM can access the upper QSPI or OSPI for additional image storage.

JTAG Boot Mode

The JTAG interface is a multipurpose interface used for both boot and debug functions (PMC TAP JTAG operations, Arm® DAP debug, and interfaces to the debug packet controller for ChipScope™ solution debug). Due to this flexibility, the JTAG boot mode is popular for initial design bring-up and is a recommended interface for all applications. The JTAG boot mode uses only dedicated I/O allowing the PMC MIO to be used for peripheral system requirements. See Section XV: Test and Debug for information on JTAG instructions and JTAG chain.

Table 11: **JTAG Boot Mode Interface**

Pin Name	Pin Type	Direction	Description
TDI	Dedicated	Input	Test data input
TDO	Dedicated	Output	Test data output
TMS	Dedicated	Input	Test mode select
TCK	Dedicated	Input	Test clock

Quad SPI Boot Mode

The Versal ACAP supports a 24-bit addressing mode (QSPI24) boot mode or a 32-bit addressing (QSPI32) boot mode option. The QSPI32 boot mode option addresses flash sizes greater than 128 Mb. The QSPI boot mode supports multiple data bus widths and setups. In the QSPI boot modes, the BootROM runs at a QSPI device clock frequency between 11 MHz and 24.5 MHz dependent on the REF_CLK setting. For additional information on the quad SPI controller, see Quad SPI Controller.

The QSPI boot mode setups supported are listed in the following table.

Table 12: **Quad SPI Boot Mode Setups**

Quad SPI Setup	Flash Device Count	Chip Select Count	Data Width Max
Single (1-bit, 2-bit, 4-bit)	1	1	4
Dual-stacked[1] (1-bit, 2-bit, 4-bit)	2	2	4
Dual-parallel (8-bit)	2	2	8

Notes:

1. When using QSPI dual-stacked mode, the BootROM can only access the lower QSPI addressable flash memory space for boot. After boot, the PLM can access the upper QSPI for additional image storage.

The boot mode image search limits are listed in Table 10: Master Boot Mode Search Limit.

QSPI devices support different commands. The RCU supports a common subset of commands for boot as listed in the following table.

Table 13: **Quad SPI Commands Supported by the RCU**

Boot Mode	Data Width	Read Mode	Command Code	Dummy Cycles
QSPI24	1	Normal read (3-Byte)	03h	-
QSPI24	2	Dual Output fast read (3-Byte)	3Bh	8
QSPI24	4	Quad Output fast read (3-Byte)	6Bh	8
QSPI32	1	Normal read (4-Byte)	13h	-
QSPI32	2	Dual Output fast read (4-Byte)	3Ch	8
QSPI32	4	Quad Output fast read (4-Byte)	6Ch	8

Quad SPI Signals

The following table lists the bidirectional PMC multiplexed I/Os (MIOs) and their functions used in the Quad SPI boot mode setup.

Table 14: **Quad SPI Boot Mode Signals**

PMC MIO Pin	Signal Name	Description
0	QSPI0_CLK	QSPI0 clock output
4	QSPI0_IO[0]	I/O pin used as MOSI in 1-bit mode I/O pin used as the lower QSPI0_IO[0] in 2-bit or 4-bit single or dual-stacked setups, and in 8-bit dual-parallel setups
1	QSPI0_IO[1]	I/O pin used as MISO in 1-bit mode I/O pin used as the lower QSPI0_IO[1] in 2-bit or 4-bit single or dual-stacked setups, and in 8-bit dual-parallel setups
2	QSPI0_IO[2]	I/O pin used as the lower QSPI0_IO[2] in 4-bit single or dual-stacked setups, and in 8-bit dual-parallel setups
3	QSPI0_IO[3]	I/O pin used as the lower QSPI0_IO[3] in 4-bit single or dual-stacked setups, and in 8-bit dual-parallel setups
5	QSPI0_CS_b	Active-Low chip select output that enables QSPI0 (lower) flash device

Table 14: **Quad SPI Boot Mode Signals** *(cont'd)*

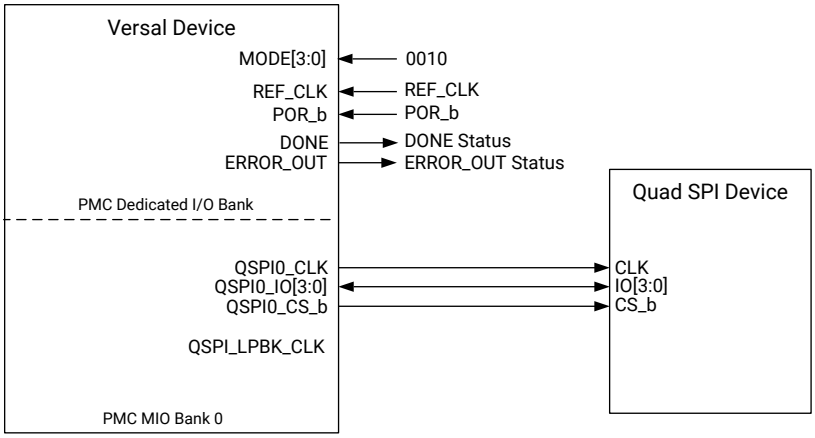
PMC MIO Pin	Signal Name	Description
12	QSPI1_CLK	QSPI1 clock output
8	QSPI1_IO[0]	I/O pin used as MOSI in 1-bit mode I/O pin used as the upper QSPI1_IO[0] in 2-bit or 4-bit dual-stacked setups, and in 8-bit dual-parallel setups
9	QSPI1_IO[1]	I/O pin used as MISO in 1-bit mode I/O pin used as the upper QSPI1_IO[1] in 2-bit or 4-bit dual-stacked setups, and in 8-bit dual-parallel setup
10	QSPI1_IO[2]	I/O pin used as the upper QSPI1_IO[2] in 4-bit dual-stacked setups, and in 8-bit dual-parallel setups
11	QSPI1_IO[3]	I/O pin used as the upper QSPI1_IO[3] in 4-bit dual-stacked setups, and in 8-bit dual-parallel setups
7	QSPI1_CS_b	Active-Low chip select output enables QSPI1 (upper) flash device
6	QSPI_LPBK_CLK	I/O pin used for loopback clock The loopback clock is an internal clock signal that is routed through the output buffer to this pin and returned back through the pin's input buffer to the quad SPI controller for I/O delay compensation. When the quad SPI device clock frequency is >37.5 MHz, the loopback clock must be enabled in the CIPS IP core and must be left connected on the board. When the quad SPI device clock frequency ≤37.5 MHz, the loopback clock should be disabled so MIO[6] is not used by the quad SPI controller and it can be used as another peripheral I/O.

Single Device Interface

The QSPI single-device mode is a common setup because it is low cost and has a lower pin count boot and configuration option. The QSPI single-device mode supports 1-bit, 2-bit, and 4-bit bus widths. This mode also supports 24-bit addressing and 32-bit addressing modes.

An example QSPI interface setup for a 4-bit bus width and 32-bit addressing mode is shown in the following figure.

Figure 18: **Quad SPI Single Device Interface**

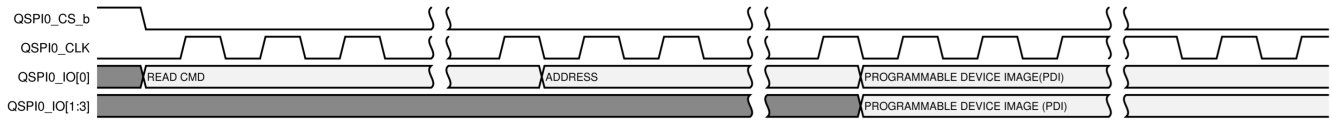


Note: For QSPI0_CLK >37.5 MHz, QSPI_LPBK_CLK must be enabled in the design and left unconnected on the board.

X22624-071220

The following figure shows an example QSPI read waveform with the relative sequence of events.

Figure 19: **QSPI Example Read Waveform**

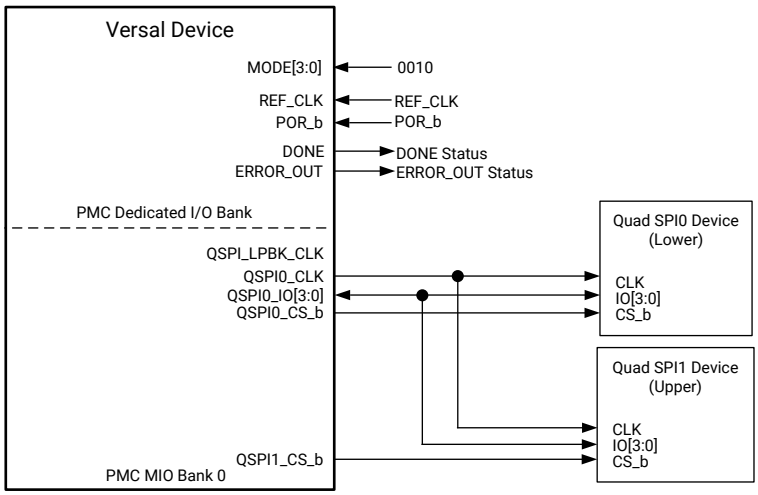


Dual-Stacked Interface

Two QSPI devices share the same bus in QSPI dual-stacked mode to double the maximum addressable flash memory storage for the application. This mode also reduces the boot interface I/O pin count because the bus is shared and only one additional interface pin is needed for the flash select. In this mode, only the lower QSPI addressable flash memory space can be used for boot and the throughput remains the same as it is in the QSPI single-device mode.

An example of the dual-stacked QSPI setup is with the 32-bit addressing mode is shown in the following figure.

Figure 20: **Dual-Stacked Quad SPI Interface Example**



Note: For QSPI0_CLK >37.5 MHz, QSPI_LPBK_CLK must be enabled in the design and unconnected on the board.

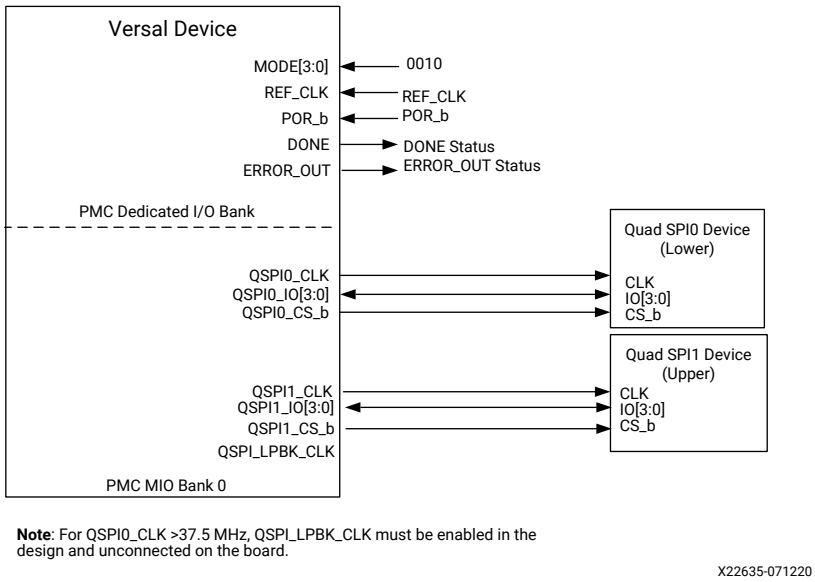
X22635-071220

Dual-Parallel Interface

The QSPI dual-parallel mode increases performance by combining two QSPI devices with 4-bit data widths to create an 8-bit data bus width. The QSPI dual-parallel mode only supports each QSPI device in a 4-bit data bus width. The QSPI dual-parallel mode does not support QSPI devices in 1-bit or 2-bit data bus widths.

An example of the dual-parallel QSPI setup with the 32-bit addressing mode is shown in the following figure.

Figure 21: **Dual-Parallel Quad SPI Interface Example**



Note: For QSPI0_CLK >37.5 MHz, QSPI_LPBK_CLK must be enabled in the design and unconnected on the board.

X22635-071220

SD Boot Modes

There are two SD/eMMC controllers on the Versal ACAP. The SD/eMMC controllers can be used for SD or eMMC and are mutually exclusive. When the controller is selected to support SD, the eMMC mode cannot be used.

The SD/eMMC controllers support three SD boot modes with different MIO pin usage. The SD1 (2.0) boot mode supports the SD2.0 specification. The SD1 (3.0) and SD0 (3.0) boot modes support the SD3.0 specification with an external SD3.0 compliant voltage-level shifter. In the SD boot modes, the RCU BootROM runs at an SD device clock frequency between 8.7 MHz and 19.3 MHz dependent on the REF_CLK setting. FAT 16/32 file systems are supported for reading the boot images.

The image search limit for SD boot mode is listed in Table 10: Master Boot Mode Search Limit.

For additional information on the SD/eMMC controller, see SD/eMMC Controller.

SD Signals

The following table lists the bidirectional PMC multiplexed I/Os (MIOs) and their functions used in the SD boot mode setup.

Table 15: **SD1 (2.0) Boot Mode Signals**

PMC MIO Pin	Signal Name	Description
26	SD1_CLK	SD1 clock output
29	SD1_CMD	SD1 command
30	SD1_DATA[0]	Data pin used in SD1 boot mode
31	SD1_DATA[1]	Data pin used in SD1 boot mode
32	SD1_DATA[2]	Data pin used in SD1 boot mode
33	SD1_DATA[3]	Data pin used in SD1 boot mode

***Note*:** For SD1 (2.0) boot mode, the BootROM does not support the SD1_WP, SD1_DETECT, or SD1_BUSPWR optional signals.

Table 16: **SD1 (3.0) Boot Mode Signals**

PMC MIO Pin	Signal Name	Description
26	SD1_CLK	SD1 clock output
29	SD1_CMD	SD1 command
30	SD1_DATA[0]	Data pin used in SD1 boot mode
31	SD1_DATA[1]	Data pin used in SD1 boot mode
32	SD1_DATA[2]	Data pin used in SD1 boot mode
33	SD1_DATA[3]	Data pin used in SD1 boot mode
34	SD1_SEL	Select signal output is automatically asserted when SD3.0 mode is selected Select signal enables an external voltage translator to switch from 3.3V to 1.8V to operate the SD card at the highest performance supported
35	SD1_DIR_CMD	DIR CMD output determines if the command is an input or output
36	SD1_ DIR0	DIR0 output determines if Data[0] is an input or output
27	SD1_DIR1	DIR1 output determines if Data[3:1] direction is an input or output
51	SD1_BUSPWR	Bus power output is an optional power pin that can be used to gate or reset the SD card power on the board
50	SD1_WP	Write protect input is an optional write protect signal
28	SD1_DETECT	Detect input is an optional card detect input that reflects the state of the mechanical switch on the SD card

Table 17: **SD0 (3.0) Boot Mode Signals**

PMC MIO Pin	Signal Name	Description
38	SD0_CLK	SD0 clock output
40	SD0_CMD	Command signal
41	SD0_DATA[0]	Data pin used in SD0 boot mode
42	SD0_DATA[1]	Data pin used in SD0 boot mode
43	SD0_DATA[2]	Data pin used in SD0 boot mode
44	SD0_DATA[3]	Data pin used in SD0 boot mode

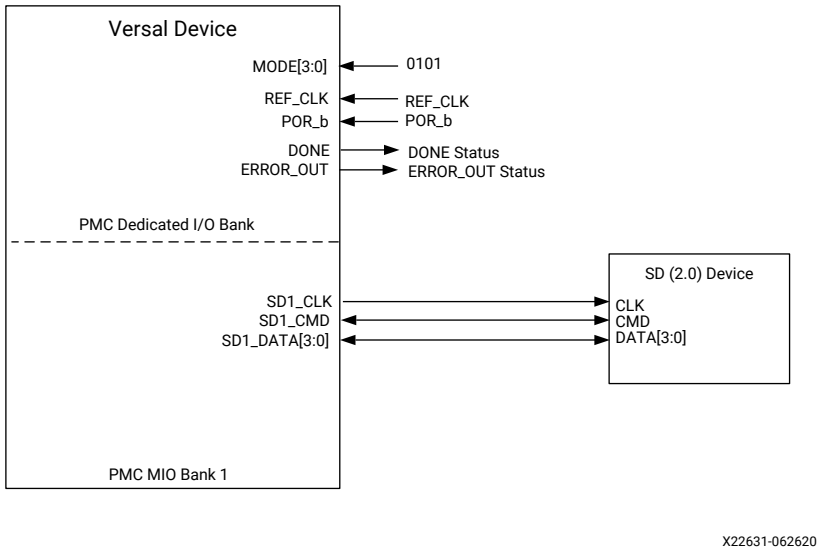
Table 17: **SD0 (3.0) Boot Mode Signals** *(cont'd)*

PMC MIO Pin	Signal Name	Description
45	SD0_SEL	Select signal is automatically asserted when SD3.0 mode is selected Select signal enables an external voltage translator to switch from 3.3V to 1.8V to operate the SD card at the highest performance supported
46	SD0_DIR_CMD	DIR_CMD output, determines if the command is an input or output
47	SD0_DIR0	DIR0 output determines if Data[0] is an input or output
48	SD0_DIR1	DIR1 output determines if Data[3:1] direction is an input or output
49	SD0_BUSPWR	Bus power output is an optional power pin that can be used to gate or reset the SD card power on the board
37	SD0_WP	Write protect input is an optional write protect signal
39	SD0_DETECT	Detect input is an optional card detect input that reflects the state of the mechanical switch on the SD card

SD2.0 Interface

The following figure shows the SD1 (2.0) boot mode interface from a single SD flash device and expects the PMC_MIO Bank1 to be at 3.3V.

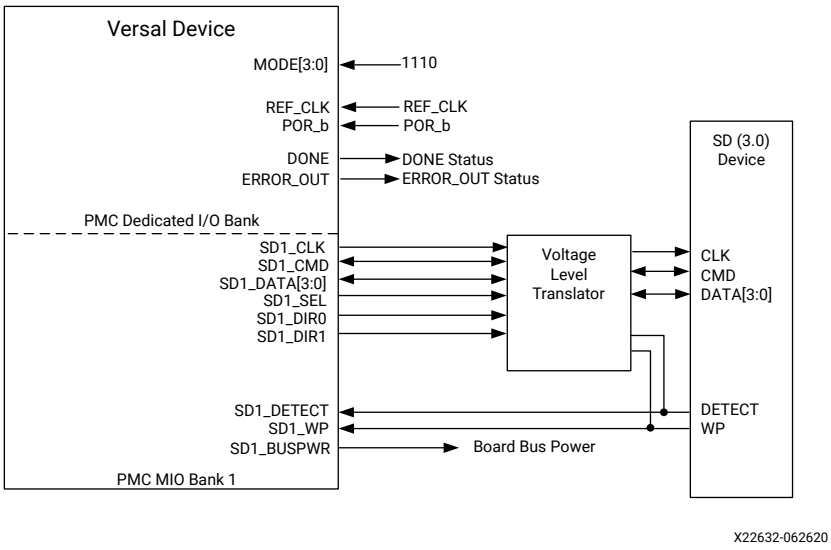
Figure 22: **SD 2.0 Interface**



SD3.0 Interface

The following figure shows the SD1 (3.0) boot mode interface from a single SD flash device and voltage level translator. This example expects the PMC_MIO Bank1 to be at 1.8V.

Figure 23: **SD1 (3.0) Interface Example**



This figure shows the requirement of a voltage level translator. Voltage level translators can provide ESD protection to the SD1_DETECT and SD1_WP optional signals and to pull-ups. The SD0 (3.0) controller interface setup is the same except the boot mode setting is MODE[3:0]=`0011` and the SD0 named signals are used instead of SD1.

eMMC1 Boot Mode

There are two SD/eMMC controllers on the Versal ACAP. The SD/eMMC controllers can be used for SD or eMMC and are mutually exclusive. When the controller is selected to support eMMC the SD mode cannot be used. Only one of the SD/eMMC controllers supports the eMMC boot mode, the SD1/eMMC1 controller. The SD1/eMMC1 controller supports the 4.51 eMMC specification and FAT 16/32 file systems for reading the boot images from eMMC. The image search limit for eMMC1 boot mode is listed in Table 10: Master Boot Mode Search Limit.

In the eMMC1 boot mode, the RCU BootROM runs at an eMMC1 device clock frequency between 8.7 MHz and 19.3 MHz dependent on the REF_CLK setting. The eMMC1 boot mode supports 1.8V and 1-bit, 4-bit, and 8-bit data interfaces. The BootROM uses auto-width detection to determine the data bus width for initial boot. The auto-bus width detection starts by checking the 8-bit data bus width, followed by 4-bit data bus width, and then 1-bit data bus width.

***Note*:** When connecting to the eMMC1 controller using a 1-bit data bus width, note the detection order because the remainder of the 8-bit data bus MIO data pins toggle during the initial bus-width detection.

For additional information on the SD/eMMC controller, see SD/eMMC Controller.

eMMC1 Signals

The following table lists the bidirectional PMC multiplexed I/Os (MIOs) and their functions used in the eMMC boot mode setup.

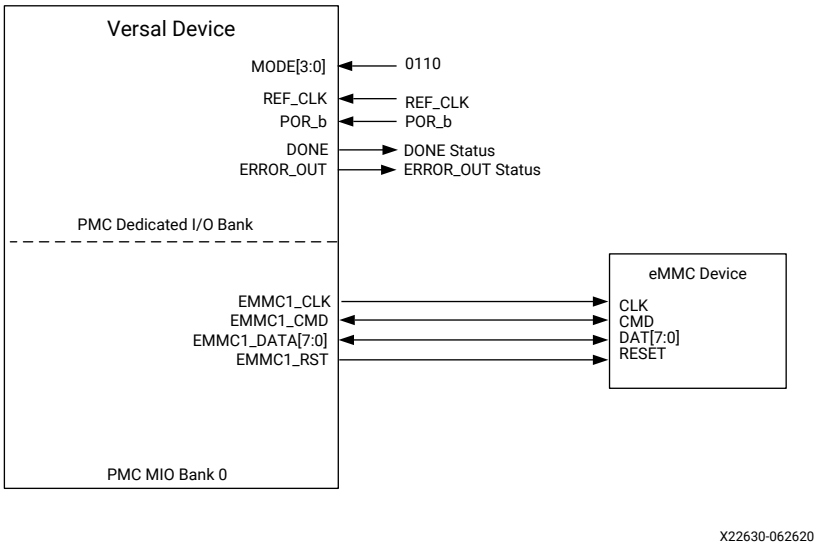
Table 18: **eMMC1 Boot Mode Signals**

PMC MIO Pin	Signal Name	Description
0	EMMC1_CLK	eMMC1 clock output
3	EMMC1_CMD	eMMC1 command
4	EMMC1_DATA[0]	Data pin used in eMMC1 boot mode (1-bit, 4-bit, 8-bit)
5	EMMC1_DATA[1]	Data pin used in eMMC1 boot mode (4-bit, 8-bit)
6	EMMC1_DATA[2]	Data pin used in eMMC1 boot mode (4-bit, 8-bit)
7	EMMC1_DATA[3]	Data pin used in eMMC1 boot mode (4-bit, 8-bit)
8	EMMC1_DATA[4]	Data pin used in eMMC1 boot mode (8-bit)
9	EMMC1_DATA[5]	Data pin used in eMMC1 boot mode (8-bit)
10	EMMC1_DATA[6]	Data pin used in eMMC1 boot mode (8-bit)
11	EMMC1_DATA[7]	Data pin used in eMMC1 boot mode (8-bit)
12	EMMC1_RST	Reset output that resets the eMMC flash

eMMC1 Interface

The following figure shows an example setup for eMMC1 boot mode from a single flash device.

Figure 24: **eMMC1 Interface Example**



Octal SPI Boot Mode

The octal SPI (OSPI) boot mode has an SPI compatible serial bus interface with extended octal commands. The OSPI boot mode supports an 8-bit data bus width and single transfer rate (STR) during the RCU BootROM execution. The BootROM runs at an OSPI device clock frequency between 11 MHz and 24.5 MHz dependent on the REF_CLK setting. After the BootROM execution, the PLM can support the double data rate (DDR) with strobe for higher performance. The OSPI boot mode can be configured to a OSPI single or dual-stacked setup. For additional information on the OSPI controller, see Octal SPI Controller.

Note: When using OSPI dual-stacked mode, the BootROM can only access the lower OSPI0 addressable flash memory space for boot. After boot, the PLM can access the upper OSPI1 for additional image storage.

The following table lists the STR OSPI commands supported by the RCU BootROM.

Table 19: **OSPI Commands Supported by RCU for Boot**

Boot Mode	Data Width	Read Command	Command Code	Dummy Cycles
OSPI	1	Read	03h	-
OSPI	1	4-byte read	13h	-
OSPI	8	4-byte octal output fast read	7Ch	8

In OSPI boot mode, the device initiates the boot sequence with the default 4-byte address octal output fast read command code 7Ch and the BootROM searches for a valid boot header. If a valid boot header is not found, the Versal ACAP attempts to load the image using the 4-byte alternate addressing read command code 13h. If a valid boot header is still not detected, the basic read command 03h is tried. If the boot attempt is unsuccessful after the third command, the BootROM increments the image header address register by 32 KB and tries the OSPI command sequence again to locate a valid boot header. If the OSPI boot mode search limit is reached without a successful boot, the RCU goes into lockdown and the ERROR_OUT pin is set.

The image search limit for each boot mode is listed in Table 10: Master Boot Mode Search Limit.

Octal SPI Signals

The following table lists which bidirectional PMC multiplexed I/O (MIO) are used and their function in the Octal SPI boot mode setups.

Table 20: **Octal SPI Boot Mode Signals**

PMC_MIO Pin	Signal Name	Description
0	OSPI_CLK	OSPI clock output for OSPI0 in single setup, or OSPI clock output for OSPI0 and OSPI1 in dual-stacked setup.

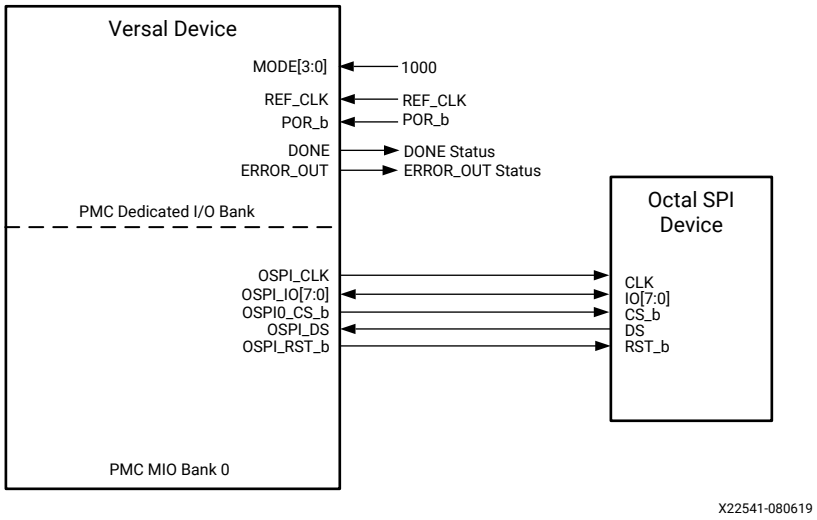
Table 20: **Octal SPI Boot Mode Signals** *(cont'd)*

PMC_MIO Pin	Signal Name	Description
1	OSPI_IO[0]	Data pin used for OSPI single or dual-stacked boot mode setup
2	OSPI_IO[1]	Data pin used for OSPI single or dual-stacked boot mode setup
3	OSPI_IO[2]	Data pin used for OSPI single or dual-stacked boot mode setup
4	OSPI_IO[3]	Data pin used for OSPI single or dual-stacked boot mode setup
5	OSPI_IO[4]	Data pin used for OSPI single or dual-stacked boot mode setup
7	OSPI_IO[5]	Data pin used for OSPI single or dual-stacked boot mode setup
8	OSPI_IO[6]	Data pin used for OSPI single or dual-stacked boot mode setup
9	OSPI_IO[7]	Data pin used for OSPI single or dual-stacked boot mode setup
10	OSPI0_CS_b	Active-Low select output enables OSPI0 (lower) flash device
11	OSPI1_CS_b	Active-Low select output enables OSPI1 (upper) flash device, used in dual-stacked setup
6	OSPI_DS	Data strobe input, supports the DDR option in octal SPI boot mode. The octal SPI compatible flash must support SDR at power on. During the RCU boot phase initial checks SDR is required, then the PPU can switch the flash to DDR mode for faster boot time.
12	OSPI_RST_b	Active-Low reset output to reset the OSPI flash. The OSPI_RST_b signal must be connected to the OSPI flash. The PMC_GPIO channel is assigned to the OSPI flash reset pin MIO[12]. When the PMC RCU detects the octal SPI boot mode, it asserts and then deasserts the MIO[12] pin to the reset during boot to bring the device to a default stated.

Single Device Interface

The following figure shows an example OSPI setup for boot from a single flash device.

Figure 25: **Single Octal SPI Interface Example**

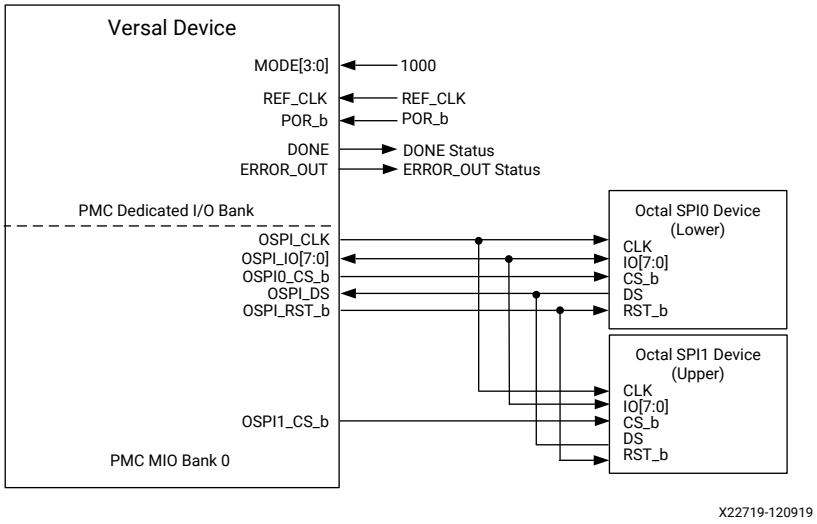


Dual-Stacked Interface

In OSPI boot mode, the dual-stacked configuration uses two OSPI flash devices to double the maximum addressable flash memory storage for the application. The OSPI dual-stacked configuration requires one additional chip select pin compared to the OSPI single configuration. In this mode, only the lower OSPI addressable flash memory space can be used for boot and the throughput remains the same as it is in the OSPI single-device mode.

An example of the dual-stacked OSPI setup with the 32-bit addressing mode is shown in the following figure.

Figure 26: **Dual-Stacked Octal SPI Interface Example**



SelectMAP Boot Mode

The SelectMAP boot mode supports a 8-bit, 16-bit, or 32-bit bidirectional data bus interface. This mode can boot and configure a single Versal ACAP or multiple Versal ACAPs. In this mode, an external processor or controller drives the SelectMAP data, clock, and control signals (read/write and chip select). The external processor also needs to monitor the BUSY signal for SelectMAP boot initiation and flow control.

When the SelectMap boot mode is detected during system start-up, the path to receive data from the SelectMAP interface is configured. The PMC controllers and blocks used to enable the path for SelectMAP are highlighted in Platform Management Controller.

The RCU is responsible for enabling the path for the SelectMAP boot interface. The RCU configures the slave boot interface, the secure stream switch (SSS), and dedicates the PMC DMA1 to the slave boot interface (SBI). The PMC MIO SelectMAP 32-bit pins are enabled and are placed into input mode. After the RCU configures the SBI control register the BUSY signal is deasserted. After power-up, when BUSY is deasserted, this indicates that the Versal ACAP is ready to receive data from the SelectMAP interface host.

Because the BUSY signal can be asserted at any stage during boot and configuration, this signal must be monitored to ensure the interface is ready to accept data. When the BUSY signal is asserted it indicates the chip select must be deasserted to stop the data loading within 24 clock (SMAP_CLK) cycles or the SBI FIFO (8 KB) buffer used for SelectMAP data processing overflows.

For SelectMAP, the I/O configuration default is a weak pull-up. When connecting to SelectMAP data bus width 8-bit, 16-bit, or 32-bit only the selected bus width data signals are used for boot and configuration.

SelectMAP Pattern and Bit Order

The Versal ACAP programmable device image (PDI) boot header is read by the RCU BootROM to determine the SelectMAP bus width. The first 16 bytes in the PDI boot header determine the SelectMAP bus width.

The SelectMAP bus detection PDI pattern options include:

8-bit bus width	00	00	00	DD	11	22	33	44	55	66	77	88	99	AA	BB	CC
16-bit bus width	00	00	DD	00	22	11	44	33	66	55	88	77	AA	99	CC	BB
32-bit bus width	DD	00	00	00	44	33	22	11	88	77	66	55	CC	BB	AA	99

SelectMAP Bit Order

The SelectMAP interface is typically driven by a user application residing on a microprocessor, microcontroller, or another FPGA or SoC. For these applications, it is important to understand how the data ordering in the programmable device image corresponds to the data ordering expected by the Versal ACAP interface. In SelectMAP 8-bit mode, the programmable device image data is loaded at one byte per clock with the bits of each byte presented to the SelectMAP pins. The following table shows how to load the SelectMAP PDI data bits onto the SelectMAP data pins.

Table 21: **SelectMAP Bit Order**

SelectMAP Signal Names (SMAP_IO[#])	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
32-bit PDI data order	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	23	22	21	20	19	18	17	16	31	30	29	28	27	26	25	24
16-bit PDI data order																	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8

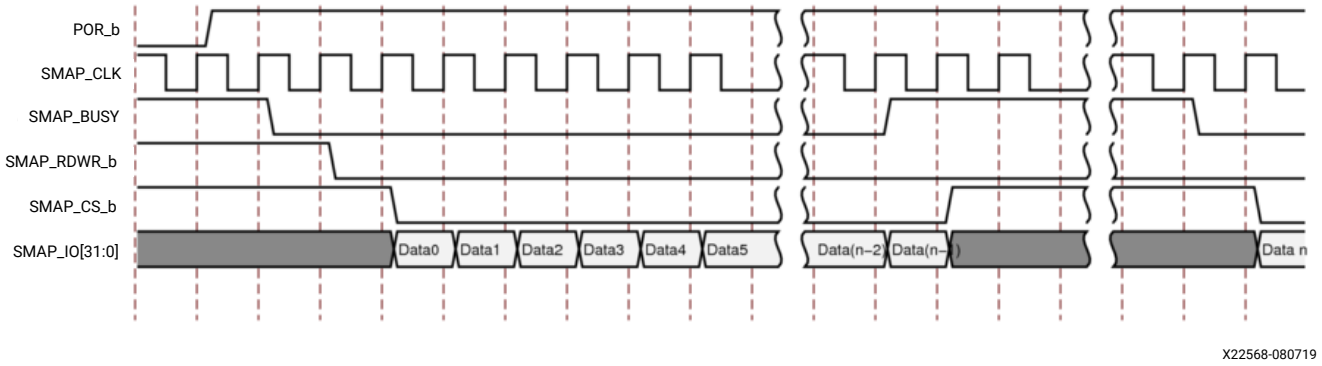
Table 21: **SelectMAP Bit Order** *(cont'd)*

SelectMAP Signal Names (SMAP_IO[#])	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
8-bit PDI data order																									7	6	5	4	3	2	1	0

SelectMAP Sequence

The SelectMAP interface allows an external processor to load the boot and configuration data. The functional waveform in this section shows an example of the SelectMAP interface data, clock, and control signals operation to load data into the Versal ACAPs. The waveform shows an example BUSY response. The BUSY response time can vary but must be within 24 SMAP_CLK cycles. BUSY is clocked by the SMAP_CLK and does not transition back to low if the SMAP_CLK is stopped.

Figure 27: **SelectMAP Data Loading**



SelectMAP Signals

The following table lists the bidirectional PMC multiplexed I/Os (MIOs) and their functions used in the SelectMAP boot mode setup.

Table 22: **SelectMAP Boot Mode Signals**

PMC_MIO Pin	Signal Name	Description
18	SMAP_CLK	SelectMAP clock output
14	SMAP_IO[0]	Data pin used in SelectMAP boot mode (8-bit, 16-bit, 32-bit)
15	SMAP_IO[1]	Data pin used in SelectMAP boot mode (8-bit, 16-bit, 32-bit)
16	SMAP_IO[2]	Data pin used in SelectMAP boot mode (8-bit, 16-bit, 32-bit)
17	SMAP_IO[3]	Data pin used in SelectMAP boot mode (8-bit, 16-bit, 32-bit)
22	SMAP_IO[4]	Data pin used in SelectMAP boot mode (8-bit, 16-bit, 32-bit)
23	SMAP_IO[5]	Data pin used in SelectMAP boot mode (8-bit, 16-bit, 32-bit)

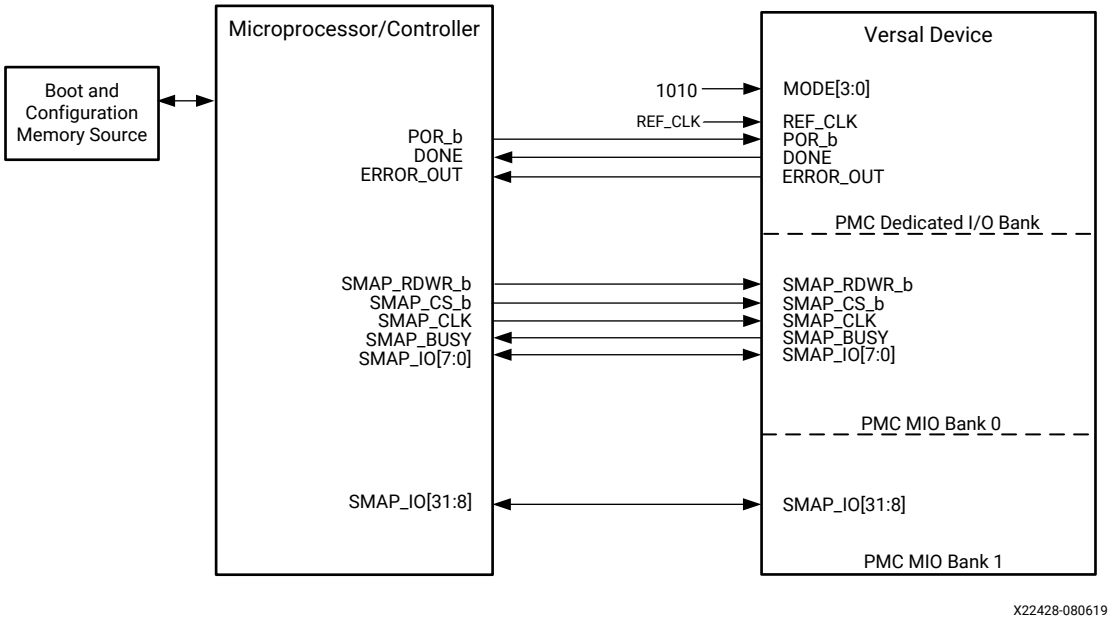
Table 22: **SelectMAP Boot Mode Signals** *(cont'd)*

PMC_MIO Pin	Signal Name	Description
24	SMAP_IO[6]	Data pin used in SelectMAP boot mode (8-bit, 16-bit, 32-bit)
25	SMAP_IO[7]	Data pin used in SelectMAP boot mode (8-bit, 16-bit, 32-bit)
28	SMAP_IO[8]	Data pin used in SelectMAP boot mode (16-bit, 32-bit)
29	SMAP_IO[9]	Data pin used in SelectMAP boot mode (16-bit, 32-bit)
30	SMAP_IO[10]	Data pin used in SelectMAP boot mode (16-bit, 32-bit)
31	SMAP_IO[11]	Data pin used in SelectMAP boot mode (16-bit, 32-bit)
32	SMAP_IO[12]	Data pin used in SelectMAP boot mode (16-bit, 32-bit)
33	SMAP_IO[13]	Data pin used in SelectMAP boot mode (16-bit, 32-bit)
34	SMAP_IO[14]	Data pin used in SelectMAP boot mode (16-bit, 32-bit)
35	SMAP_IO[15]	Data pin used in SelectMAP boot mode (16-bit, 32-bit)
36	SMAP_IO[16]	Data pin used in SelectMAP boot mode (32-bit)
37	SMAP_IO[17]	Data pin used in SelectMAP boot mode (32-bit)
38	SMAP_IO[18]	Data pin used in SelectMAP boot mode (32-bit)
39	SMAP_IO[19]	Data pin used in SelectMAP boot mode (32-bit)
40	SMAP_IO[20]	Data pin used in SelectMAP boot mode (32-bit)
41	SMAP_IO[21]	Data pin used in SelectMAP boot mode (32-bit)
42	SMAP_IO[22]	Data pin used in SelectMAP boot mode (32-bit)
43	SMAP_IO[23]	Data pin used in SelectMAP boot mode (32-bit)
44	SMAP_IO[24]	Data pin used in SelectMAP boot mode (32-bit)
45	SMAP_IO[25]	Data pin used in SelectMAP boot mode (32-bit)
46	SMAP_IO[26]	Data pin used in SelectMAP boot mode (32-bit)
47	SMAP_IO[27]	Data pin used in SelectMAP boot mode (32-bit)
48	SMAP_IO[28]	Data pin used in SelectMAP boot mode (32-bit)
49	SMAP_IO[29]	Data pin used in SelectMAP boot mode (32-bit)
50	SMAP_IO[30]	Data pin used in SelectMAP boot mode (32-bit)
51	SMAP_IO[31]	Data pin used in SelectMAP boot mode (32-bit)
19	SMAP_CS_b	Chip select input enables the SelectMAP bus When CS_b is Low, the SelectMAP interface is enabled When CS_b is High, the SelectMAP interface is disabled
20	SMAP_RDWR_b	Read/Write input that controls whether the data pins are inputs or outputs When RDWR_b is High, data is output or read onto the SelectMAP data bus When RDWR_b is Low, an external controller can write data or boot and configure the device through the SelectMAP data bus interface
21	SMAP_BUSY	Busy output is High when there are 24 clock cycles left before the slave boot interface (SBI) FIFO data buffer overflows

Single Device Interface

The SelectMAP mode single device option uses an external processor or controller to provide the clock, read/write enable, chip select, and data, as well as monitors the busy signal for flow control to boot and configure a Versal ACAP. As shown in the following figure, this high-bandwidth interface spans multiple banks. The PMC MIO bank0 and bank1 must be powered at the same voltage. Performance, bank voltage, and MIO usage should be evaluated when selecting the boot mode.

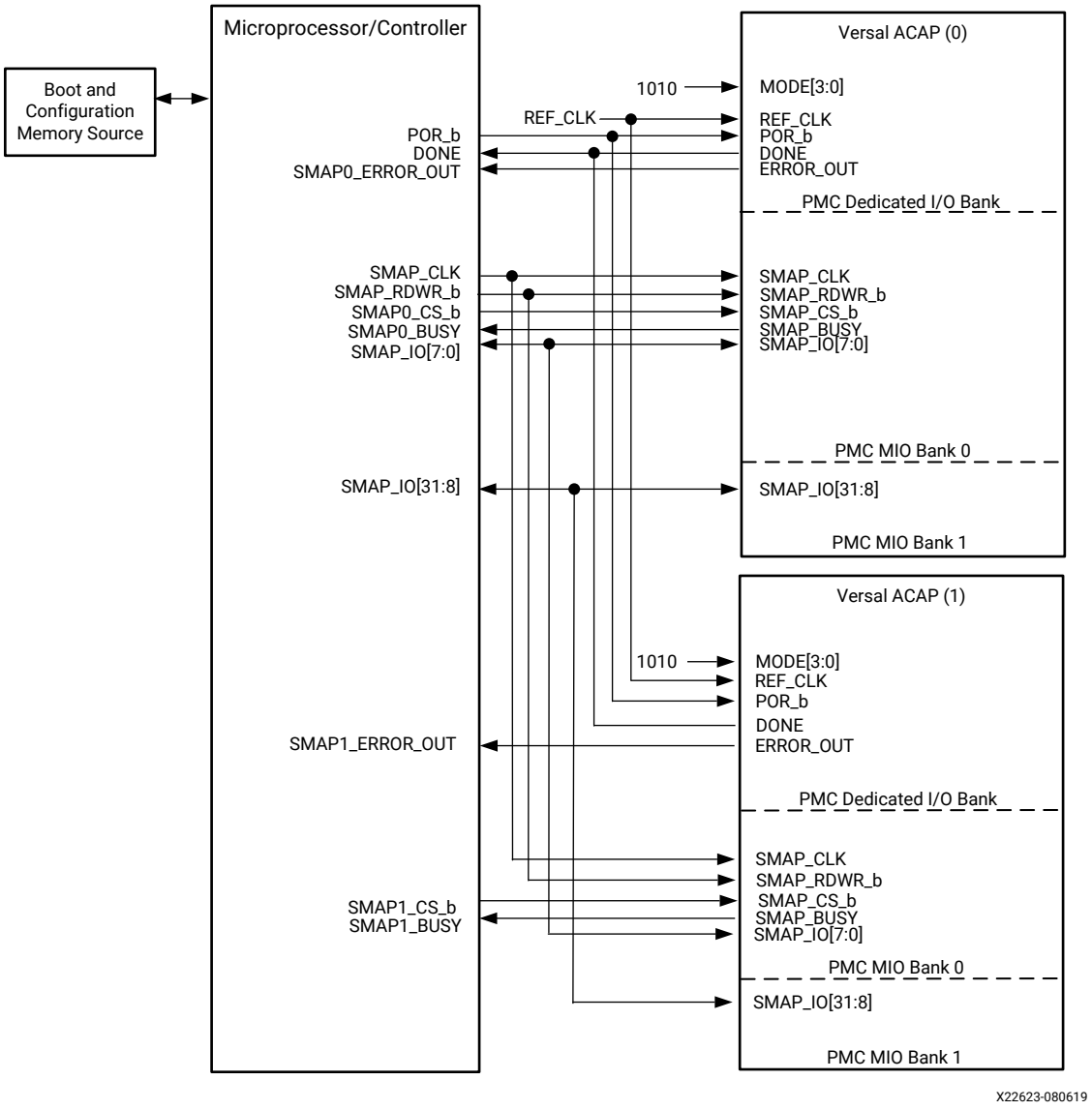
Figure 28: **SelectMAP Single Device Interface Example**



Multiple Device Interface

The SelectMAP mode multiple device option uses an external processor or controller to provide the clock, read/write enable, two chip selects, and data, as well as monitors two busy signals for flow control to boot and configure multiple Versal ACAPs with different images. As shown in the following figure, this high-bandwidth multiple device interface spans multiple banks. The PMC MIO bank0 and bank1 on both Versal ACAPs must be powered at the same voltage. Performance, bank voltage, and MIO usage should be evaluated when selecting the boot mode.

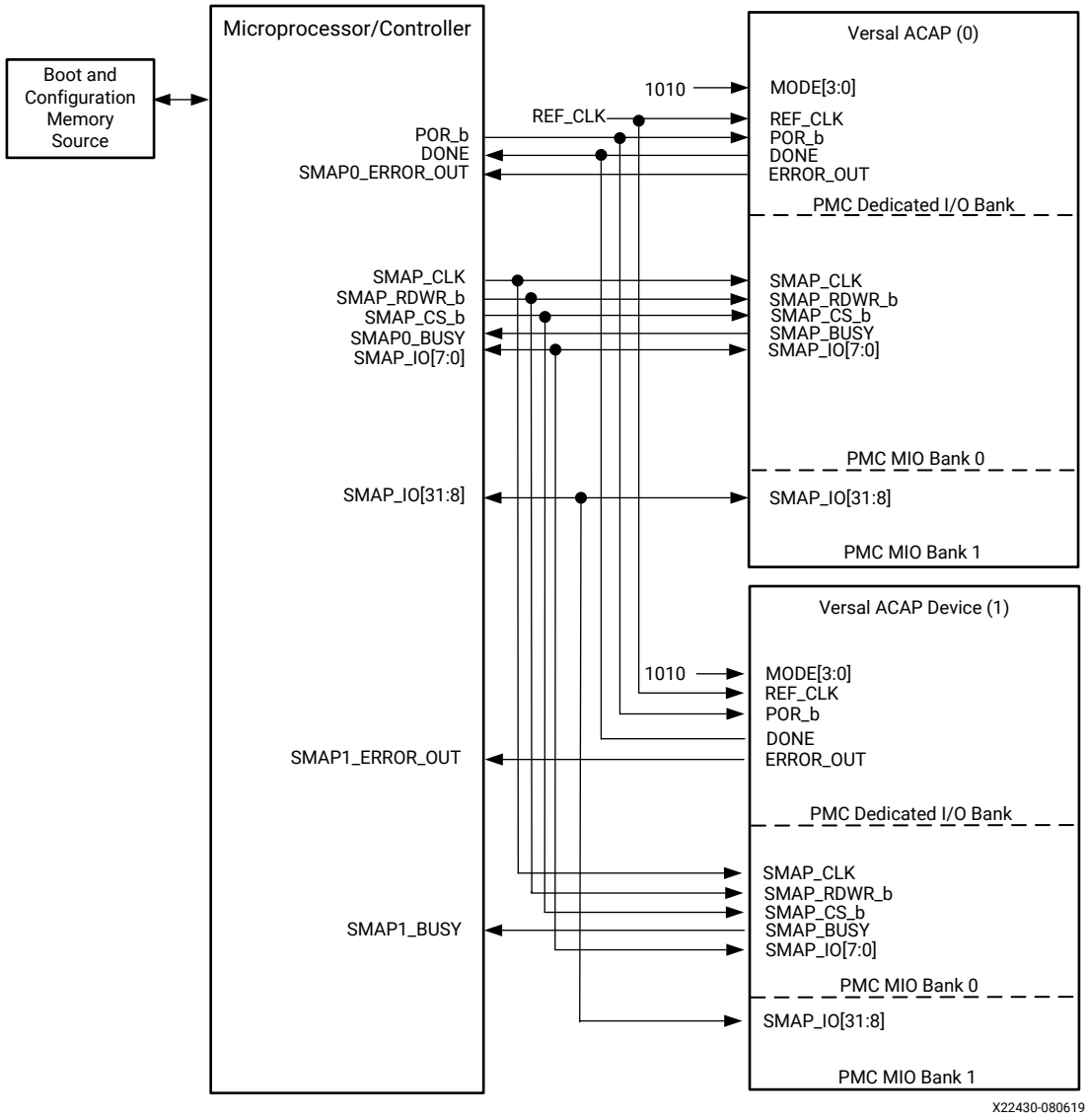
Figure 29: **SelectMAP Multiple Device Interface Example**



Ganged Device Interface

In SelectMAP ganged, an external processor or controller provides the clock, read/write enable, chip select, and data, as well as monitors the busy signal for flow control to boot and configure multiple Versal devices with the same image in parallel. As shown in the following figure, this high-bandwidth multiple device interface spans multiple banks. The PMC MIO bank0 and bank1 on both Versal devices must be powered at the same voltage. Performance, bank voltage, and MIO usage should be evaluated when selecting the boot mode.

Figure 30: **SelectMAP Ganged Interface Example**



XILINX

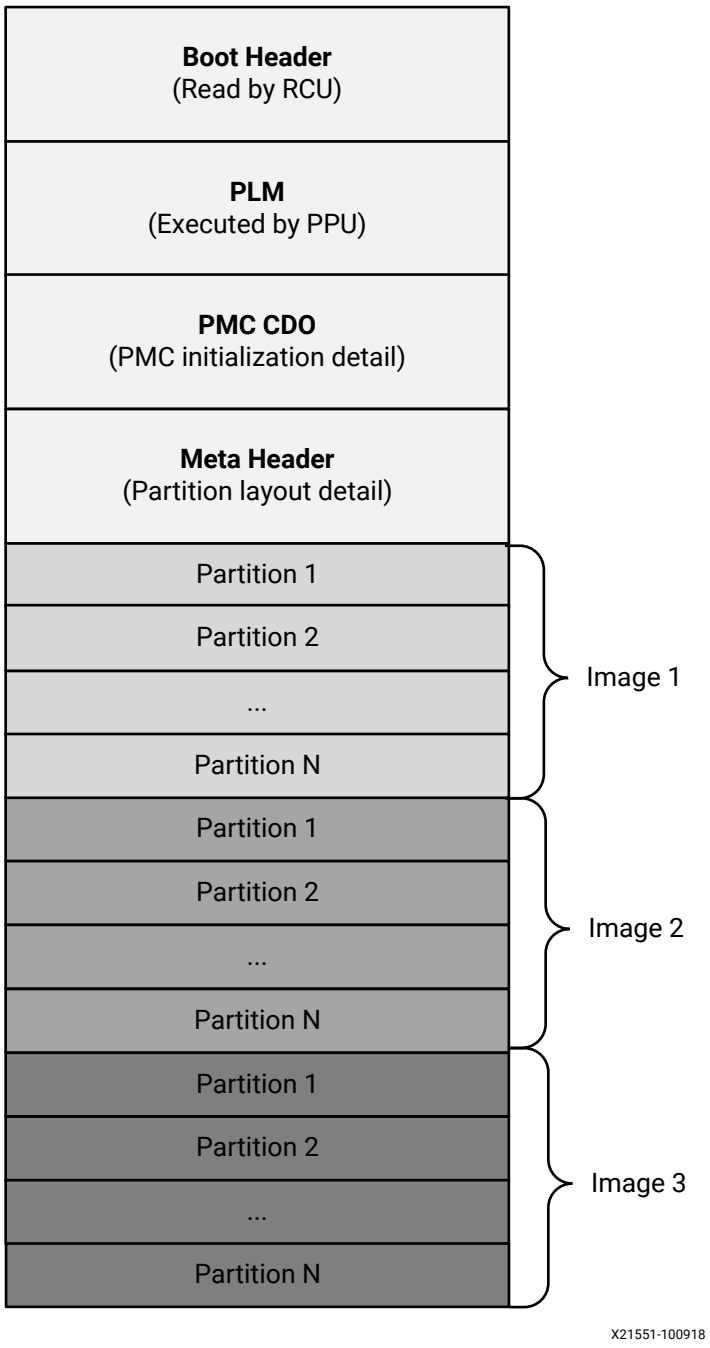
Chapter 19

Boot Image - Programmable Device Image

The platform management controller (PMC) uses a proprietary programmable device image (PDI) file format to boot and configure the Versal™ ACAP. The Vivado®/Vitis™ development environment generates the PDI image. For more information on creating the PDI image and composition, see the *Bootgen User Guide* (UG1283). The image is composed of the following partitions:

- Boot header includes image identification, platform loader and manager (PLM) partition size and offset, and other attributes for the image
- PLM (PPU execution code)
- PMC configuration data object includes topology and initialization information on PMC and LPD commands for block configuration
- Meta header includes image and partition headers that contain image partition count and location information, as well as image and partition ID
- Images and partitions include PS (APU, RPU), PL, and AI Engine data or software images and the corresponding configuration data object

Figure 31: **Programmable Device Image (PDI) Format**



Boot Header

The boot header is in the programmable device image (PDI). The PDI boot header is read by the ROM code unit (RCU) to determine information such as the platform loader and manager (PLM) location and size, the boot mode bus width, and security encryption key details. The boot header format information is listed in the following table. For additional information on the PDI format, meta header, and partition header format, see *Bootgen User Guide* (UG1283).

Table 23: **Boot Header Format**

Offset (Hex)	Size (Bytes)	Description	Details
0x00	16	SelectMAP bus width	Used to determine if the SelectMAP bus width is x8, x16, or x32 See SelectMAP Pattern and Bit Order for the unique entries.
0x10	4	QSPI bus width	QSPI bus width description This is required to identify the QSPI flash in single/dual stacked or dual parallel mode. 0xAA995566 in the little endian format.
0x14	4	Image identification	Boot image identification string Contains 4 bytes X, N, L, X in byte order, which is 0x584c4e58 in the little endian format.
0x18	4	Encryption key source	This field is used to identify the AES key source: 0x00000000 - Unencrypted 0xA5C3C5A3 - eFUSE red key 0xA5C3C5A5 - eFUSE black key 0x3A5C3C5A - BBRAM red key 0x3A5C3C59 - BBRAM black key 0xA35C7C53 - Boot header black key
0x1C	4	PLM source offset	PLM source start address in PDI
0x20	4	PMC data load address	PMC CDO address to load
0x24	4	PMC data length	PMC CDO length
0x28	4	Total PMC data length	PMC CDO length including authentication and encryption overhead
0x2C	4	PLM length	PLM original image size
0x30	4	Total PLM length	PLM image size including the authentication and encryption overhead
0x34	4	Boot header attributes	Boot header attributes
0x64	12	Secure header IV	Secure header initialization vector
0x70	4	PUF shutter value	Length of time the PUF samples before it closes the shutter
0x74	12	Secure header IV for PMC data	The IV used to decrypt secure header of PMC data
0x80	68	Reserved	Populate with zeroes
0xC4	4	Meta header offset	Offset to the start of the meta header
0xC8-0x124	88	Reserved	

Table 23: **Boot Header Format** *(cont'd)*

Offset (Hex)	Size (Bytes)	Description	Details
0x128	2048	Register init	Stores register write pairs for system register initialization
0x928	1544	PUF helper data	PUF helper data
0xF30	4	Checksum	Header checksum
0xF34	76	SHA3 padding	SHA3 standard padding

XILINX

Chapter 20

Platform Management

The PLM with its libraries support many run-time services. See the *Versal ACAP System Software Developers Guide* (UG1304) for more information on the run-time services. This chapter highlights the hardware features provided to support these example services:

- Functional safety management
- Dynamic Function eXchange (DFX)
- Power management
- Security management
- Soft error mitigation

Functional Safety Management

The functional safety of a system or part of a system refers to the correct operation of the system in response to its input, which includes management of errors, hardware failure, and changes to operating conditions. The two types of faults that can cause a system failure and result in a violation of the functional safety goals are systematic faults and random faults.

Systematic faults arise from errors in the development or manufacturing processes. When defects appear in hardware or software, they are systematic faults. Some of the causes of systematic faults are a failure to verify intended functionality, manufacturing test escapes, or operating a device outside of a specified range. The mitigation of systematic faults is achieved by robust best practices and processes defined by safety standards.

Random faults are inherent due to silicon aging, environmental conditions, etc. Safety standards focus on detecting and managing random faults.

This chapter provides an overview of the safety mechanisms implemented in the Versal™ ACAP. The features are grouped into these categories:

- Single point fault detection
- Common cause failure detection
- Latent fault detection
- Isolation features

- Additional features

Single Point Fault Detection

The detection of single point faults is supported with these features:

- ECC protection for OCM, PPU RAM, PMC RAM, RPU L1 cache, and TCM memories
  - Address decode error detection
  - Separate RAMs for ECC syndrome and data
  - 4:1 or greater interleaving of memory cells protected by ECC
- Hash validation of RCU ROM contents at every boot
- Lockstep and redundancy covers Cortex-R5F processor
  - Lockstep with physical and temporal diversity
  - Redundant logic in critical control logic including the lockstep checkers
- PPU and RCU controllers are implemented with redundancy
  - MicroBlaze TMR (triple modular redundancy) cores with physical diversity
  - Triple redundant flip-flops for critical control bits such as security state
- XMPU and XPPU protect memory space
- Windowed watchdog timers in LPD and FPD

Common Cause Failure Detection

The detection of common cause faults is supported with these features:

- System monitoring
  - Voltage monitoring
  - Temperature monitoring
  - Clock frequency monitoring
- Error management
  - Error management is handled and implemented within the PPU
  - Errors can be signaled as interrupts and mirrored to the PL
- Monitoring of activation of common cause failures (CCF) by PPU
  - MBIST
  - SCAN

- Reset, power control
- Hang protection
  - Cleanup of outstanding transactions under partial reset
- Aging errors
  - Large on-chip variation (OCV) margin to account for aging effects

Latent Fault Detection

The detection of latent faults is supported with these features:

- All check logic such as XMPU, lockstep, and ECC checkers are checked at boot by LBIST
- All LPD memories can be tested at full-processing speed during boot by MBIST
  - Most of these memories (excluding PPU and RCU RAMs) can be tested on demand during execution
- The functionality and status of TCM, OCM, PPU RAM, RPU lockstep, PPU, XMPU, XPPU, clocks, voltages, and temperatures can be tested and evaluated through the dedicated software test library (STL)

Isolation Features

Power domain isolation is supported with these features:

- LPD supports isolation from the rest of the system
- Flexible reset management
  - Enables use of the controllers for redundant processing
  - Reset management is implemented in the PPU
  - Independent reset for LPD, FPD, PL, and PS only
- Independent power domains
  - PMC, LPD, FPD, and PL
- Built-in AXI timeout on PL master interfaces

Additional Features

The following features provide additional safety support:

- DDR interface supports ECC for 32-bit and 64-bit words
  - Double-error detection

    - Single-error correction
- ECC in APU L2-cache
- ECC in L1 data cache memory
- Parity in L1 instruction cache memory
- QOS management
    - QOS controls on masters
    - QOS management in PS AXI
    - QOS management in DDR memory controller
- Leverage of PL for implementation of safety features
    - Provides HFT channel capability
    - Provides error logging
    - PL can remain active if PS is reset due to an error

Dynamic Function eXchange

The Dynamic Function eXchange (DFX) is managed by PLM services running in the PMC. These services control the events needed before, during, and after dynamic reconfiguration of the NPI and CFI resources throughout the device. These events include controlling the isolation of the target region, unloading (and loading) of software drivers (as appropriate for modified applications), delivery of programming images from any secondary boot interface, and image authentication and integrity checking before programming is done. All boot modes can be used for partial device image delivery.

Power Management

The Versal ACAP includes two dedicated controllers, the platform management controller (PMC) and the PSM controller.

The PMC facilitates the isolation of the power domains outside of the processing system (PS) full-power domain (FPD). The PMC is used for power, error management, and the execution of an optional software test library (STL) for functional safety applications.

The processing system manager (PSM) controller has the hardware interfaces to manage the isolation for the PS FPD. The PSM controller serves as a PMC proxy to the PS power islands. The PSM directly controls the power islands within the PS FPD.

The primary power domains for power management are listed in the following table.

Table 24: **Primary Power Domains**

Power Domain	Description
PMC power domain	Includes the RCU, PPU, PMC flash controllers, PMC I2C controller, PMC GPIO controller and is the core domain for device start-up
Low-power domain (LPD)	Includes the RPU, PSM, SPI controller, LPD I2C controllers, LPD GPIO controller, UART controller, USB controller, PS gigabit ethernet MAC, and CAN FD controller
Full-power domain (FPD)	Includes the APU
Battery power domain (BPD)	Includes the real-time clock core and the battery-backed RAM
System power domain (SPD)	Includes the NoC, NPI, and the DDR controller, the XPIO rail is tied to this domain
PL power domain	Includes the programmable logic, GT, AI Engine, CPM, and XPIPE

Power Modes

The modes for power management operation are outlined in this section. To comply with the power domain requirements, there are separate power rails to supply the power for each domain. The following PS power modes can be integrated with various power modes, including but not limited to PL on, PL off, and PL clock gated.

Sleep Modes

To meet the requirements of a very low sleep-power state, the device provides sleep and deep sleep modes. The PMC domain is always on. The PMC can be active during the sleep mode. The low-power domain (LPD) can be on or off in some sleep modes. The RTC or GPIO wake sources are supported for deep sleep. The USB and PS GEM blocks in the low-power domain support wake on USB and wake on LAN for sleep. The following table lists sleep mode examples.

Table 25: **Sleep Modes (No APU/RPU Processing)**

Power Mode	Description
Deep sleep	LPD is off FPD is off PMC is wake on RTC, GPIO, or USB
LPD_Off_FPD_Off_PMC_Active	LPD is off FPD is off PMC power domain is active and the PMC can be processing
Deep sleep - fast resume	LPD is on but the R5s are off FPD is off PMC is wake on RTC, GPIO, or USB
Sleep - GigE	LPD is on but the R5s are off FPD is off PMC is wake on GigE (PS GEM)

Low-Power Modes

In the low-power operation mode, the RPU is idle or in wait for an interrupt. With the LPD, integrated blocks on the low-power rail can be powered up in the PS block (RPU, TCM, OCM, and PSM). The PSM assists the PMC for events local to the PS. The LPD includes additional peripherals for low-power operation. The following table lists low-power mode examples.

Table 26: **Low-Power Modes (FPD Off, RPU Processing)**

Power Mode	Description
R5s_Idle_FPD_Off_DDR_Off	RPU cores are idle FPD is off DDR is off
R5s_Idle_FPD_Off_DDR_Self_Refresh	RPU cores are idle FPD is off DDR is in self-refresh mode to maintain memory
R5s_Idle_FPD_Off	RPU cores are idle FPD is off DDR is on
R5s_Active_FPD_Off	RPU cores are both active FPD is off DDR is on

Full-Power Modes

All domains are powered in the full-power mode. Power dissipation depends on the components that are running and their frequencies. The following table lists full-power mode examples.

Table 27: **Full-Power Modes (FPD On, APU/RPU Processing)**

Power Mode	Description
Linux boot idle	RPU cores are idle One APU core is off and one is idle
R5s_Idle_1_A72_250MHz	RPU cores are idle One APU core is off and one is running at 250 MHz
R5s_Idle_1_A72_Active	RPU cores are idle One APU core is off and one is active
R5s_Idle_A72s_Active	RPU cores are idle Both APU cores are active
Performance mode - R5s active, A72s active	RPU cores are both active Both APU cores are active

Security Management

The increasing ubiquity of Xilinx devices makes protecting the intellectual property (IP) within them as important as protecting the data processed by the device. As security threats have increased, the range of security threats or potential weaknesses that must be considered to deploy secure products has grown as well. The Versal ACAP provides features to help secure applications. These features include:

- Tamper monitoring and response
- Secure key storage and management
- User access to Xilinx hardware cryptographic accelerators

See Secure Boot Flow for details on system start-up secure boot flow.

Tamper Monitoring and Response

Versal ACAPs provide a range of anti-tamper features to help secure applications and manage potential security threats. The Versal ACAP hardware provides features that not only detect security intrusions but also allow a response with selected penalties. This tamper resistance protection needs to be effective during all four of the system start-up phases, which include pre-boot (PMC hardware, power-up and reset), boot setup (RCU, initialization and boot header processing), load platform (PPU, boot image processing and configuration), and post-boot (PPU, platform management and monitoring services).

Sensitive data can include the software and configuration data that sets up the functionality of the device logic, critical data, or parameters that might be included in the boot image (for example, initial memory contents and initial state). It also includes external data that is dynamically transported in and out of the device during the post-boot operation.

The primary function of the RCU post-boot is to monitor the system for tamper events. There are different monitoring functions that can be configured, including:

- System Monitor (SYSMON) triggering limits for voltage and temperature alarms are user-defined and configured. The tamper registers generate an over and under temperature alarm when the SYSMON unit “threshold mode” is set to 1.
- RCU can act as a centralized tamper monitor and response hub for a system.
- System extensible using MIO to trigger an external tamper event.
- Detection of power supply glitches.
- Detection of activity on debug ports (such as JTAG).

Secure Key Storage and Management

The Versal ACAP AES-GCM cryptographic engine has access to a diverse set of key sources. Non-volatile key sources include eFUSE, BBRAM, and PUF key encryption key (KEK). These keys maintain their values even when the device is powered down. Volatile key sources include a boot header (BH) key and a key update (KUP) register key.

The device provides a variety of options for securing both boot images and user data. Boot image keys can be stored in BBRAM, eFUSE, or in the boot image itself. These keys can be in plain text (red) or encrypted with the PUF KEK (black).

Table 28: **General Key Terms**

Key Name	Description
Device	Symmetric key that is stored on the device (eFUSE, BBRAM, boot header)
PPK: Primary public key	Public key for asymmetric authentication, used to authenticate the secondary public key
SPK: Secondary public key	Public key for asymmetric authentication, used to authenticate partitions
AES	Symmetric key used for AES encrypt/decrypt

The following table provides the different key options and includes a 32-bit key selection code that specifies which key is used by the AES core.

Table 29: **Key Sources**

Key Name	Source	Key Selection	Size (bits)	Description
BBRAM	BBRAM	0xBBDE6600	256	The BBRAM key is used to store an AES key for boot. This key can be protected by the PUF KEK.
BH	Register	0xBDB06600	256	The BH (boot header) key is stored encrypted inside the programmable device image (PDI) boot header and once decrypted it is stored inside the BH key register.
EFUSE	eFUSE	0xEFDE6600	256	The eFUSE key is used for boot and is stored in the eFUSEs. It can be plain text or encrypted with the PUF KEK.
EFUSE_USER (x2)	eFUSE	0xEF856601 0xEF856602	256, 128	The two eFUSE user keys are key storage available for user run-time keys and stored in eFUSE.
Key update register (KUP)	Register	0xBDC98200	256,128	Key source used when key rolling is employed. The next user defined block of data is stored in the KUP.
PUF KEK	PUF	0xDBDE8200	256	The PUF KEK is a key-encryption key that is generated by the PUF.

Table 29: **Key Sources** *(cont'd)*

Key Name	Source	Key Selection	Size (bits)	Description
USER (x8)	Register	0xBD858201 0xBD858202 0xBD858204 0xBD858208 0xBD858210 0xBD858220 0xBD858240 0xBD858280	256, 128	Write only registers available for holding user run-time keys. Each register can be individually locked.

The MSB of the 32-bit key selection code decodes to:

- BB – BBRAM (battery-backed RAM)
- BD – BH/KUP/USER key
- EF – EFUSE/EFUSE_USER
- DB –PUF key

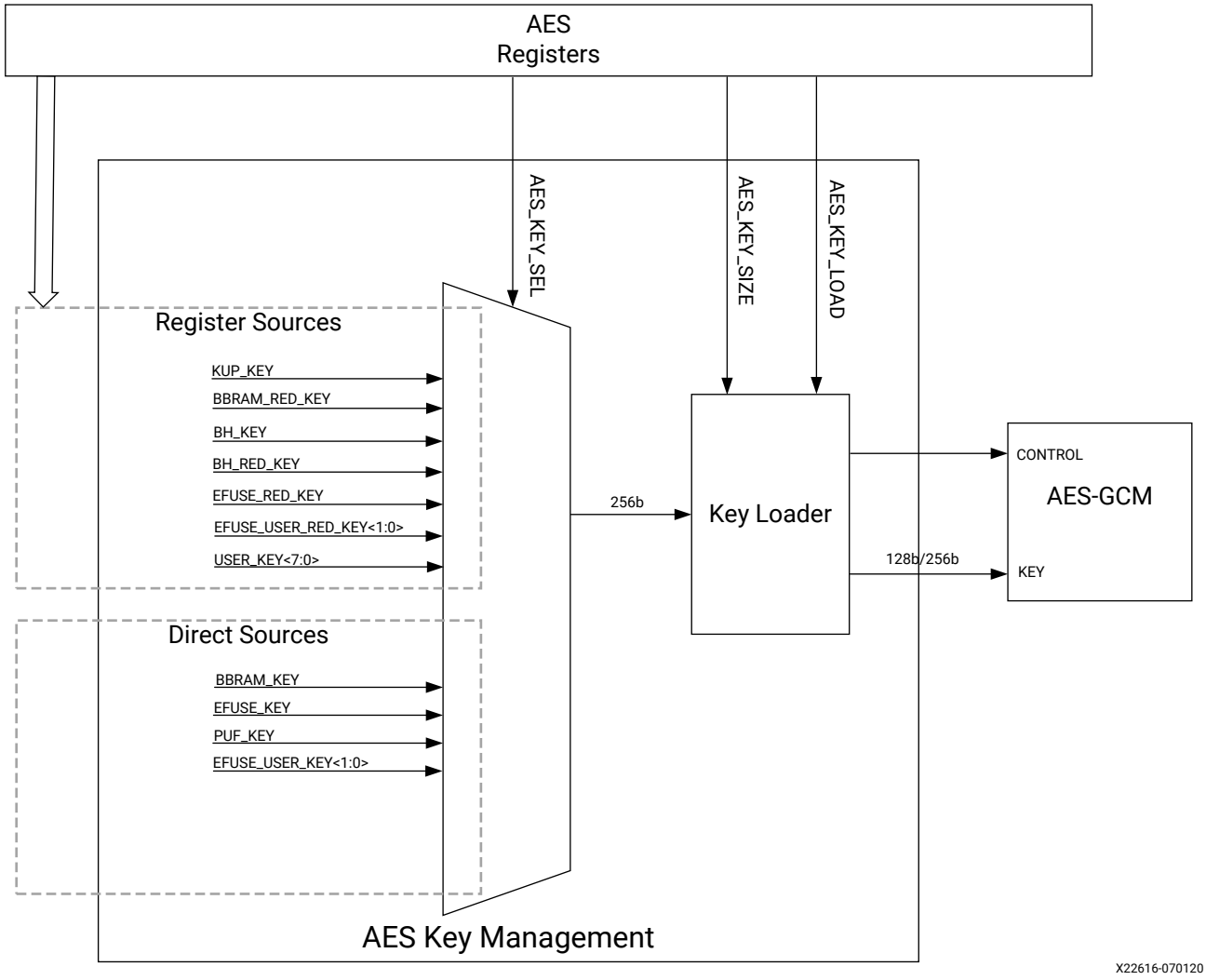
Key Selection

The device key source selection is exclusively performed by the RCU ROM based on the authenticated boot image header. The AES key management block selects the appropriate key that needs to be input to the AES core.

The AES_KEY_SEL register determines which of the keys available is used for the encryption and decryption operation. The AES_KEY_SIZE register determines if the key is 128 bits or 256 bits. The AES_KEY_LOAD register loads the key value into the AES core.

In addition to the BBRAM and eFUSE key storage locations, the Versal ACAP also allows for the device key to be stored externally in the boot flash. This key can be stored in its black form (i.e., encrypted with the PUF KEK).

Figure 32: **AES Key Selection**



Battery-Backed RAM Key

The battery-backed RAM (BBRAM) is one of the available options for storing the device AES key. The BBRAM is a static RAM array. When the device has power on the VCCAUX_PMC supply, the BBRAM is powered by the VCCAUX_PMC supply. When the VCCAUX_PMC supply is switched off, the device automatically switches the BBRAM power domain (BPD) using the VCC_BATT pin. The key stored in BBRAM can be stored in plain text form (red) or encrypted form (black). The BBRAM can also be cleared, which is valuable as a tamper response.

The BBRAM key memory space is 288 bits. The BBRAM contains 256 bits of the AES key and 32 bits for general purpose usage. The 256-bit AES key is available for the ROM based on its selection. When the AES key in BBRAM is selected as the AES device key, the AES key management block loads the key value into the AES.

eFUSE Key

The eFUSE array contains a block of 256 eFUSEs that can provide a key to the AES-256 cryptographic engine. This block of eFUSEs has dedicated read and write disables controlled by additional eFUSEs. The eFUSE key can be stored in plain text form (red) or encrypted form (black).

Key Update Register

The key update register is used during boot to support the key rolling feature, where a different AES key must be loaded multiple times. A 256-bit KUP key is stored in the eight AES key update registers.

Boot Header Key

The boot header (BH) key is stored encrypted inside the programmable device image (PDI) boot header and once decrypted, it is stored inside the BH key register.

Storing Keys in Encrypted Form (Black)

The black key storage solution uses a cryptographically strong key encryption key (KEK) generated from a physical unclonable function (PUF) to encrypt the user key. The resulting black key can then be stored either in eFUSEs or as part of the authenticated boot header resident in external memory. The black key storage provides these advantages:

- The user key is the same for all devices. Consequently, the encrypted boot images are the same for all devices that use the same user key.
- The PUF KEK is unique for each device. Consequently, the black key stored with the device is unique for each device.
- The PUF KEK value is only known by the device. There is no readback path and, consequently, cannot be read by the user.

Physically Unclonable Function

The Versal™ device contains a physically unclonable function (PUF). The PUF creates a signature (or fingerprint) of each device that is unique to that device. Its value is not “knowable” by Xilinx or the user enabling usage as a key encryption key (KEK).

The KEK is 256 bits in length with 256 bits of entropy and is used to encrypt the users red key allowing its storage in black (encrypted) form. The black key can be stored in either eFUSEs, BBRAM, or external storage.

Enhanced from the previous generation, the Versal device PUF also outputs a user accessible unique ID that is cryptographically isolated from the PUF KEK despite using the same entropy source. While unique to each device, it is not considered a “secret” and does not have the same access protections as the KEK.

The silicon manufacturing process includes inherent, random, and uncontrollable variations that cause unique and different characteristics from device to device. The Xilinx devices operate within these variations and device functionality is not affected. PUF includes tiny circuits that exploit these chip-unique process variations to generate unique keys. The type of PUF used to generate the KEK is also an important consideration. The Versal ACAP PUF uses an asymmetric technology that is different from the device key storage technology (e.g., SRAM or eFUSE). This asymmetric technology increases the security level above what can be achieved with a single technology.

The PUF uses approximately 4 Kb of helper data to help the PUF recreate the original KEK value over the guaranteed operating temperature and voltage range over the life of the part. The helper data consists of a syndrome, aux, and chash value. The helper data can either be stored in eFUSEs or in the boot image. The following table lists the PUF helper data.

Table 30: **PUF Helper Data**

Field	Size (Bits)	Description
Syndrome	4060	These bits aid the PUF in recovering the proper PUF signature given slight variations in the ring oscillators over temperature, voltage, and time
Aux	24	This is a Hamming code that allows the PUF to perform some level of error correction on the PUF signature
Chash	32	This is a hash of the PUF signature that allows the PUF to recognize if the regenerated signature is correct

Access to the PUF is restricted by the RCU. The PUF can be accessed through the RCU registers.

The PUF undergoes a registration process when a key is initially loaded into the device. The registration process initializes the PUF so that a KEK is created, and the following options are available.

- The registration software can then use the KEK to encrypt the user key and program the eFUSEs.
- The encrypted user key can be output for inclusion into a boot image. The registration software also programs the helper data into the eFUSEs.
- The helper data can be output for inclusion into a boot image.

The helper data and the encrypted user key must be stored in the same location (i.e., both in eFUSE or both in the boot image). When the device powers on, the RCU examines the boot image header (the boot image header is authenticated when authentication is enabled). The boot image header contains information on whether the PUF is used, where the encrypted key is stored (eFUSE or boot image), and where the helper data is stored (eFUSE or boot image). The RCU then initializes the PUF, loads the helper data, and regenerates the KEK. This process is called regeneration. After the KEK is regenerated, the RCU can use it to decrypt the user key, which is then used to decrypt the rest of the boot image.

Key Management Summary

The following table provides a key management summary for BBRAM, eFUSE, and boot image.

Table 31: **Key Management Summary**

Features	BBRAM	eFUSE	Boot Image
Programming method	• Internal via PLM library	• Internal via PLM library • External via JTAG PUF registration software	• Bootgen • Bootgen + PUF registration software
Program verification	CRC32 only	CRC32 only	N/A
Key state during storage	Red or black	Red or black	Black
In-use protections	• Temporary storage in registers, not RAM • Transferred in parallel not serial • DPA countermeasures		

User Access to Xilinx Hardware Cryptographic Accelerators

The Versal ACAP flexibility allows for many ways to implement cryptographic functions. The most common is software using the Arm® cryptographic extensions and custom accelerators built in the programmable logic (PL). The Versal ACAP built-in cryptographic accelerators can also be used to secure the device configuration. These cryptographic accelerators are available post-boot:

- AES-GCM
- RSA/ECDSA
- SHA3-384
- True Random Number Generator

Soft Error Mitigation

The Versal ACAP PMC hardware supports the ability to validate the integrity of the device configuration and perform readback of configuration data (in the background) using the Xilinx soft error mitigation library.

The Xilinx Soft Error Mitigation (XilSEM) library is a pre-configured, pre-verified solution to detect and optionally correct soft errors in the configuration memory of Versal ACAPs. A soft error is caused by ionizing radiation and is extremely uncommon in commercial terrestrial operating environments. While a soft error does not damage the device, it carries a small statistical possibility of transiently altering the device behavior.

The XilSEM library does not prevent soft errors; however, it provides a method to better manage the possible system-level effect. Proper management of a soft error can increase reliability and availability, and reduce system maintenance and downtime. In most applications, soft errors can be ignored. In applications where a soft error cannot be ignored, see the *Versal ACAP System Software Developers Guide* (UG1304) for additional information about the XilSEM library prior to configuring it for use through the CIPS IP core.

XILINX

Section IV

Address Maps and Programming Interfaces

This section includes a hierarchy of address maps and tables that summarize the register sets, and includes these chapters:

- Address Maps
- Programming Interfaces

Address Maps

The global address maps are based around the NoC interconnect and provide a high-level view. Each of the subsystems as individual address maps are:

- Global Address Map
- PMC and PS Address Map
- FPD Address Map
- LPD Address Map
- PMC Address Map

***Note*:** Not all devices have the locations listed in these maps. The address maps might vary by device series. See the Device Matrix table for a quick overview of what is included in each device series. For a comprehensive list, see the product Versal ACAP data sheets.

Global Address Map

The global address map is based on the NoC and spans 0 to 16 Terabytes (TB) as shown in the following table.

Table 32: **Global Address Map (0 to 16 TB)**

Destination	Size (GB)	Address Range		Notes
		Start	End	
0 to 4 GB	4	0x0000_0000_0000	0x0000_FFFF_FFFF	See PMC and PS Address Map
reserved	4	0x0001_0000_0000	0x0001_FFFF_FFFF	reserved
reserved	8	0x0002_0000_0000	0x0003_FFFF_FFFF	reserved
M_AXI_FPD	8	0x0004_0000_0000	0x0005_FFFF_FFFF	High address 0
PCIe	8	0x0006_0000_0000	0x0007_FFFF_FFFF	PCIe region 1
DDR	32	0x0008_0000_0000	0x000F_FFFF_FFFF	DDR channel 0 region 1
reserved	448	0x0010_0000_0000	0x007F_FFFF_FFFF	reserved
PCIe	256	0x0080_0000_0000	0x00BF_FFFF_FFFF	PCIe region 2
DDR	256	0x00C0_0000_0000	0x00FF_FFFF_FFFF	DDR channel 0 region 2
DDR	734	0x0100_0000_0000	0x01B7_7FFF_FFFF	DDR channel 0 region 3

Table 32: **Global Address Map (0 to 16 TB)** *(cont'd)*

Destination	Size (GB)	Address Range		Notes
		Start	End	
reserved	290	0x01B7_8000_0000	0x01FF_FFFF_FFFF	reserved
AI Engine	1	0x0200_0000_0000	0x0200_3FFF_FFFF	AI Engine interface tiles
reserved	2047	0x0200_4000_0000	0x03FF_FFFF_FFFF	reserved
reserved	1024	0x0400_0000_0000	0x04FF_FFFF_FFFF	reserved
DDR_CH1_L	512	0x0500_0000_0000	0x057F_FFFF_FFFF	DDR channel 1 low
DDR_CH1_H	512	0x0580_0000_0000	0x05FF_FFFF_FFFF	DDR channel 1 high
DDR_CH2_L	512	0x0600_0000_0000	0x067F_FFFF_FFFF	DDR channel 2 low
DDR_CH2_H	512	0x0680_0000_0000	0x06FF_FFFF_FFFF	DDR channel 2 high
DDR_CH3_L	512	0x0700_0000_0000	0x077F_FFFF_FFFF	DDR channel 3 low
DDR_CH3_H	512	0x0780_0000_0000	0x07FF_FFFF_FFFF	DDR channel 3 high
NOC_PL_H	8192	0x0800_0000_0000	0x0FFF_FFFF_FFFF	NoC ports to PL, high region

PMC and PS Address Map

The 4 gigabyte (GB) address space for the PMC and PS is summarized in the following table.

Table 33: **PMC and PS Address Map (4 GB)**

Interface	Size (MB)	Address Range		Notes
		Start	End	
DDR memory	2048	0x0000_0000	0x7FFF_FFFF	DDR Channel 0 region 0
M_AXI_LPD	512	0x8000_0000	0x9FFF_FFFF	LPD to PL AXI interface, low address
reserved	64	0xA000_0000	0xA3FF_FFFF	reserved
M_AXI_FPD	192	0xA400_0000	0xAFFF_FFFF	FPD to PL AXI interface, low address 0
	256	0xB000_0000	0xBFFF_FFFF	FPD to PL AXI interface, low address 1
reserved	512	0xC000_0000	0xDFFF_FFFF	reserved
PCIe0	256	0xE000_0000	0xEFFF_FFFF	PCIe revision 0
PMC	128	0xF000_0000	0xF7FF_FFFF	Includes 32 MB address space for NPI
Coresight_STM	16	0xF800_0000	0xF8FF_FFFF	System trace macro (STM)
APU GIC	1	0xF900_0000	0xF90F_FFFF	APU generic interrupt controller
reserved	47	0xF910_0000	0xFBF_FFFF	reserved
CPM	16	0xFC00_0000	0xFCFF_FFFF	Coherent PCIe Module (when present)
FPD peripherals	16	0xFD00_0000	0xFDFF_FFFF	FPD Address Map
LPD peripherals	31	0xFE00_0000	0xFFEF_FFFF	LPD Address Map
	1	0xFFF0_0000	0xFFFF_FFFF	OCM

FPD Address Map

The FPD address map is summarized in the following table. The FPD address map range is 0xFD00_0000 to 0xFDFF_FFFF.

Table 34: **FPD Address Map**

Interface	Base Address	Size (KB)	Security	Notes
FPD_CCI programming	0xFD00_0000	1024		CCI-500 data
reserved	0xFD10_0000	640	–	reserved
FPD_CLK_RST	0xFD1A_0000	1280		FPD clock and reset (CRF registers)
reserved	0xFD2E_0000	512	–	reserved
S_AXI_HP	0xFD36_0000	64		PL to TBU 5 to FPD main switch or NoC
FPD_AMBA_NIU	0xFD37_0000	64		Interconnect control and status
S_AXI_HPC	0xFD38_0000	64		PL to TBU2 to CCI
FPD_XMPU	0xFD39_0000	64		Programming interface to FPD blocks
reserved	0xFD3A_0000	1216	–	reserved
FPD_SWDT	0xFD4D_0000	64		System watchdog timer
reserved	0xFD4E_0000	896	–	reserved
FPD_APU	0xFD5C_0000	64		APU programming interface
reserved	0xFD5D_0000	64	–	rserved
FPD_CCI	0xFD5E_0000	64		CCI-500 programming interface
SMMU_CSR	0xFD5F_0000	64		SMMU-500 in the FPD
reserved	0xFD60_0000	64	–	reserved
FPD_SLCR	0xFD61_0000	64		FPD system-level control and status
reserved	0xFD62_0000	448	–	reserved
FPD_SLCR_SECURE	0xFD69_0000	64	Self-secured	FPD system-level control and status
reserved	0xFD6A_0000	384	–	reserved
FPD_SMMU_SECURE	0xFD80_0000	8192	Always secure	SMMU-500 core, requires a secure transaction[1]
FPD_SMMU_NONSECURE			Always non-secure	SMMU-500 core, requires a non-secure transaction[1]

Notes:

1. The SMMU-500 core registers are banked by security. This means that a non-secure access will access the FPD_SMMU_NONSECURE register set, and a secure access will access the FPD_SMMU_SECURE register set.

LPD Address Map

The LPD address map is shown in the following table. The LPD address map range is `0xFE00_0000` to `0xFFFF_FFFF`.

This is a global view of the LPD address map. The address maps from the perspective of the RPU are described in CPU Local Memory Maps.

Table 35: **LPD Address Map**

Interface	Base Address	Size (KB)	Security	Notes
LPD_IOP_GPV	0xFE00_0000	1024		LPD IOP interconnect
reserved	0xFE10_0000	1024	–	reserved
USB_2_XHCI	0xFE20_0000	1024		USB 2.0 controller xHCI
reserved	0xFE30_0000	1024	–	reserved
LPD_GPV	0xFE40_0000	128	LPD_XPPU	LPD main interconnect
reserved	0xFE42_0000	1856	–	reserved
HSDP_DMA	0xFE5F_0000	64	LPD_XPPU	High-speed debug data port
LPD NIU	0xFE60_0000	2048	–	Interconnect control and status
XRAM memory	0xFE80_0000	8192	LPD_XPPU	XRAM memory (accelerator RAM option)
LPD I/O peripherals	0xFF00_0000	2048	LPD_XPPU	LPD IOP Address Map
reserved	0xFF20_0000	704	–	reserved
IOP	0xFF2B_0000	320	LPD_XPPU	LPD I/O peripherals
IPI	0xFF30_0000	1024	LPD_XPPU	Inter-processor interrupt controller
reserved	0xFF40_0000	64	–	reserved
LPD_SLCR	0xFF41_0000	1024	LPD_XPPU	LPD system-level control
LPD_SLCR_SECURE	0xFF51_0000	256	LPD_XPPU	Secure LPD system-level control
reserved	0xFF55_0000	576	–	reserved
CRL	0xFF5E_0000	3072	LPD_XPPU	LPD clocks and resets
XRAM_CSR	0xFF8E_0000	512	LPD_XPPU	XRAM memory control and status
OCM	0xFF96_0000	64	LPD_XPPU	On-chip memory control
XRAM_PMC_XMPU	0xFF97_0000	64	LPD_XPPU	XRAM protection unit on the PMC port
OCM_XMPU	0xFF98_0000	64	LPD_XPPU	OCM XMPU protection unit
LPD_XPPU	0xFF99_0000	64	LPD_XPPU	LPD XPPU protection unit
RPU	0xFF9A_0000	64	LPD_XPPU	RPU control
M_AXI_LPD	0xFF9B_0000	64	LPD_XPPU	LPD to PL AXI interface control
AURORA	0xFF9C_0000	64	LPD_XPPU	Aurora debug interface
USB_2_CSR	0xFF9D_0000	64	FPD_XPPU	USB 2.0 control and status

Table 35: **LPD Address Map** *(cont'd)*

Interface	Base Address	Size (KB)	Security	Notes
reserved	0xFF9E_0000	128	–	reserved
reserved	0xFFA0_0000	512	–	reserved
LPD_DMA	0xFFA8_0000	512	LPD_XPPU	LPD DMA control
reserved	0xFFB0_0000	1024	–	reserved
PSM	0xFFC0_0000	1024	LPD_XPPU	PSM controller
reserved	0xFFD0_0000	1024	–	reserved
RPU cache TCM	0xFFE0_0000	1024	LPD_XPPU	Control for RPU caches and tightly-coupled memory
reserved	0xFFF0_0000	768	–	reserved
OCM	0xFFFC_0000	256	LPD_XPPU	256 KB of on-chip memory

LPD IOP Address Map

The address map for the LPD I/O peripherals is shown in the following table. The LPD IOP address map range is 0xFF00_0000 to 0xFF1F_FFFF.

Table 36: **LPD I/O Peripherals Address Map**

Interface	Base Address	Size (KB)	Secure Access	Notes
UART0	0xFF00_0000	64		
UART1	0xFF01_0000	64		
LPD_I2C0	0xFF02_0000	64		
LPD_I2C1	0xFF03_0000	64		
SPI0	0xFF04_0000	64		
SPI1	0xFF05_0000	64		
CANFD0	0xFF06_0000	64		
CANFD1	0xFF07_0000	64		
LPD_IOP_SLCR	0xFF08_0000	128		
LPD_IOP_SECURE_SLCR	0xFF0A_0000	64		
LPD_GPIO	0xFF0B_0000	64		PS LPD GPIO controller
GEM0	0xFF0C_0000	64		
GEM1	0xFF0D_0000	64		
TTC0	0xFF0E_0000	64		
TTC1	0xFF0F_0000	64		
TTC2	0xFF10_0000	64		
TTC3	0xFF11_0000	64		
LPD_SWDT	0xFF12_0000	64		System watchdog timer

Table 36: **LPD I/O Peripherals Address Map** *(cont'd)*

Interface	Base Address	Size (KB)	Secure Access	Notes
LPD_SCNTR	0xFF13_0000	64		
LPD_SCNTRS	0xFF14_0000	64	Yes	
reserved	0xFF15_0000	704		
USB 2.0				See LPD Address Map

PMC Address Map

The PMC address map is shown in the following table. The PMC address map range is 0xF000_0000 to 0xF7FF_FFFF. The programming interfaces are attached to a 32-bit APB port except where noted to be a 32-bit AXI interface port.

Table 37: **PMC Address Map**

Interface	Base Address	Size (KB)	Security	Notes
reserved	0xF000_0000	128	–	reserved
PPU_IOMODULE	0xF028_0000	4		Control and status
reserved	0xF028_1000	13824	–	reserved
PMC_I2C	0xF100_0000	64		Control and status
OSPI	0xF101_0000	64		Control and status
PMC_GPIO	0xF102_0000	64		Control and status
QSPI	0xF103_0000	64		Control and status
SD0	0xF104_0000	64		AXI programming interface
SD1	0xF105_0000	64		AXI programming interface
PMC_IOP_SLCR	0xF106_0000	64		IOP programming interface
PMC_IOP_SECURE_SLCR	0xF107_0000	64		
reserved	0xF108_0000	512	–	reserved
PMC_RAM	0xF110_0000	64		Control and status
PMC_GLOBAL	0xF111_0000	320		Control and status
PMC_ANALOG	0xF116_0000	256		Control and status
PMC_TAP	0xF11A_0000	128		Control and status
PMC_DMA0	0xF11C_0000	64		Control and status
PMC_DMA1	0xF11D_0000	64		Control and status
AES	0xF11E_0000	64		Control and status
BBRAM	0xF11F_0000	64		Control and status
ECDSA_RSA	0xF120_0000	64		Control and status
SHA3	0xF121_0000	64		Control and status
SBI	0xF122_0000	64		Slave boot interface

Table 37: **PMC Address Map** *(cont'd)*

Interface	Base Address	Size (KB)	Security	Notes
TRNG	0xF123_0000	64		True random number generator
reserved	0xF124_0000	64	–	reserved
EFUSE_CACHE	0xF125_0000	64		Control
CRP	0xF126_0000	64		PMC clock and reset registers
PMC_SYSMON	0xF127_0000	192		Control and status
RTC	0xF12A_0000	64		Control and status
CFU_CSR	0xF12B_0000	64		Control and status
reserved	0xF12C_0000	192	–	
PMC_XMPU	0xF12F_0000	64		Control and status
PMC_XPPU_NPI	0xF130_0000	64		Control and status
PMC_XPPU	0xF131_0000	64		Control and status
reserved	0xF132_0000	64	–	reserved
PMC_INT_CSR	0xF133_0000	2560		Control and status
reserved	0xF15B_0000	10560	–	reserved
PMC_RAM	0xF200_0000	128		128 KB memory space
reserved	0xF202_0000	896	–	reserved
SBI_STREAM	0xF210_0000	64		Streaming port
reserved	0xF211_0000	64448	–	reserved
NPI master	0xF600_0000	32768		Entire NPI programming interface address space

Programming Interfaces

There are several types of programming interfaces:

- APB, AXI Programming Interface with single 32-bit read/write access
- NPI Programming Interface with burst 32-bit read/write access
- PL Configuration Frame Programming Interface in the configuration frame unit (CFU) with its 128-bit read/write with single port and memory-mapped block access

Programming Interfaces for Memory-mapped Registers

The APB, AXI registers are used by the PMC, PS, and CPM subsystems. The NPI interface is used for the NoC interconnect, DDR memory controller, and other peripherals attached to the NoC interconnect. Many of the NPI registers are configured by the Vivado® tools and are not user accessible.

The programming interfaces are protected by the Xilinx protection units:

- PMC APB programming interface (PMC_XPPU)
- NPI programming interface (PMC_XPPU_NPI)
- LPD APB, AXI programming interface (LPD_XPPU)
- FPD AXI programming interface (FPD_XMPU)

Configuration Frame Interface

The configuration frame interface (CFI) receives programming instructions from the configuration frame unit (CFU) to program several integrated blocks and I/Os. The CFU receives programming instructions from the PLM processing the .cfi files.

APB, AXI Programming Interface

The PMC, PS, and CPM subsystems include an interconnect switch to enable software to address the APB4 programming interfaces to access control and status registers. The APB interface also includes the programming interface error signal output, PSLVERR, to the APB interconnect, and a system interrupt output, IRQ.

The memory-mapped registers are accessed using 32-bit words as described in Programming Interfaces.

The programming registers for the blocks in the PMC, PS, and CPM are accessed using 32-bit R/W transactions to APB programming interface ports on each unit. The AXI interconnect of each subsystem has an APB switch that connects to the APB programming interface of the functional units. Interrupts are signaled directly to the system interrupt controllers in the FPD, LPD, PMC, and to the PL fabric.

Address Maps

The APB, AXI interfaces are divided into several address maps for the FPD, LPD, and PMC as described in these sections:

- FPD Address Map
- LPD Address Map
- PMC Address Map

Accessibility

The accessibility of the APB programming interfaces depends on the configuration of the XPPUs that are protecting them.

Memory-mapped Register Access Types

The memory-mapped (MM) register can be read, write, or have another access type as shown in the following table.

Table 38: **Memory-mapped Register Access Types**

Access Type	Description
R	Read-only
W	Write-only
RW	Read or write
WTC or W1C	Write 1 to clear (readable unless noted)

NPI Programming Interface

The NoC structure includes a register programming interface (NPI) to configure the NoC components and many of the attached programming interfaces. The NPI also transports interrupts from the subsystem unit back to the NPI controller where they are signaled as system interrupts.

The single NPI controller is accessible a programming interface on the PMC AXI main switch. The NPI master unit includes its own XPPU protection unit. The NPI bus structure is in the SoC power domain, same as the NoC, but it functions completely independently of the NoC interconnect. The NPI supports burst accesses to reduce register programming times.

Features

The NPI features include:

- Read/write pathway to the programming control and status registers (PCSRs)
- Burst read/write transactions
- Ordered reads and writes
- APB error interrupt (programming interface)

Slave Errors and Interrupts

If a subsystem unit generates an access error, or system interrupt, it is signaled back to the NPI master. The interrupts are routed to the system interrupt controllers, including the RPU and APU GICs, the PMC system interrupt controller, and to PL outputs in the fabric. The errors are routed to the system error controller in the PMC and to PL outputs in the fabric.

Accessibility

The NPI controller is attached to the AXI main switch in the PMC. Accesses are monitored by the PMC_XPPU_NPI protection unit.

Configuration Frame Programming Interface

The PL adaptable engines, fabric, clocks, and integrated hardware is programmed using configuration frames (CFRAME). The configuration frames are written to the configuration frame unit (CFU). The CFU receives data files and generates configuration packets out on to the configuration frame interface (CFI) to program the device.

The PL building blocks are introduced in Programmable Logic with references to related documents.

Section V

Signals, Interfaces, and Pins

This section includes these chapters:

PMC Dedicated Pins

There are two sets of PMC dedicated pins:

- 15 dedicated digital pins, DIO
- 6 dedicated analog pins, DIO_A

The DIO pins are on the bank_503 package bank. These pins provide functions such as boot mode selection, external reference clock input, power-on reset input, JTAG interface, status signals, error signals, and crystal oscillator pins for the real-time clock (RTC).

The DIO_A pins are on the bank_500 bank along with the digital PMC MIO bank 0 pins. The analog pins are for the SYSMON voltage reference and the anode/cathode connections to the on-chip thermal diode. The analog reference pins use the VCCO and ground for ESD protection only.

DIO Pins on Bank 503

The following table lists these dedicated digital I/O pins.

Table 39: **PMC Dedicated Digital Pins, DIO**

Pin Name	Direction	Description
DONE	Output	The DONE pin is an output-only, open-drain signal with a weak internal pull-up. An external pull-up is recommended. DONE is controlled by the PMC_GLOBAL.DONE register. After POR, the DONE signal is Low. When the PLM successfully completes the boot sequence, the software sets the [DONE] bit High, which causes the output buffer to float and be pulled High externally.
ERROR_OUT	Output	The ERROR_OUT pin is an output-only, open-drain signal with a weak internal pull-up. An external pull-up is recommended. ERROR_OUT is in a High-Z state and pulled High when an error occurs in the device. The specific errors that cause this pin to assert can be determined and programmed by software.
MODE[3:0]	Input	The MODE[3:0] pins are used to select the boot mode for the device. The value of these pins is captured on the rising edge of POR_b. See Boot Modes for available boot mode details.
POR_b	Input	The active-Low POR_b pin is the global power-on reset for the Versal ACAP. POR_b must remain asserted Low until power is fully applied to at least the VCC_PMC, VCCAUX_PMC, and VCCO_503. When the reset is released, the PMC begins the initialization and boot process.
PUDC_b	Input	The active-Low PUDC_b (pull-up during configuration) pin is used to select the behavior of the programmable logic (PL) I/O during configuration. If the PUDC_b pin is High, the PL I/O are put into 3-state mode. If the PUDC_b is Low, internal pull-ups at each programmable logic I/O are enabled. The PUDC_b pin does not affect the PS or PMC I/O during boot and configuration.

Table 39: **PMC Dedicated Digital Pins, DIO** *(cont'd)*

Pin Name	Direction	Description
REF_CLK	Input	System reference clock pin. The system reference clock is required for all boot modes.
RTC_PADI	Input	RTC crystal input pin.
RTC_PADO	Output	RTC crystal output pin.
TCK	Input	JTAG test clock pin.
TDI	Input	JTAG test data input pin.
TDO	Output	JTAG test data output pin.
TMS	Input	JTAG test mode select pin.

DIO_A on Bank 500

There are two groups of DIO_A pins:

- Analog SYSMON pins with ESD protection
- Thermal diode pins, isolated from bank power

***Note*:** The DIO_A pins and the PMC MIO bank 0 pins share the same package bank, VCCO_500. However, the analog pins only use the VCCO_500 bank for ESD protection.

Table 40: **Dedicated Analog Pins, DIO_A**

Pin Name	Power Connection	Description
SYSMON_VREFN	Ground voltage reference	ADC reference voltage, negative (optional)
SYSMON_VREFP	Positive voltage reference	ADC reference voltage, positive
SYSMON_VN	Differential voltage	SYSMON analog input, negative
SYSMON_VP		SYSMON analog input, positive
DXN	Forward voltage measurement	Thermal diode, cathode
DXP		Thermal diode, anode

Multiplexed I/O

The peripheral I/Os are normally routed to the multiplexed I/O (MIO) pins. There are three MIO banks that connect I/O peripherals in the PMC and LPD to device pins. If an I/O interface is not routed via an MIO, its availability on the PL extended MIO (EMIO) interface is described in MIO-EMIO Interface Options.

I/O Pinout Considerations

***Note*:** There are several important MIO pin assignment considerations. The MIO-at-a-Glance table and the I/O pin assignment considerations are helpful for pin planning. Each chapter includes individual MIO signal tables for each controller/unit that uses the MIO pins. The MIO-at-a-Glance table includes links to these individual MIO signal tables.

I/O Interface Grouping

I/O interfaces include bus protocol signals with timing specifications and signals without timing requirements. The signals with timing requirements must be routed to the device pins as a group. The MIO pin groupings are shown in the MIO-at-a-Glance Tables. The non-timing related signals can be split up and routed individually through an MIO or EMIO.

The pin groupings are shown in columns in the individual MIO signal tables. Select one column of pin assignments for the timing-sensitive signals, and do not mix and match column entries.

For I/O signals without a timing specification (e.g., write protect, card detect, etc.), their own individual pinout routing can be used.

Interface Frequencies

The clocking frequency for an interface usually depends on the device speed grade and whether the interface is routed through the MIO or EMIO. Nominal interface frequencies are usually included in the associated chapter with some restrictions for EMIO routing shown in the MIO-EMIO Interface Options table. The I/O timing specifications are provided in the Versal ACAP data sheets.

I/O Buffer Output Enable

The output enable for each MIO I/O buffer is normally controlled by the peripheral controller. There are tristate override registers that can be used to tristate an entire bank of signals, if needed.

- PMC_IOP_SLCR.MIO_MST_TRI0 (PMC MIO Bank 0)
- PMC_IOP_SLCR.MIO_MST_TRI1 (PMC MIO Bank 1)
- LPD_IOP_SLCR.MIO_MST_TRI (LPD MIO Bank)

When the tristate override control bit equals 1, the output on the I/O buffer is disabled and the pin will float according to the weak pull-up or pull-down settings.

Internal Pull-up, Pull-down Resistors

Each I/O buffer in the MIOs has a pull-up and pull-down option. The PMC MIO bank is controlled by the PMC_IOP_SLCR.BANK_WK_PU and BANK_WK_PD registers. If both the pull-up and pull-down bits are set = 1, the I/O buffer will weakly hold the output in its last driven state.

Boot Device Options

The boot device options are listed in the Boot Modes chapter.

MIO-at-a-Glance Tables

MIO Device Pins

There are 78 sets of signals to control the MIO pins.

- 52 signals in the PMC MIO (banks 500 and 501)
- 26 signals in the LPD MIO (bank 502)

Signal Route Control

The PMC MIO device pins include signals associated with functionality in the PMC and LPD subsystem. The LPD MIO device pins only include signals associated with the LPD subsystem.

Many of the IOP controller and other signals are routed to the EMIO by default if they are not specifically routed to MIO pins using the following registers.

- PMC_IOP_SLCR.MIO_PIN_n
- LPD_IOP_SLCR.MIO_PIN_n

The MIO interfaces for the LPD-based controllers are routed to either the LPD or PMC MIO banks. The selection is done using the LPD_IOP_SLCR.LPD_MIO_SEL register.

PL EMIO Signal Route

Some interfaces and signals also go to the PL, and for most interfaces, these are listed in MIO-EMIO Interface Options.

MIO Pin Assignments By Banks

The MIO pin assignments are shown in the following tables with links to the chapter sections that list the I/O interface signals.

***Note*:** The pins that can connect to a primary boot device are shaded in the following tables. See Boot Modes for exact pin usages.

- Bank 500 includes eMMC1 boot interface
- Bank 501 includes SD0 and SD1 boot interfaces

Table 41: **PMC MIO (Bank 500)**

PMC MIO Pins:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
PMC Power Domain																										
Quad SPI	0	1	2	3	4	5	6	7	8	9	10	11	12													
Octal SPI	0	1	2	3	4	5	6	7	8	9	10	11	12													
SD_eMMC_0														4	5	6	7	12	2	8	9	10	11	3	1	0
SD_eMMC_1	2	0	1	3	4	5	6	7	8	9	10	11	12													
SelectMAP															0	1	2	3	32	33	34	35	4	5	6	7
TamperTrig													0	0									0	0		
PMC_I2C			0	1			0	1			0	1			0	1			0	1			0	1		
PMC_GPIO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Low Power Domain																										
LPD_GPIO																										
GEM0																										
GEM1																										
GEM MDIO																										
TSU Clock																										
CAN0	0	1			0	1			0	1			0	1			0	1			0	1				
CAN1			1	0			1	0			1	0			1	0			1	0			1	0		
LPD_I2C0			0	1			0	1			0	1			0	1			0	1			0	1		
LPD_I2C1	0	1			0	1			0	1			0	1			0	1			0	1				
SYSMON_I2C	0	1	2		0	1	2			0	1	2		0	1	2			0	1	2			0	1	2
PCIe resets																									0	1
SPI0	5	4	3	2	1	0							5	4	3	2	1	0								
SPI1							5	4	3	2	1	0							5	4	3	2	1	0		
Trace					1	2	0	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17				
TTC0							0	1							0	1							0	1		
TTC1					0	1							0	1							0	1				
TTC2			0	1							0	1							0	1						
TTC3	0	1							0	1							0	1								
UART0	0	1	2	3					0	1	2	3					0	1	2	3						
UART1					1	0	3	2					1	0	3	2					1	0	3	2		
USB 2.0														12	4	5	6	7	0	8	9	10	11	1	2	3
SWDT0	0	1	2	3	4	5							0	1	2	3	4	5								
SWDT1							0	1	2	3	4	5							0	1	2	3	4	5		

Table 42: **PMC MIO (Bank 501)**

PMC MIO Pins:	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
PMC Power Domain																										
Quad SPI																										
Octal SPI																										
SD_eMMC_0												0	2	1	3	4	5	6	7	8	9	10	11	12		
SD_eMMC_1	2	11	1	3	4	5	6	7	8	9	10														0	12
SelectMAP			8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
TamperTrig	0											0													0	0
PMC_I2C	0	1			0	1			0	1			0	1			0	1			0	1			0	1
PMC_GPIO	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
Low Power Domain																										
LPD_GPIO																										
I/O Signal Reference	0	1	2	3	4	5	6	7	8	9	10	11														
I/O Signal Reference													0	1	2	3	4	5	6	7	8	9	10	11		
GEM MDIO																									0	1
TSU Clock																									0	0
CAN0	0	1			0	1			0	1			0	1			0	1			0	1				
CAN1			1	0			1	0			1	0			1	0			1	0			1	0		
LPD_I2C0	0	1			0	1			0	1			0	1			0	1			0	1				
LPD_I2C1			0	1			0	1			0	1			0	1			0	1			0	1		
SYSMON_I2C	0	1	2		0	1	2			0	1	2		0	1	2			0	1	2			0	1	2
PCIe resets													0	1												
SPI0	5	4	3	2	1	0							5	4	3	2	1	0								
SPI1							5	4	3	2	1	0							5	4	3	2	1	0		
Trace					1	2	0	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17				
TTC0							0	1							0	1							0	1		
TTC1					0	1							0	1							0	1				
TTC2			0	1							0	1							0	1						
TTC3	0	1							0	1							0	1								
UART0	0	1	2	3					0	1	2	3					0	1	2	3						
UART1					1	0	3	2					1	0	3	2					1	0	3	2		
USB 2.0																										
SWDT I/O Signals	0	1	2	3	4	5							0	1	2	3	4	5								
SWDT I/O Signals							0	1	2	3	4	5							0	1	2	3	4	5		

Table 43: **LPD MIO (Bank 502)**

LPD MIO Pins:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
PMC Power Domain																										
Quad SPI																										
Octal SPI																										
SD_eMMC_0																										
SD_eMMC_1																										
SelectMAP																										
TamperTrig																										
PMC_I2C																										
PMC_GPIO																										
Low Power Domain																										
LPD_GPIO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
I/O Signal Reference	0	1	2	3	4	5	6	7	8	9	10	11														
I/O Signal Reference													0	1	2	3	4	5	6	7	8	9	10	11		
GEM MDIO																									0	1
TSU Clock																									0	0
CAN0			0	1			0	1			0	1			0	1			0	1			0	1		
CAN1	1	0			1	0			1	0			1	0			1	0			1	0			1	0
LPD_I2C0			0	1			0	1			0	1			0	1			0	1			0	1		
LPD_I2C1	0	1			0	1			0	1			0	1			0	1			0	1			0	1
SYSMON_I2C	0	1	2		0	1	2			0	1	2		0	1	2			0	1	2			0	1	2
PCIe resets																			0	1						
SPI0	5	4	3	2	1	0							5	4	3	2	1	0								
SPI1							5	4	3	2	1	0							5	4	3	2	1	0		
Trace			1	2	3	4	0	5	6	7	8	9	10	11					12	13	14	15	16	17		
TTC0							0	1							0	1							0	1		
TTC1					0	1							0	1							0	1				
TTC2			0	1							0	1							0	1						
TTC3	0	1							0	1							0	1								
UART0	0	1	2	3					0	1	2	3					0	1	2	3						
UART1					1	0	3	2					1	0	3	2					1	0	3	2		
USB 2.0																										
SWDT0	0	1	2	3	4	5							0	1	2	3	4	5								
SWDT1							0	1	2	3	4	5							0	1	2	3	4	5		

Special MIO Clock Routing

There are other MIO multiplexing mechanisms that can work together with MIO_PIN_xx registers and can also be independent.

- CAN reference clock input on MIO (see the LPD_IOP_SLCR.CAN_MIO_CTRL register)
- SWDT reference clock select (MIO or APB interface)
- PSM wake-up input on MIO[0:5]

MIO-EMIO Interface Options

The I/O interfaces for the IOP controllers and other units are routed to the PMC and LPD MIO multiplexers. Some signals can be routed to the EMIO interface to the PL. Some IOP interfaces and signals are only available on the MIO (e.g., quad SPI). Other I/O signals are only available on the EMIO interface (e.g., LPD DMA handshake control).

The routing is configured by the MIO_PIN_x registers in the PMC_IOP_SLCR and LPD_IOP_SLCR register sets. The interfaces and signals that are routed through the MIO-EMIO are listed in the following table with their I/O interface routing options.

Table 44: **MIO-EMIO Interface Routing Options**

Interface or Signal	Controller Location	Access			Notes
		PMC MIO	LPD MIO	EMIO	
CAN_FD0 CAN_FD1	LPD	Yes	Yes	Yes	
GEM0 GEM1	LPD	Yes	Yes	-	RGMII
		-	-	Yes	GMII/MII, TSU, and external FIFO interfaces
		Yes	Yes	Yes	MDIO
LPD DMA	LPD	-	-	Yes	Flow control
PMC_GPIO	PMC	Yes	-	-	Banks 0, 1 (no bank 2)
		-	-	Yes	Banks 3, 4
LPD_GPIO	LPD	-	Yes	-	Bank 0 (no banks 1, 2)
		-	-	Yes	Bank 3
LPD_I2C0 LPD_I2C1	LPD	Yes	Yes	Yes	
PMC_I2C	PMC	Yes	-	Yes	
Octal SPI	PMC	Yes		-	
SD/eMMC0 SD/eMMC1	PMC	Yes	-	Yes	The clock frequency for the EMIO interface is <= 25 MHz
SelectMAP	PMC	Yes	-	-	
SPI0 SPI1	LPD	Yes	Yes	Yes	
Quad SPI	PMC	Yes	-	-	

Table 44: **MIO-EMIO Interface Routing Options** *(cont'd)*

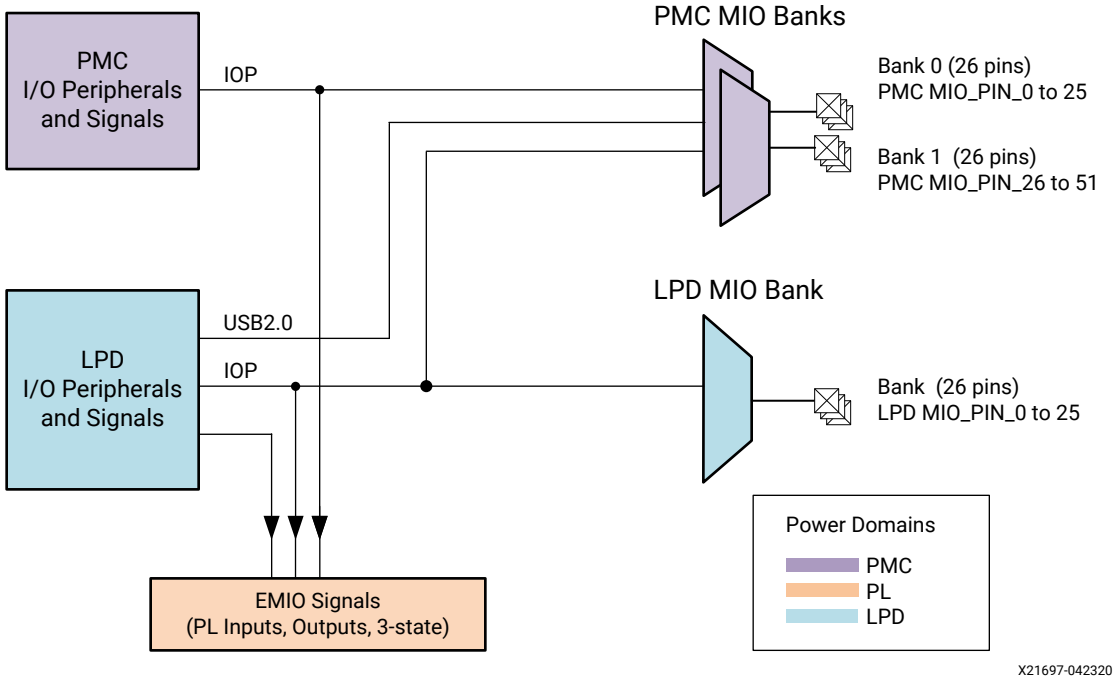
Interface or Signal	Controller Location	Access			Notes
		PMC MIO	LPD MIO	EMIO	
CoreSight™ Trace Out	FPD	16-bit	16-bit	32-bit	
TTC0 TTC1 TTC2 TTC3	LPD	Yes	Yes	Yes	Clock in and wave out
UART0 UART1	LPD	Yes	Yes	Yes	MIO only includes RX, TX, CTS, and RTS
USB_2.0	LPD	Yes	-	-	ULPI
LPD_SWDT	LPD	Yes	Yes	Yes	
FPD_SWDT	FPD	~	~	~	

MIO-EMIO Wiring

The I/O interfaces for the IOP controllers and others are routed to the PMC and LPD MIO multiplexers. They can also be routed to the EMIO interface to the PL. Some IOP interfaces and signals are only available on the MIO multiplexers (e.g., quad SPI). Other I/O signals are only available on the EMIO interface (e.g., LPD DMA handshake control).

The routing of the I/O interface signals through the MIO and EMIO is shown in the following figure.

Figure 33: **MIO-EMIO Wiring**



MIO Pin Configuration

Each MIO pin includes programming registers for signal routing and I/O buffer configuration. The MIO_PIN_xx registers control the signal routing; one register per pin.

There are master tristate control registers that override the output enable signal from the peripheral.

- PMC_IOP_SLCR.MIO_MST_TRI0
- PMC_IOP_SLCR.MIO_MST_TRI1
- LPD_IOP_SLCR.MIO_MST_TRI

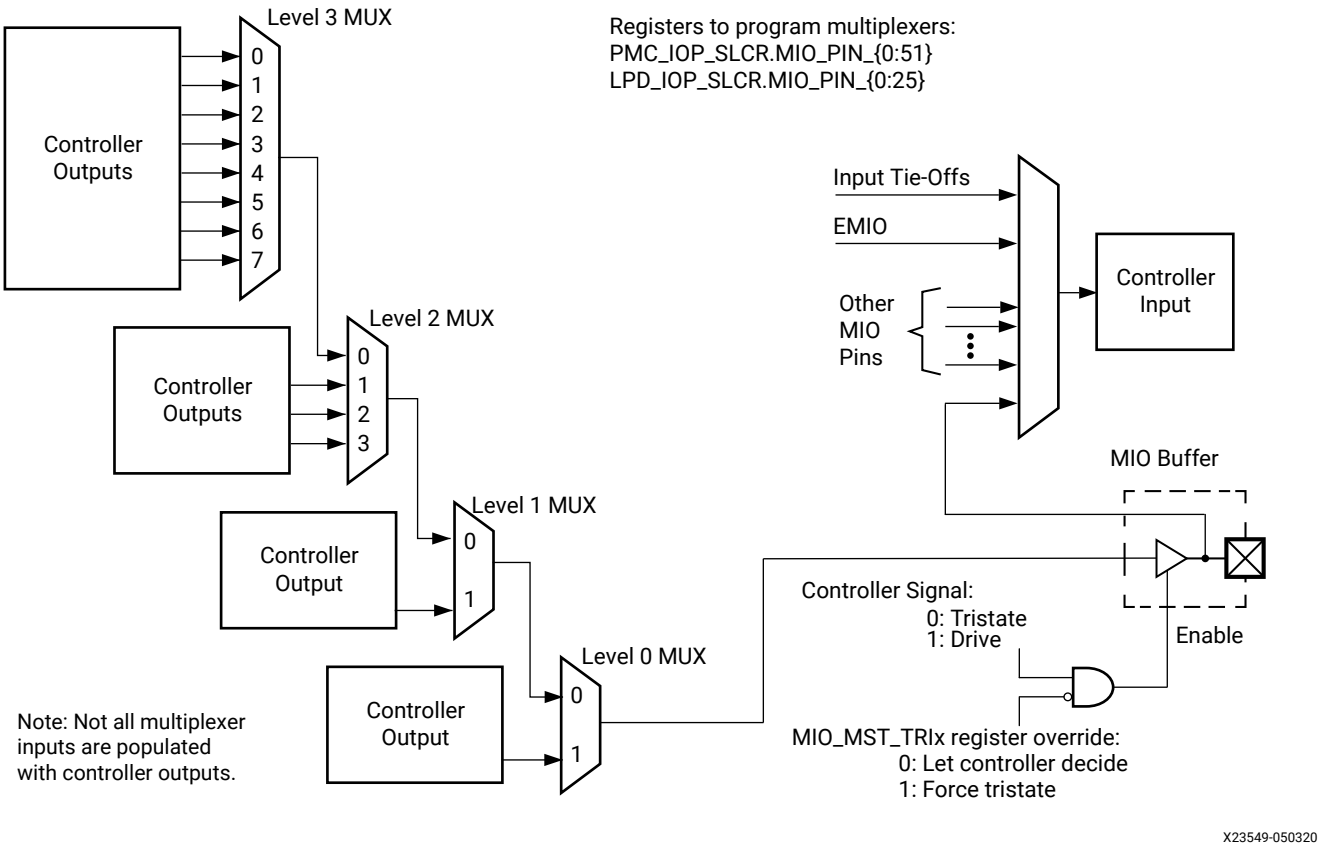
Signal Routing Controls

For LPD pin multiplex, the first routing decision for connection to an MIO pin is to select between the PMC and the LPD MIO multiplexers. The selection is done using the LPD_IOP_SLCR.LPD_MIO_SEL register. Not all, but most PMC controllers have a choice. The I/O for all PMC controllers is via the PMC MIO, so there is no decision to make; they always go to the PMC MIO.

Output Multiplexer

The output multiplexer has several cascading multiplexers as shown in the following figure. The level 3 multiplexing is used for low-speed signals. The level 0 multiplexing is used for high-speed signals and many of the clocks.

Figure 34: **MIO Multiplexing Levels**



MIO PSIO Buffer Configuration

The characteristics of each MIO buffer are individually controlled. There are two sets of registers for the two sets of MIO banks.

- PMC_IOP_SLCR for the PMC MIO banks 0 and 1
- LPD_IOP_SLCR for the LPD MIO bank (in the PS)

These banks have the following pins.

- PMC MIO pins 0 to 25
- PMC MIO pins 26 to 51
- LPD MIO pins 0 to 25

Input Controls

The input buffer controls for the MIO pin banks are shown in the following table.

***Note*:** If both bits associated with pins in BANK_WK_PU and BANK_WK_PD = 1, the I/O keeps its last value when put in tristate.

Table 45: **MIO Input Buffer Control Registers**

Feature	PMC_IOP_SLCR Registers		LPD_IOP_SLCR Registers	Description
	PMC MIO Bank 0	PMC MIO Bank 1	LPD MIO Bank	
	Pins 0 to 25	Pins 26 to 51	Pins 0 to 25	
Weak pull-up	BANK0_WK_PU [pins_0_25]	BANK1_WK_PU [pins_26_51]	BANK_WK_PU [pins_0_25]	0: Disable 1: Enable
Weak pull-down	BANK0_WK_PD [pins_0_25]	BANK1_WK_PD [pins_26_51]	BANK_WK_PD [pins_0_25]	
Schmitt trigger with hysteresis	BANK0_RX_SCHMITT [pins_0_25]	BANK1_RX_SCHMITT [pins_26_51]	BANK_RX_SCHMITT [pins_0_25]	

Output Controls

The output buffer controls for the MIO pin banks are shown in the following table.

Table 46: **MIO Output Buffer Control Registers**

Feature	PMC_IOP_SLCR Registers		LPD_IOP_SLCR Registers	Description
	PMC MIO Bank 0	PMC MIO Bank 1	LPD MIO Bank	
	Pins 0 to 25	Pins 26 to 51	Pins 0 to 25	
Drive strength	BANK0_SEL_DRV0 [pins_0_15] BANK0_SEL_DRV1 [pins_16_25]	BANK1_SEL_DRV0 [pins_26_41] BANK1_SEL_DRV1 [pins_42_51]	BANK_SEL_DRV0 [pins_0_15] BANK_SEL_DRV1 [pins_16_25]	drv1 | drv0: 00: 2 mA 01: 4 mA 10: 8 mA 11: 12 mA
Skew	BANK0_SEL_SKEW [pins_0_25]	BANK1_SEL_SKEW [pins_26_51]	BANK_SEL_SKEW [pins_0_25]	0: Slow-slew 1: Fast-slew
Master 3-state control	MIO_MST_TRI0 [PIN_xx_TRI]	MIO_MST_TRI1 [PIN_xx_TRI]	MIO_MST_TRI [PIN_xx_TRI]	0: Controller 1: Tristated

Pin Programming Examples

Route Signal Through MIO

Routing a signal through an MIO is sometimes a two-step process:

- Configure the MIO pin (required)
- Select between the PMC and LPD MIOs (not always required)

Configure I/O Buffer for Input

The I/O buffer attributes are listed in MIO PSIO Buffer Configuration.

The steps to configure the input for LPD MIO pin 18 include:

1. Select the MIO (PMC or LPD) using LPD_IOP_SLCR.LPD_MIO_SEL (0: PMC MIO).
2. Route the signal through the MIO using LPD_IOP_SLCR.MIO_PIN_18 register.
3. Disable the output driver. Write a 1 to the LPD_IOP_SLCR.MIO_MST_TRI [PIN_18_TRI] bit.
4. Select the CMOS input (not Schmitt). Write 0 to the BANK_RX_SCHMITT [18] bit.
5. Enable the weak internal pull-up resister. Write 1 to the BANK_WK_PU [18] bit.
6. Set the output drive to 2 mA. Write 0 to the BANK_SEL_DRV0 [18] and BANK_SEL_DRV1 [18] bits.
7. Select a slow slew rate. Write 0 to BANK_SEL_SLEW [18].

PCIe Reset on MIO

The following table includes the PCIe reset signals routed to the MIO pins. This is a software-defined input signal using a GPIO channel in the PMC GPIO controller.

Table 47: **PCIe Controller Reset Input Signals**

MIO					
Signal Name	**I/O**	**PMC MUX Pin options**		**LPD MUX Pin**	**MIO-at-a-Glance Table**
		A	**B**		
PCIe_RESET1_b	Input	24	38	18	0
PCIe_RESET2_b	Input	25	39	19	1

Power Pins

The following table lists the Versal™ ACAP power supply pins.

The power domain inter-dependencies are explained in the Power chapter. This chapter also includes how these power pin connect to the power domains as illustrated in the Power Diagram.

The power specifications are listed in the Versal ACAP data sheets.

Table 48: **Power Pins**

Pin Name	Description
VCC_BATT	Battery power supply pin
VCC_FUSE	Power supply for eFUSE programming
VCC_IO	Power supply pins for the XPIO banks
VCC_PMC	Power supply pins for PMC
VCC_PSFP	Power supply pins for the PS full-power domain
VCC_PSLP	Power supply pins for the PS low-power domain
VCC_RAM	Power supply pins for block RAM, UltraRAM, and clocking network
VCC_SOC	Power supply pins for NoC, NPI, and the DDR
VCCAUX	Power supply pins for the auxiliary circuits
VCCAUX_PMC	Auxiliary power supply pins for the PMC
VCCAUX_SMON	Analog power supply pin for the ADC and other analog circuits in the SYSMON
VCCINT	Power supply pins for the internal logic (programmable logic, AI Engine, and CPM)
VCCO_[bank number]	Power supply pins for the output drivers (per bank)
VCCO_500	PMC MIO bank 0 power supply
VCCO_501	PMC MIO bank 1 power supply
VCCO_502	LPD MIO bank power supply in PS
VCCO_503	PMC dedicated I/O bank power supply
GTY_AVCC	Analog power supply pins for the receiver and transmitter internal circuits
GTY_AVCCAUX	Auxiliary analog power supply pins for the transceivers
GTY_AVTT	Analog power supply pins for the transmit driver

Port Interface Signals

The port interface signals traverse these power domain boundaries.

- PS-PL boundary
- PMC-PL boundary

PS-PL Boundary

The PS-PL boundary has many interfaces and signals that cross it, including:

- AXI PS-PL interfaces
- EMIO signals from PMC and LPD MIO multiplexers for I/O peripherals, flash, and more
- System interrupt and error signals
- Clock and reset signals
- PMC and PS configuration signals

AXI Interfaces

The PS-PL AXI interfaces provide direct connections between the PL and the LPD and FPD interconnect. These connections are shown in PS Interconnect Diagram and listed in Logical Link Layer. There are also two AXI4 masters in the PL. One connects to the LPD interconnect and the other connects to the FPD Cache Coherent Interconnect (CCI).

PS-PL Signals

The PS-PL boundary includes the following signals:

- EMIO from LPD to PS power domains

PMC-PL Boundary

The PMC-PL boundary includes the following signals:

- System interrupts
- System errors
- Extended MIO (EMIO)

Engines

This section includes the following chapters. Many engines are described in other documents as shown in the overview.

- Overview of all engines in the device
- Real-time Processing Unit RPU in PS
- Application Processing Unit APU in PS
- LPD DMA in PS

Overview

The Versal™ ACAP functionality includes scalar, intelligent, and adaptable engines. These are groups of building blocks for the system platform.

The scalar engines include RPU and APU. The DMA engines are also included. The intelligent engines include the AI engine and the DSPs in the PL. The PL has the adaptable engines. This includes the combinational look-up tables, RAMs, and programmable fabric.

Scalar Engines

The scalar engines include:

- Real-time processing unit (RPU)
- Application processing unit (APU)

Application Processing Unit

The APU MPCore processor is integrated into the FPD subsystem. The implementation and functionality of the Cortex™-A72 processor is detailed in the Application Processing Unit chapter with additional information in the Arm documents.

The main features of the APU:

- Dual Arm Cortex-A72 cores
- VFPv4 floating point, NEON, Crypto extension
- 48 KB instruction, 32 KB data caches
- GIC-500 interrupt controller
- 1 MB L2 Cache Coherent Interconnect (CCI)

Real-time Processing Unit

The RPU MPCore processor is integrated into the LPD subsystem of the PS as shown in the LPD Interconnect. The implementation and functionality of the Cortex-R5F processor is detailed in the Real-time Processing Unit chapter with additional information in the Arm documents.

The main features of the RPU:

- Dual Arm® Cortex™-R5F cores
- Lock-step and dual processor modes
- Tightly coupled memories for predictive execution times

Intelligent Engines

The Intelligent Engines include the AI Engine and the DSP Engine.

AI Engine

The AI Engine is a two-dimensional array of AI Engine tiles that each contain a high-performance VLIW vector (SIMD) processor, integrated memory, as well as interconnects for streaming, configuration, and debug. At the bottom of these tiles are the AI Engine array interface tiles that provide the necessary logic to connect the AI Engine to other resources including the PL, PS, and the NoC.

The AI Engine is integrated into the Versal™ ACAP AI Core series. For more information, see *Xilinx AI Engine and Their Applications* (WP506) and for additional information, see *Versal ACAP AI Engine Architecture Manual* (AM009).

DSP Engine

The DSP Engine combines high speed with small size to provide high performance and system design flexibility. The DSP Engines are integrated into the PL.

Each engine includes a dedicated 27 × 24 bit multiplier and a 58-bit accumulator. The multiplier can be dynamically bypassed, and two 58-bit inputs can feed a single-instruction multiple-data (SIMD) arithmetic unit (dual 24-bit or quad 12-bit add/subtract/accumulate), or a logic unit that can generate any one of ten different logic functions on the two operands.

New functional modes are implemented in the DSP Engine, including:

- 18 x 18 + 58 two's complement MAC with back-to-back DSP Engines
- Single-precision floating-point (binary32) accumulation
- Three-element two's complement vector doc product with accumulate or post-add in INT8 mode

For more information, see *Versal ACAP DSP Engine Architecture Manual* (AM004).

Adaptable Engines

The adaptable engines include:

- Configurable logic block (CLB)
- Block RAM memory array
- UltraRAM memory array

Configurable Logic Block

The CLB is briefly described in Configurable Logic Block. For more information, see the *Versal ACAP Configurable Logic Block Architecture Manual* (AM005).

Block RAM

The block RAM is briefly described in Block RAM. For more information, see the *Versal ACAP Memory Resources Architecture Manual* (AM007).

UltraRAM

The UltraRAM is briefly described in UltraRAM. For more information, see the *Versal ACAP Memory Resources Architecture Manual* (AM007).

DMA Units

The general purpose DMA controllers include:

- LPD DMA controller
- PL-instantiated DMA controllers

The integrated blocks include DMA controllers:

- Flash memory interface controllers (OSPI, QSPI, and SD/eMMC)
- I/O peripheral controllers (GEM Ethernet and USB 2.0)

XILINX

Chapter 28

Real-time Processing Unit

The real-time processing unit (RPU) provides predicable software execution times using Arm® Cortex™-R5F processors for real-time applications. The RPU is located in the LPD of the PS. Each processor includes separate L1 instruction and data caches and tightly coupled memories (TCM) to narrow down the deterministic behavior for real-time data processing applications. System memory is cacheable, but the TCM memory space is non-cacheable.

The RPU is a dual MPCore that can be configured for dual-processor or lock-step mode. The dual-processor mode provides higher performance. The lock-step configuration provides a high level of reliability for functional safety.

The RPU can execute instructions and access data from its TCMs, the OCM memory, the main DDR memory, and other system memories. When addressing system memory, the transactions can be routed directly to the NoC for accessing DDR memory, or through the APU Cache Coherent Interconnect (CCI) in the FPD for hardware coherent transactions with the APU's L2 cache.

Arm Documentation

This chapter describes general processor features and the implementation included in the Versal™ device. See the online Arm Cortex-R5F processor documentation for details.

Features

The Cortex-R5F processors include the following features:

- Integer execution unit with the Arm v7-R instruction set
- Single and double precision FPU with VFPv3 instructions
- Arm v7-R architecture memory protection unit (MPU)
- Dynamic branch prediction with a global history buffer and a 4-entry return stack
- 32 KB instruction L1 cache with ECC protection
- 32 KB data L1 cache with ECC protection
- 128 KB of TCM memory with ECC protection for each processor (256 KB total)

- CoreSight™ debug embedded trace module (ETM)
- Low latency and non-maskable, fast interrupts
- 64-bit master interface for accessing memory and shared peripherals
- 64-bit slave interface for system masters to access the TCMs

Comparison to Previous Generation Xilinx Devices

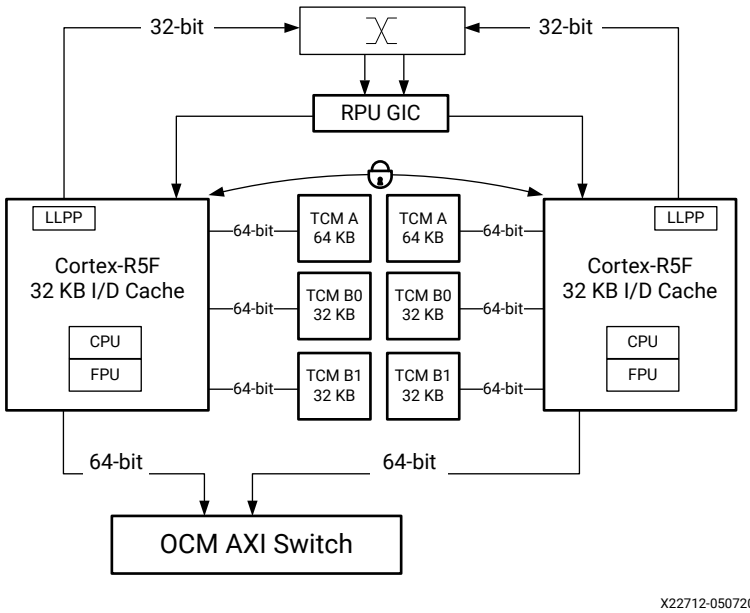
The RPU in the Versal ACAP is the same as the one in the Zynq® UltraScale+™device.

System Perspective

Block Diagram

The RPU block diagram is shown in the following figure.

Figure 35: **RPU Block Diagram**



AXI System Interfaces

There are several AXI interfaces:

- Two 64-bit AXI masters: connect to the OCM switch to access system slaves
- 32-bit AXI slave: provides access to the RPU GIC interrupt controller

- 64-bit AXI slave: provides access to the CPU memories (ICache, DCache, and TCMs)

Access to the caches by other system masters is only available during debug when the CPUs are put into their idle state. The AXI master interfaces enable the CPU memory system to have access to peripherals and system memories including OCM and DDR.

Implementation

The following table describes the Arm Cortex-R5F processor implementation. These are fixed in hardware.

***Note*:** For more information, see the *Arm Cortex-R5F Technical Reference Manual*.

Table 49: **RPU Implementation Settings**

Configuration Parameter	Value	Description
INITPPX	1	AXI peripheral interface enabled at reset
SLBTCMSB	0	B0 and B1 TCM interleaving by addr [3]
INITRAMA	0	Enable TCM_A
INITRAMB	1	Enable TCM_B
ENTCM1IF	1	Enable TCM_B1 interface
LOCZRAMA	1	TCM_A initial base address is zero
PPXBASE	Global	Base address of AXI peripheral interface
PPXSIZE	16 MBs	Size of AXI peripheral interface
PPVBASE	Global	Base address of virtual-AXI peripheral interface
PPVSIZE	8 KBs	Size of virtual-AXI peripheral interface

Hardware Configuration

The following table lists the RPU configuration controls.

Table 50: **RPU Configuration Controls**

Register	Bit	Reset Value	Description
RPU.RPU0_CONFIG	[VINITHI]	1	Reset V-bit value; when High indicates HIVECS mode at reset
	[CFGNMFI]	0	Maskable FIQ

Table 50: **RPU Configuration Controls** *(cont'd)*

Register	Bit	Reset Value	Description
RPU.GLOBAL_CNTL	[CFGEE]	0	Data is little endian
	[CFGIE]	0	Instructions are little endian
	[TEINIT]	0	At reset

Operating States

The two Cortex-R5F processors can operate independently or together in lock-step mode.

- Dual-processor configuration operates as two separate CPUs, which is also known as performance mode
- Lock-step configuration operates as a redundant CPU configuration, which is also known as safety mode

The processor supports static configuration. Switching between the lock-step and dual-processor configurations is only permitted immediately after a processor reset.

The RPU.GLOBAL_CNTL [SLCLAMP] and [SLSPLIT] register bits control the processor configuration.

Lock-Step Architecture

When the Cortex-R5F processors are configured to operate in the lock configuration, only one set of CPUs interface with the system, and memories are used.

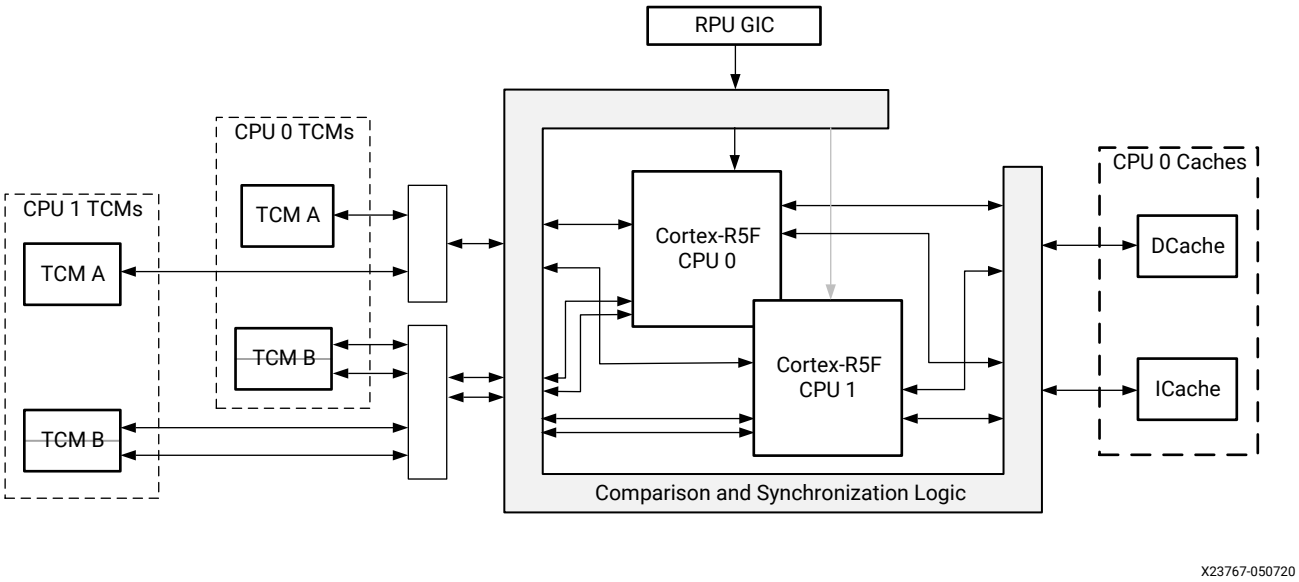
When the Cortex-R5F processors are in the lock-step mode (see the following figure), there should be code in the reset handler to ensure that the distributor within the generic interrupt controller (GIC) dispatches interrupts only to CPU0.

IMPORTANT! *During the lock-step operation, the TCMs that are associated with the redundant processor become available to the unified lock-step processors. The size of each TCM A and B becomes 128 KB with TCM B interleaved accesses from the processor and AXI slave interface.*

The RPU lock-step architecture includes comparison and synchronization logic to continually compare the data coming out of the CPUs. The reset handler software programs the interrupt distributor within the GIC to dispatch interrupts only to CPU0.

The RPU processor lock-step architecture is shown in the following figure.

Figure 36: **RPU Lock-Step Architecture**



Power Modes

The CPUs include three power management states as follows:

- Run
- Standby
- Shutdown

Each power level provides decreasing levels of power consumption, but increases the entry and exit requirements.

The following table lists the power management modes.

Table 51: **Power Management Modes**

Power Mode	CPU Clock	Power State	TCM Memory Retention	Exit to Run Mode Description
Run	Active	On	Yes	~
Standby	When idle	On	Yes	Pipeline restart
Shutdown	Inactive	Off	No	Pipeline restart restores registers and configuration from memory, invalidates caches, and reinitializes caches and TCMs

Address Maps

There are several memory maps related to the RPU. These include local and global perspectives.

- In this chapter:
  - CPU Local Memory Maps
  - Address Map from Global Perspective
  - Memory Map Diagram
- Content in the Address Maps and Programming Interfaces section:
  - LPD Address Map
  - Global Address Map

CPU Local Memory Maps

The local memory maps for the RPU CPUs include:

- Local TCM memory map
- Local interrupt control memory map

The CPUs have direct access to this memory space without a protection unit.

Local TCM Memory Map

The memory map from the CPU's point of view is shown in the following table.

Table 52: **Local TCM Memory Map**

Slave Type	Address Block Size (KB)	Local Map			Global Map
		Dual Processor Map		Lock-Step	
		RPU 0	RPU 1	RPU	
TCM_A	64	0x0000_0000	~	0x0000_0000	0xFFE0_0000
		~	0x0000_0000	0x0001_0000	0xFFE9_0000
TCM_B	64	0x0002_0000	~	0x0002_0000	0xFFE2_0000
		~	0x0002_0000	0x0003_0000	0xFFEB_0000
Instruction cache	32	I-Cache	~	~	0xFFE4_0000
		~	I-Cache	~	0xFFEC_0000
Data cache	32	D-Cache	~	~	0xFFE5_0000
		~	D-Cache	~	0xFFED_0000

Local Interrupt Registers

The address location of the RPU interrupt controller registers is shown in the following table.

Table 53: **Local Interrupt Control Address Map**

Slave	Address Block Size (KB)	Base Address	Register Set
CPU interrupt controller	4		
RPU 0 interrupt distributor	4		
RPU 1 interrupt distributor	4		

Address Map from Global Perspective

The RPU TCMs and L1 caches are mapped to the global memory space for potential access by any system master via the RPU slave interface port. This memory space is protected by the LPD_XPPU protection unit.

The global address map of the these RPU memories is shown in the following table.

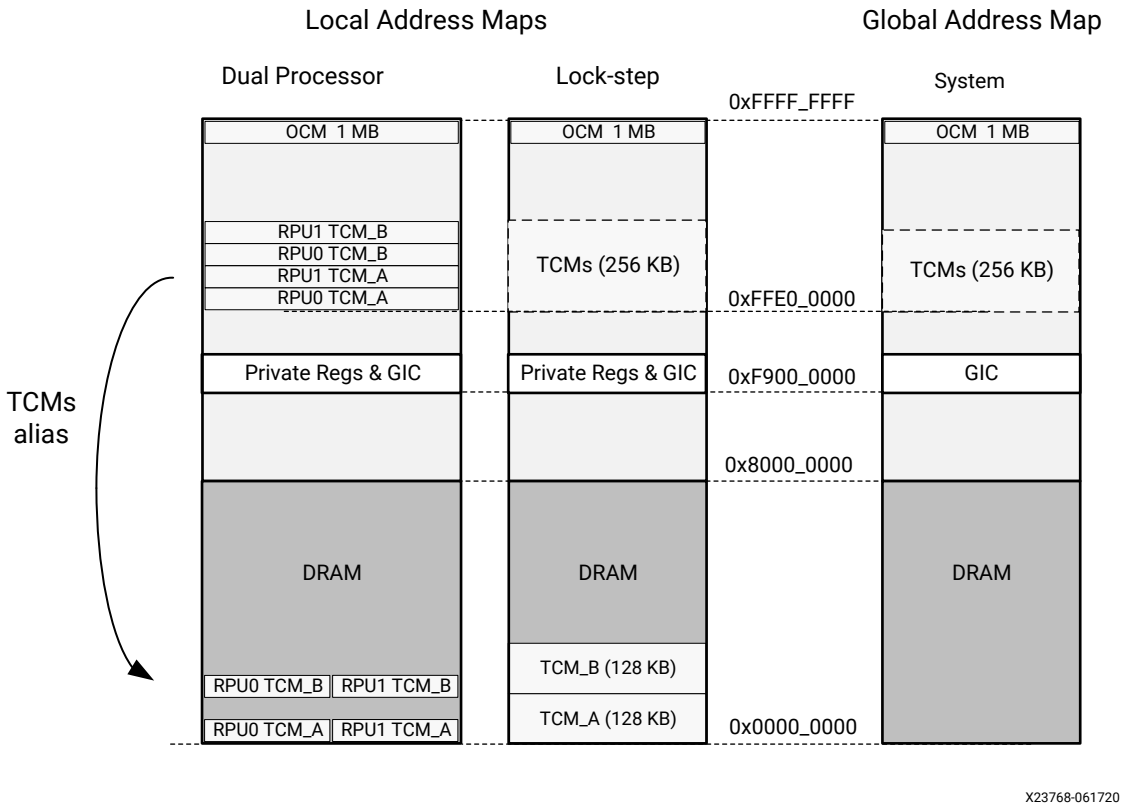
Table 54: **RPU Slaves in a Global Address Map**

Slave	Address Block Size (KB)	Base Address
RPU 0 TCM_A	64	0xFFE0_0000
RPU 0 TCM_B	64	0xFFE2_0000
RPU 0 instruction cache	32	0xFFE4_0000
RPU 0 data cache	32	0xFFE5_0000
Reserved	96	0xFFE6_0000
RPU 1 TCM_A	64	0xFFE9_0000
RPU 1 TCM_B	64	0xFFEB_0000
RPU 1 instruction cache	32	0xFFEC_0000
RPU 1 data cache	32	0xFFED_0000
Reserved	64	0xFFEE_0000

Memory Map Diagram

The basic local and global views of the 4 GB RPU address space are shown in the following figure.

Figure 37: **APU and RPU CPUs TCM Address Map**



The RPU exception vectors can be configured to be HIVEC (`0xFFFF_0000`) or LOVEC (`0x0000_0000`). Because the OCM is mapped at HIVEC, and for the RPU to be able to execute interrupt handlers directly from TCMs, the TCMs must be mapped starting at address `0x0000_0000` (=LOVEC). Also, to configure the APU with LOVEC in DRAM, the APU cannot access TCMs at LOVEC. Consequently, TCMs are aliased into a local address map of the RPU for the Cortex-R5F processor to access them starting at address `0x0000_0000`.

Processor Memory Datapaths

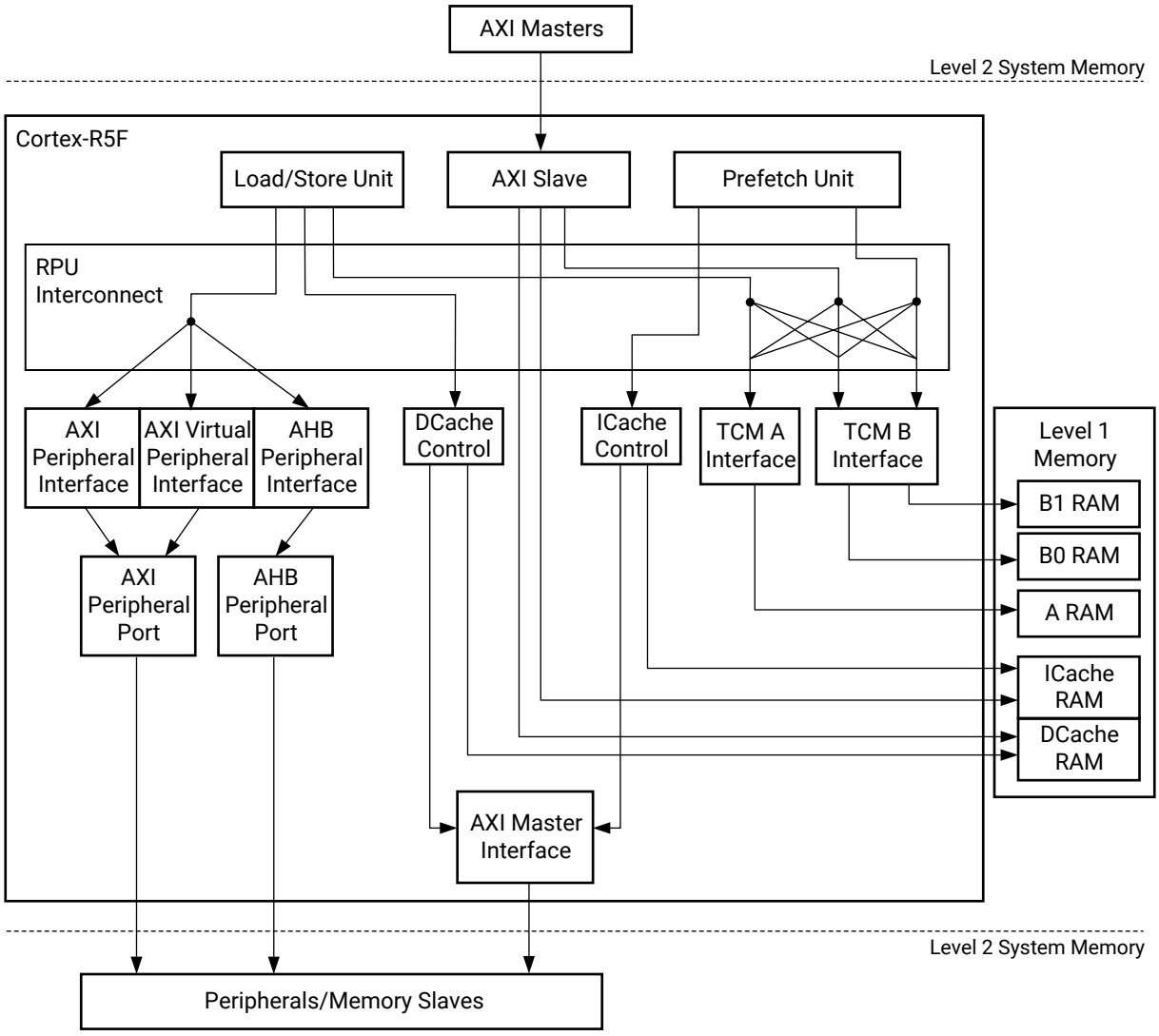
AXI Master Access to CPU Memory

The TCMs can be loaded with code and data coefficients by another system master when the RPU is in idle mode. The RPU and TCMs must be powered on and the RPU must be out of reset.

Datapaths

The datapaths to these memories are shown in the following figure.

Figure 38: **CPU Memory Datapaths**



Tightly Coupled Memories

The low-latency, tightly coupled memories (TCMs) provide predictable instruction execution and predictable data load/store timing for the RPU processors. TCM memory address space is not cached.

Each RPU processor contains two 32 KB memories and one 64 KB memory that are accessed via the TCM A and B port interfaces, for a total of 128 KB. The parallel memory architecture allows concurrent accesses of all three banks by the CPU's load-store unit, instruction prefetch unit, and AXI slave port. The TCM_B includes two 32 KB banks for concurrent, parallel access.

Datapaths are 64-bits wide and are protected by ECC. Each CPU includes:

- 64 KB in TCM_A
- 32 KB in TCM_B0
- 32 KB in TCM_B1

TCMs are accessible after the processor is taken out of reset. The processor must be inactive (idle) or in the halt state to allow another master to access the TCMs. The processors have direct connections to their TCMs for low-latency access and there are no protection units.

The datapaths through the RPU are shown in Processor Memory Datapaths.

Usages

The TCMs can be used for any purpose, but are typically used as follows:

- TCM_A for interrupt or exception code for high speed, without cache miss delays
- TCM_B for data in process-intensive applications such as audio or video processing

Power Islands

The PSM controls power islands for each 64 KB TCM bank using register controls. The power islands are described in Power Islands.

Memory Error Detection and Correction

The processor provides error checking and correction (ECC) data hardware.

The ECC bits are computed on 32-bit data sets; they are computed and then stored in memory with the data. When the data is accessed, the hardware can detect one and two-bit errors within the 32-bit data and its ECC bits. The hardware detects all two-bit errors and can correct single-bit errors, which is sometimes referred to as a single-error correction, double-error detection (SEC-DED) ECC scheme.

RPU Memory Protection Unit

The RPU memory protection unit (MPU) works with the CPU's L1 memories to control the accesses to and from the TCMs, caches, and external memory.

For a detailed description of the MPU, see the *Cortex-R5F Technical Reference Manual*.

The MPU partitions memory into regions and sets individual protection attributes for each region. When the MPU is disabled, no access permission checks are performed and memory attributes are assigned according to the default memory map. The MPU divides memory into a maximum of 16 regions.

The following can be specified for each region using the MPU memory region programming registers:

- Region base address
- Region size
- Sub-region enables
- Region attributes
- Region access permissions
- Region enable

Interrupts

The RPU includes local and system interrupt controllers.

Interrupt Types

The controller supports the following types of interrupts:

- **Shared peripheral interrupts:** Shared peripheral interrupts (SPIs) are general-purpose interrupts generated by various sources in the system and manged by the GIC RPU interrupt controller. The SPI interrupts are listed in the System Interrupts chapter.
- **Software generated interrupts:**
  - Software generated interrupts (SGIs) are inter-processor interrupts that are generated by writing to the software generated interrupt register (GICD_SGIR).
  - There are 16 SGIs available for each processor, and they have no effect on the hardware.

General Interrupt Controller and Configurations

The general interrupt controller (GIC) is based on the Arm GIC-390 and it is configurable.

- Security state for an interrupt
- Priority level of an interrupt
- Enabling or disabling of an interrupt
- Processors that receive an interrupt

GIC Programming Interface

The GIC distributor receives interrupts and provides the highest priority interrupt to the CPU interface. An interrupt with a lower priority is forwarded when it becomes the highest priority pending interrupt.

The GIC CPU interface has a priority mask and only accepts a pending interrupt if it is:

- Higher priority than the programmed interrupt mask, and
- Higher priority than the interrupt the processor is currently servicing

System Interrupts Generated by RPU

The RPU generates the system interrupts listed in the following table. These interrupts are also listed in IRQ System Interrupts.

Table 55: **RPU Generated System Interrupts**

Description	System Interrupt	IRQ #	Notes
Performance monitor (APM)	RPU0_PERF_MON RPU1_PERF_MON	40 41	
Miscellaneous processor errors	RPU0_ERR RPU1_ERR	43 44	Synchronization between RPU SW and PL

GIC Interrupt Controller

There are two interfaces between the RPU MPCore and the RPU GIC.

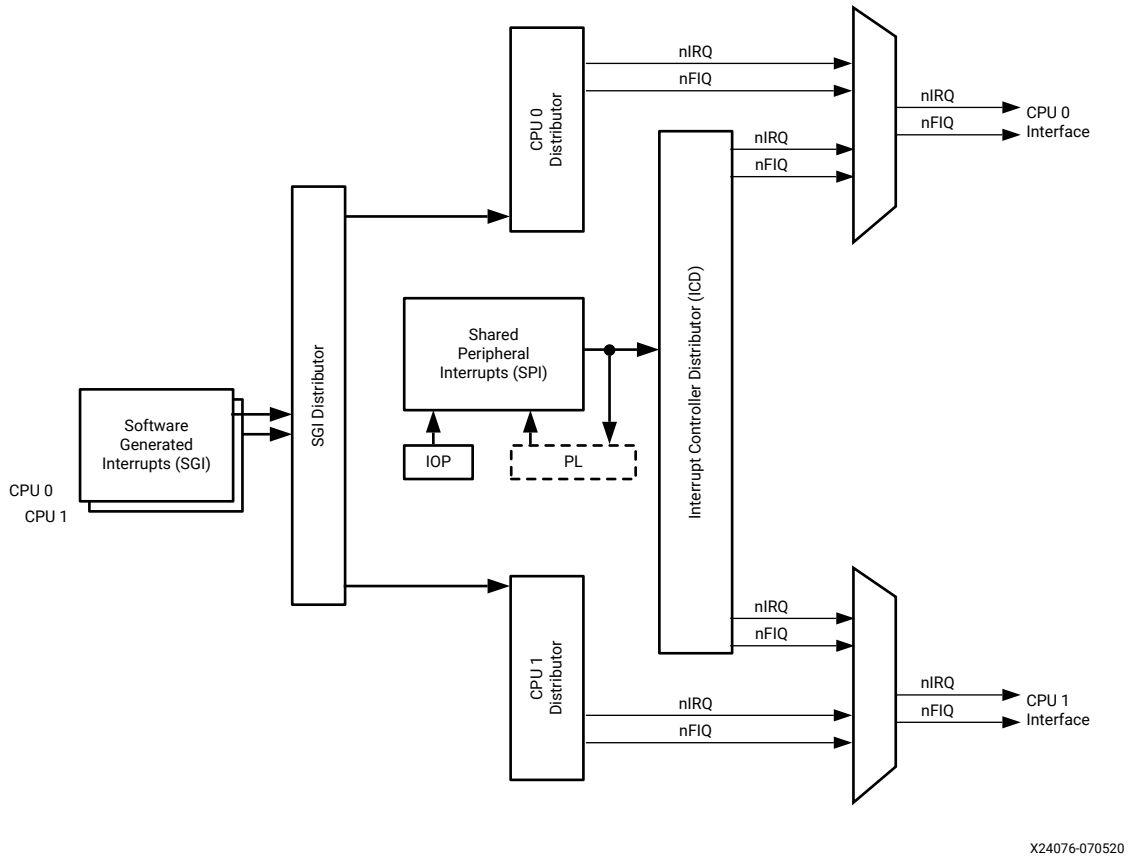
- The distributor interface is used to assign the interrupts to each of the Cortex™-R5F MPCore processors.
- The CPU interface with a separate set of 4 KB memory-mapped registers for each CPU provides protection against unwanted accesses by one CPU to interrupts that are assigned to the other.

The RPU MPCores processors access the RPU_GIC interrupt controller through their peripheral interface switch. The low-latency peripheral interfaces are really designed for strongly ordered or device type accesses, which are restrictive by nature. Memory that is marked as strongly ordered or device type is typically sensitive to the number of reads or writes performed. Consequently, instructions that access strongly ordered or device memory are never abandoned when they have started accessing memory. These instructions always complete either all or none of their memory accesses. The same is true of all accesses to the low-latency peripheral port, regardless of the memory type.

Block Diagram

The GIC block diagram is shown in the following figure with SGI and SPI interrupt inputs.

Figure 39: **RPU GIC Block Diagram**



Software Generated Interrupts

Each CPU can interrupt itself, the other CPU, or both CPUs within the MPCore using a software generated interrupt (SGI). There are 16 software generated interrupts. An SGI is generated by writing the SGI interrupt number to the PL390.enable_sgi_control (ICDSGIR) register and specifying the target CPUs. This write occurs through the CPU's own private bus. Each CPU has its own set of SGI registers to generate one or more of the 16 software generated interrupts. The interrupts are cleared by reading the interrupt acknowledge PL390.control_n_int_ack_n (ICCIAR) register or writing to the corresponding bits of the interrupt clear-pending PL390.enable_sqi_pending (ICDICPR) register.

All SGIs are edge triggered. The sensitivity types for SGIs are fixed and cannot be changed. The control register is read-only, because it specifies the sensitivity types of all the 16 SGIs.

Shared Peripheral Interrupts

A group of over 150 system interrupts from various modules can be routed to one or both of the CPUs or the PL. The interrupt controller manages the prioritization and reception of these interrupts for the CPUs. The system interrupts are listed in IRQ System Interrupts.

These system interrupts are routed to the shared peripheral interrupt (SPI) ports on the RPU general interrupt controller (GIC).

SPI Interrupt Sensitivity

The shared peripheral interrupts (SPI) from the system IRQs can be targeted to either of the CPUs, but only one CPU handles the interrupt. If an interrupt is targeted to both CPUs and they respond to the GIC at the same time, the MPCore ensures that only one of the CPUs reads the active interrupt ID#. The other CPU receives the spurious (ID 1023 or 1022) interrupt or the next pending interrupt, depending on the timing.

Except for PL-to-PS interrupt signals (IRQ 116 to 127), all interrupt sensitivity types are hardwired by the requesting sources and cannot be changed. The GIC must be programmed to accommodate this. The BootROM does not program these registers. Consequently, the SDK device drivers must program the GIC to accommodate these sensitivity types.

For an interrupt of level sensitivity type, the requesting source must provide a mechanism for the interrupt handler to clear the interrupt after the interrupt has been acknowledged. This requirement applies to any IRQ-F2P[n] (from PL) with a high-level sensitivity type.

For an interrupt of rising edge sensitivity, the requesting source must provide a pulse wide that is large enough for the GIC to catch. This is normally at least two RPU clocks. This requirement applies to any IRQ-F2P[n] (from PL) with a rising-edge sensitivity type.

The sensitivity control for each interrupt has a 2-bit field that specifies sensitivity type and handling model.

Interrupt Prioritization

All of the SGI and SPI interrupt requests are assigned a unique ID number. The controller uses the ID number to arbitrate. The interrupt distributor holds the list of pending interrupts for each CPU and then selects the highest priority interrupt before issuing it to the CPU interface. Interrupts of equal priority are resolved by selecting the lowest ID.

The prioritization logic is physically duplicated to enable the simultaneous selection of the highest priority interrupt for each CPU. The interrupt distributor holds the central list of interrupts, processors, and activation information, and is responsible for triggering software interrupts to the CPUs.

SGI distributor registers are banked to provide a separate copy for each CPU. The interrupt controller ensures that an interrupt targeting more than one CPU can only be handled by one CPU at a time.

The interrupt distributor transmits the highest pending interrupt to the CPU interface. It then receives information that the interrupt is acknowledged, and can then change the status of the corresponding interrupt. Only the CPU that acknowledges the interrupt can respond and process it.

System Errors Generated by RPU

The RPU generates several type of system errors. The system errors are processed by the system error accumulator described in System Errors.

The system errors generated by the RPU are listed in the following table.

Table 56: **RPU Generated System Errors**

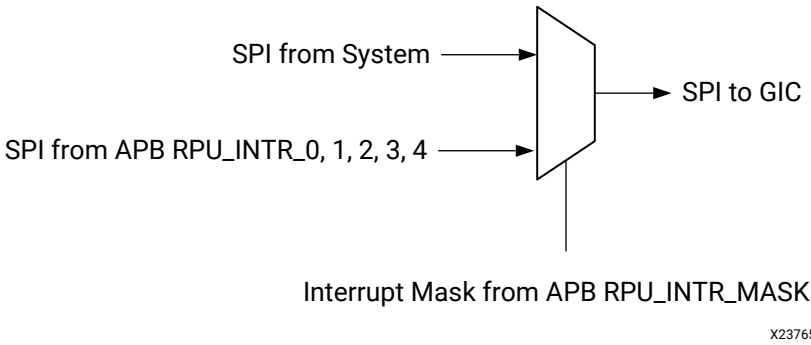
Error Description	Name	Status Register
Safety configuration lock-step	RPU_LS_ERR	PSM_SYSTEM_ERR.PSM_ERR1_STATUS [RPU_ECC]
Safety-mode common-mode failure	RPU_CCF_ERR	PSM_SYSTEM_ERR.PSM_ERR1_STATUS [RPU_CCF]
Uncorrectable memory ECC	RPU_SYS_ERR	PSM_SYSTEM_ERR.PSM_ERR1_STATUS [RPU_LS]

Test and Debug

Interrupt Injection Mechanism

The RPU implements an interrupt injection function to inject interrupts into the generic interrupt controller's shared peripheral interrupts (SPI). The RPU GIC has 160 SPIs. Software can inject an interrupt on each of the over 150 interrupt lines using this mechanism. The SPIs are divided into five, 32-bit APB registers. The RPU implements an interrupt register and an interrupt mask register. The logic in the following figure is replicated on each interrupt going to the SPI of the RPU's GIC. If the interrupt mask corresponding to the interrupt is set in the RPU_INTR_MASK register, the RPU passes the APB register version of the interrupt to the GIC.

Figure 40: **Interrupt Injection**



The following table lists the mapping of the SPI bits.

Table 57: **SPI Map to RPU Interrupt and RPU Interrupt Mask Registers**

SPI	RPU Interrupt Register	RPU Interrupt Mask Register
SPI<31:0>	RPU_INTR_0<31:0>	RPU_INTR_MASK_0<31:0>
SPI<63:32>	RPU_INTR_1<31:0>	RPU_INTR_MASK_1<31:0>
SPI<95:64>	RPU_INTR_2<31:0>	RPU_INTR_MASK_2<31:0>
SPI<127:96>	RPU_INTR_3<31:0>	RPU_INTR_MASK_3<31:0>
SPI<159:128>	RPU_INTR_4<31:0>	RPU_INTR_MASK_4<31:0>

Events and Performance Monitor

The processor includes logic to detect various events that can occur, for example, a cache miss. These events provide useful information about the behavior of the processor to use when debugging or profiling code.

The events are made visible on an output event bus and can be counted using registers in the performance monitoring unit.

Register Reference

There are several register sets associated with the RPU:

- CPU control and status
- Interrupts
- System-level control

Processor Control and Status Registers

The following table provides an overview of the RPU system control and status registers.

Table 58: **RPU Control and Status Register Overview**

Register Name	Address Offset	Access Type	Description
GLOBAL_CNTL	0x000	RW	Global control
GLOBAL_STATUS	0x004	R	Miscellaneous status information
ERROR_CNTL	0x006	RW	Error response enable/disable
CCF_VAL	0x040 - 0x050	RW	Common cause signal value
CCF_MASK	0x054	RW	Common cause signal mask
SAFETY_CHK	0x0F0	RW	Safety check register
RPU0_CONFIG RPU1_CONFIG	0x100 0x200	RW	Configuration parameters
RPU0_STATUS RPU1_STATUS	0x104 0x204	R	RPU status
RPU0_PWRDWN RPU1_PWRDWN	0x108 0x208	RW	Power-down request from the CPU
RPU0_ISR, RPU1_ISR	0x114, 0x214	WTC	Interrupt status
RPU0_IMR, RPU1_IMR	0x118, 0x218	R	Interrupt mask
RPU0_IER, RPU1_IER	0x11C, 0x21C	W	Interrupt enable
RPU0_IDR, RPU1_IDR	0x120, 0x220	W	Interrupt disable
RPU0_SLV_BASE, RPU1_SLV_BASE	0x124 0x224	RW	Slave base address
RPU0_AXI_OVERRIDE RPU1_AXI_OVERRIDE	0x128 0x228	RW	AXI master attribute override

XILINX

Chapter 29

Application Processing Unit

This chapter contains these main sections:

The application processing unit (APU) provides general-purpose computing in a standard programming environment based on powerful and feature-rich Arm® Cortex™-A72 cores with their A64 instruction set in the v8-A architecture. The APU includes two A72 cores. The generic interrupt controller (Arm GIC-500) is added to manage system interrupts. Other processors and bus masters can interact with the APU L2 cache memory with error-correction code (ECC) to form a tightly coupled heterogeneous system using the Cache Coherent Interconnect (CCI). The APU is located in the FPD of the PS.

A72 Processor Implementation

The TRM provides an overview of the processor features and implementation notes for the Versal™ device. An extensive set of documentation is available from Arm. The introduction to Arm processors and documentation begins at the Arm developer architectures website. The IP version is listed in IP Versions.

Features

CPU Pipelines

- Single and double precision floating point unit, VFPv4 (see the Arm developer website for more information)
- NEON single instruction multiple data (SIMD) extension, (see the Arm developer website for more information)
- Cryptography extension, see Arm TRM document 100097_0003_05

Caches

The architecture supports hardware virtualization. Each Cortex-A72 processor includes a 48-KB L1 instruction cache with parity protection and a 32 KB L1 data cache with ECC protection.

The processors have a built-in two-stage MMU that supports multi-threading and multi-operating system applications. Masters in all parts of the system can potentially participate in the APU L2 cache coherency address space by routing their transactions through the system memory management unit (SMMU) via its translation buffer units (TBU) that are connected to the CCI in the FPD. The SMMU maps the virtual addresses of masters to the shared physical address space in main memory.

Power islands include:

- Each processor core can be enabled and disabled individually using its own power island
- L2 cache power island

The processor also includes the GIC-500 interrupt controller with its GIC v3 architecture.

To support real-time debug and trace, each processor has an embedded trace macrocell (ETM) that communicates with the Arm CoreSight™ debug system.

Comparison to Previous Generation Xilinx Devices

The APU MPCore in the Versal device uses a dual Cortex-A72 MPCore compared to a quad Cortex-A53 MPCore in the Zynq® UltraScale+™ MPSoC. The Cortex-A72 is newer and has significantly more performance than the Cortex-A53, but also requires more power. Key pipeline features include:

- Three-way instruction dispatch instead of two
- Out-of-order execution
- Faster clocking

The FPD upgraded from the CCI-400 to the CCI-500 cache coherency interface. The SMMU-500 is in both the Zynq UltraScale+ MPSoC and the Versal ACAP.

APU MMU

The APU MMU is similar to the unit used in the Cortex-A53 processor from previous Xilinx devices with some enhancements:

- AArch64 state provides 44 bits of physical address size
- 48-entry L1 instruction TLB
- 32-entry L1 data cache TLB
- 1024-entry L2 cache TLB

GIC Interrupt Control

The APU has a built-in interrupt controller for virtual interrupts. It also has an attached system interrupt controller based on the Arm GIC-500. The programming model for interrupts is significantly different than in the Zynq UltraScale+ MPSoC.

Arm Server Base System Architecture

The APU includes features to support the Arm server base system architecture (SBSA). These include:

- APU Cortex-A72 (supersedes Cortex-A53)
- APU GIC-500 (Arm v3 architecture protocol supersedes v2)

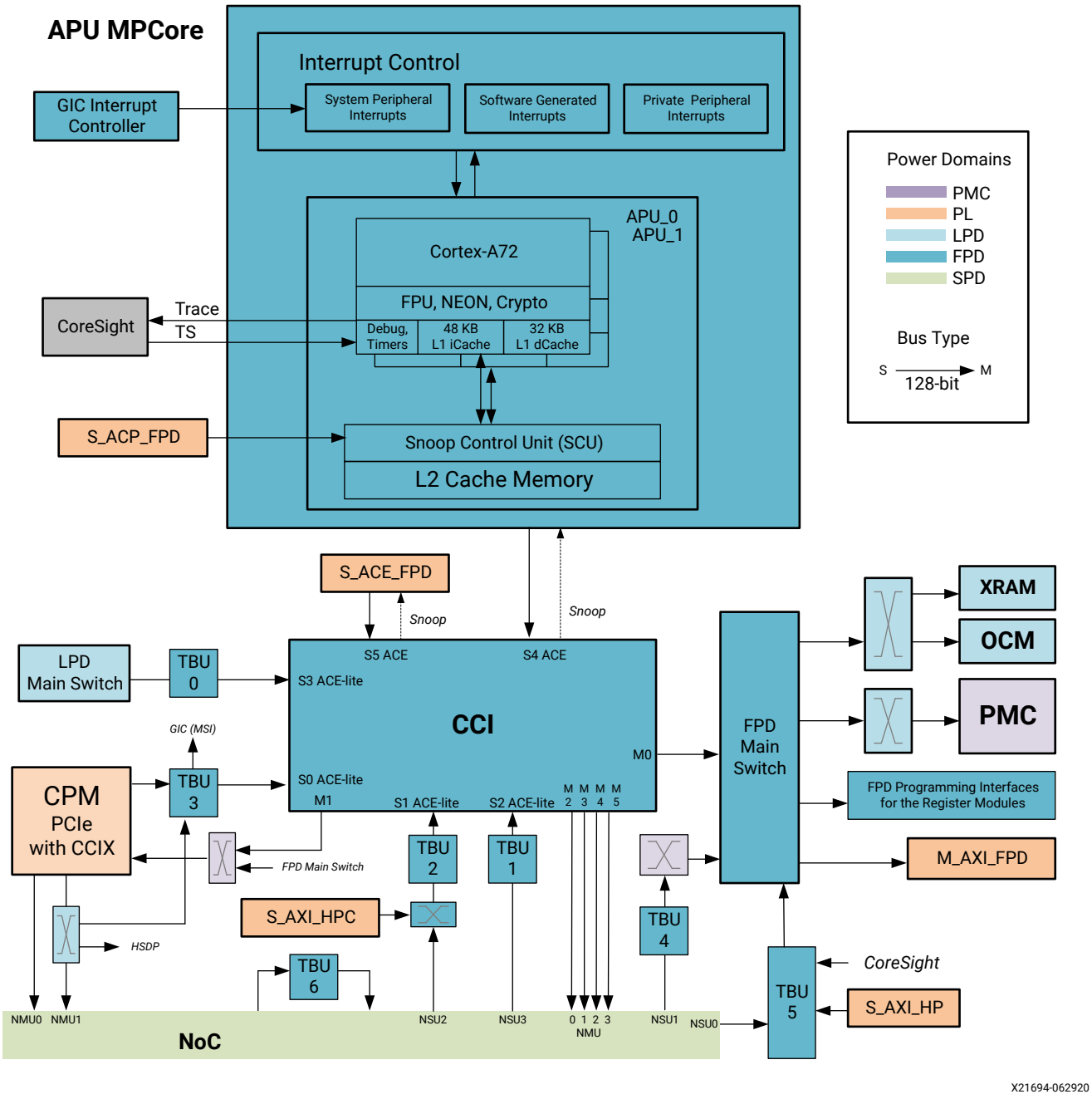
System Perspective

The APU is central to the FPD and works closely with the APU L2 cache coherent interconnect (CCI) and the system memory management unit (SMMU). The AMBA® interconnect provides access to the LPD, PL, CPM and NoC.

Block Diagram

The APU block diagram is shown in the following figure.

Figure 41: **APU High-level Block Diagram**



X21694-062920

Functional Units

Processor Pipelines

There are several pipelines and engines in the APU:

- CPU Pipeline with AArch32 and AArch64 instructions

- FPU Pipeline with 32 and 64-bit data
- NEON Pipeline with single instruction multiple data (SIMD) dispatch
- Cryptography Engine

L1 Caches

Each processor includes its own local L1 cache connected to the SCU and L2-cache memory.

Cache features include:

- 48 KB instruction cache protected with parity and includes a 48-entry fully associative TLB
- 32 KB data cache protected with ECC and includes a 32-entry fully associative TLB

L2 Cache

The 1 MB, unified L2 cache with ECC is physically addressed and physically tagged.

- 4-way set associative 1024-entry TLB
- PL can be coherent using S_ACE_FPD, S_ACP_FPD, S_AXI_HPC, NSU2, and NSU3 interfaces
- CPM can be coherent using ACE-Lite port via SMMU TBU 3

System Interfaces

The APU MPCore accesses the system through the Cache Coherency Interconnect (CCI). The CCI also connects other system masters to the APU MPCore for two-way and I/O coherency.

Execution Pipelines

CPU Pipeline

The CPU pipeline includes:

- Variable length, super-scalar pipeline (up to 15 stages) with out-of-order execution
- Arm Arch64 v8A CPU architecture
- Arm Arch32 capable for legacy applications
- Dynamic branch prediction with branch target buffer and global history buffer, a return stack, and an indirect predictor

FPU Pipeline

The floating point unit pipeline includes:

- VFPv4 execution in each core

NEON Pipeline

The NEON pipeline includes:

- Advanced SIMD extensions
- Arm v8-A architecture

Cryptography Engine

The cryptography engine builds on the advanced SIMD instruction set. The crypto engine can use to accelerate the execution of AES, SHA, and SHA2-256 algorithms.

See the Arm *Architecture Reference Manual v8* for more information.

APU Address Model

APU Addressing

In the AArch32 state, the Arm® v8 address translation system resembles the Arm v7 address translation system with large physical-address extensions (LPAE) and virtualization extensions.

In AArch64 state, the Arm v8 address translation system resembles an extension to the long descriptor format address translation system to support the expanded virtual and physical address spaces. For more information regarding the address translation formats, see the Arm *Architecture Reference Manual v8* for the Arm v8-A architecture profile.

The memory management unit (MMU) controls table-walk hardware that accesses translation tables in main memory. The MMU translates virtual addresses to physical addresses. The MMU provides fine-grained memory system control through a set of virtual-to-physical address mappings and memory attributes held in page tables. These are loaded into the translation lookaside buffer (TLB) when a location is accessed.

Address translations can have one or two stages. Each stage produces the least significant bits (LSB) output without a lookup. Each stage walks through multiple levels of translation.

Memory Space

The APU's Cortex-A72 includes an integrated memory management unit (MMU).

In the AArch32 state, the Arm® v8 address translation system resembles the Arm v7 address translation system with large physical-address extensions (LPAE) and virtualization extensions. In the AArch64 state, the Arm v8 address translation system resembles an extension to the long descriptor format address translation system to support the expanded virtual and physical address spaces. For more information on the address translation formats, see the Arm *Architecture Reference Manual v8* for the Arm v8-A architecture profile. The key differences between the AArch64 and AArch32 address translation systems are that the AArch64 state provides the ability to:

- Select the translation granule to either be 4 KB or 64 KB (AArch32 limited to be 4 KB)
- Configure the address space identifier (ASID) size to be either 8-bit or 16-bit (AArch32 limited to an 8-bit value)

The maximum physical address size is:

- 44-bit in AArch64 state
- 40-bit in AArch32 state

The APU memory management unit (MMU) controls table-walk hardware that accesses translation tables in main memory. The MMU works with the L1 and L2 memory system to translate a virtual address (VA) to a physical address (PA). The MMU provides fine-grained memory system control through a set of virtual-to-physical address mappings and memory attributes held in page tables. These are loaded into the translation lookaside buffer (TLB) when a location is accessed.

Address translations can have one or two stages. Each stage produces output LSBs without a lookup. Each stage walks through multiple levels of translation as follows:

- 48-entry fully-associative L1 instruction cache TLB
- 32-entry fully-associative L1 data cache TLB for data load and store pipelines
- 4-way set-associative 1024-entry L2 cache TLB in each processor
- Intermediate table walk caches
- TLB entries contain a global indicator or an ASID to permit context switches without TLB flushes
- TLB entries contain a virtual machine identifier (VMID) to permit virtual machine switches without TLB flushes

Virtualization

Virtualization allows multiple software stacks to run simultaneously on the same processor, which enhances the productivity of the Versal ACAP. The role of virtualization varies from system to system. For some designs, virtualization allows the processor to be kept fully loaded at all times, saving power and maximizing performance. For others, virtualization provides a way to partition the various software stacks for isolation or redundancy.

***Note*:** The support for virtualization applies only to an implementation that includes Arm exception level-2 (EL2). Armv8 supports the virtualization extension to achieve full virtualization with performance comparable to that of the native guest operating system.

The hardware provides virtualization features to support multiple virtual machines running on the APU.

- APU Virtualization
- Virtual Interrupts
- Memory Virtualization
- Processor Counters

Server Architecture

The APU and subsystems include support for the Arm server based system architecture (SBSA). The SBSA architecture aligns hardware with system software components for interoperability. The specification comprises multiple levels that build incrementally on top of each other with each level mandating additional functional aspects of the system. This includes specifying features that the CPU and some key peripherals need to support to be compliant.

The Versal ACAP system is Arm SBSA Level-1 capable, at a minimum, as defined by the Arm SBSA specification document number ARM-DEN-0029A.

The features specifically supporting the SBSA architecture include:

- SBSA L1 compatible components
  - APU Cortex-A72 core
  - UART SBSA
  - SWDT with generic and windowed timers
- APU generic interrupt controller (Arm GICv3 architecture in GIC-500)
  - Locality-specific peripheral interrupt (LPI)

    - System register interface enable (SRE)
    - Affinity routing enable (ARE)
- System firmware data structures such as ACPI or FDT

Processor Counters

- Physical Counter, also see System Counter
- Virtual Counters
- Debug Counter

Applications

The system counter can be used to generate one or more event streams to generate periodic wake-up events. An event stream might be used for these reasons:

- To impose a timeout on a wait-for-event polling loop
- To safeguard against any programming error that means an expected event is not generated

Event Stream

An event stream is configured by these selections:

- Selecting which bit from the bottom 16 bits of a counter triggers the event, which determines the frequency of the events in the stream
- Selecting whether the event is generated on each 0 to 1 transition or each 1 to 0 transition of the selected counter bit

Physical Counter

The physical and virtual counters in the Arm v8 architecture are sourced from the 64-bit system counter located in the LPD. For details, see System Counter.

The system counter provides the time base for the physical and virtual counters for the APU processors. The system count is also accessible using memory-mapped registers in the LPD memory space.

The count value for the APU processor's physical counter is the same as the system counter. For virtual count, a fixed count for a virtual channel is subtracted from the system counter value to provide a virtual count. The physical and virtual counters for the APU processors are documented in the Arm *Architecture Reference Manual Arm v8*.

The system counter is controlled by the IOP_SCNTR and IOP_SCNTRS register sets. The frequency of the system counter tic clock is controlled by the CRL.TIMESTAMP_REF_CTRL register. For more information, see System Counter.

Software can read the CNTFRQ register to determine the current system counter frequency in these states and modes:

- 64-bit counter is private to each APU core
- Same PPI interrupt number for each APU core
- Extensions to the timer to AArch64:
  - When CNTKCTL.EL0PCTEN is set to 1, secure and non-secure EL0 modes
  - Non-secure EL1 physical timer
  - Secure EL1 physical timer
  - Non-secure EL2 physical timer
  - Virtual timer based on offset from physical timer

Accessing the Physical Counter Registers

The processor physical counter is implemented as the system counter. The functionality and memory-mapped access methods are described in System Counter. The physical counter is also accessible via the processor's local registers as described in this section. For each counter, all counter registers have the same access permissions. Software with sufficient privileges can read CNTPCT using a 64-bit system register read.

EL1 Physical Counter

The EL1 physical counter is accessible from EL1 modes, except that non-secure software executing at EL2 controls access from non-secure EL1 modes.

When access from EL1 modes is permitted, CNTKCTL.EL0PTEN determines whether the registers are accessible from EL0 modes. If an access is not permitted because CNTKCTL.EL0PTEN is set to 0, an attempted access from EL0 is UNDEFINED.

The EL1 physical timer characteristics include:

- Except for accesses from the monitor mode, accesses are to the registers in the current security state.
- For accesses from monitor mode, the value of SCR_EL3.NS determines whether accesses are to the secure or the non-secure registers.
- The non-secure registers are accessible from hypervisor mode.

- CNTHCTL.NSEL1TPEN determines whether the non-secure registers are accessible from non-secure EL1 modes. If this bit is set to 1, to enable access from non-secure EL1 modes CNTKCTL.EL0PTEN determines whether the registers are accessible from non-secure EL0 modes.

If an access is not permitted because CNTHCTL.NSEL1TPEN is set to 0, an attempted access from a non-secure EL1 or EL0 mode generates a hypervisor trap exception. However, if CNTKCTL.EL0PTEN is set to 0, this control takes priority, and an attempted access from EL0 is UNDEFINED.

EL2 Physical Counter

The EL2 physical counter is accessible from non-secure hypervisor mode, and from the secure monitor mode when SCR_EL3.NS is set to 1.

Virtual Counters

Each APU core includes a virtual counter that indicates virtual time. The virtual counter contains the value of the physical counter minus a 64-bit virtual offset. When executing in a non-secure EL1 or EL0 mode, the virtual offset value relates to the current virtual machine. The CNTVOFF register contains the virtual offset. CNTVOFF is only accessible from EL2 or EL3 when SCR.NS is set to 1. The CNTVCT register holds the current virtual counter value.

Accessing the Virtual Counter

Software with sufficient privilege can read CNTVCT using a 64-bit system register read.

The virtual counter is accessible from secure and non-secure EL1 modes and from hypervisor mode. CNTKCTL.EL0VTEN determines whether the registers are accessible from EL0 modes. If an access is not permitted because CNTKCTL.EL0VTEN is set to 0, an attempted access from an EL0 is UNDEFINED.

Private Counters

- 64-bit counter is private to the APU MPCore
- Auto-incrementing feature
- 64-bit comparator can assert a private interrupt

Local Processor Access

Typically, initializing and reading the system counter frequency includes setting the system counter frequency using the system register interface, only during the system boot process. The system counter frequency is set by writing the system counter frequency to the CNTFRQ register. Only software executing at the highest exception level implemented can write to CNTFRQ.

Programming

There are several control functions as described in this section.

Local Memory-mapped System Registers

- Enabling and disabling the counter
  - CNTR, counter control register EN, bit [0]:
    - 0: System counter disabled
    - 1: System counter enabled
- Setting the counter value

  Two contiguous RW registers CNTCV [31:0] and CNTCV [63:32] that hold the current system counter value, CNTCV. If the system supports 64-bit atomic accesses, these two registers must be accessible by these accesses.
- Changing the operating mode to change the update frequency and increment value. CNTCR, counter control register FCREQ, bits [31:8]: frequency change request.
- Enabling halt-on-debug for a debugger to use to suspend counting. CNTCR, counter control register HDBG, bit [1]: Halt-on-debug. Controls whether a halt-on-debug signal halts the system counter.
  - 0: System counter ignores halt-on-debug
  - 1: Asserted halt-on-debug signal halts system counter update

Interrupts

Interrupt Types

The controller supports the four types of interrupts listed here.

- **Shared peripheral interrupt:** Shared peripheral interrupts (SPIs) are peripheral interrupts that can be routed to a specific processor core that can handle the interrupt or a core that is configured to receive this type of interrupt. These interrupts can be group 0 or group 1, and can be either wire-based or message-based.
- **Private peripheral interrupt:** Private peripheral interrupts (PPIs) target a single specific processor core and are independent for each core in the APU cluster. These interrupts are used when the peripherals are tightly coupled to a particular core, can be group 0 or group 1, and are only wire-based.

- **Software generated interrupts:** Software generated interrupts (SGIs) are inter-processor interrupts. SGIs can be generated by writing to the software generated interrupt register (GICD_SGIR). There are 16 SGIs available for each processor in the MPCore. These interrupts have no effect on the hardware.
- **Locality-specific peripheral interrupts for virtualization:** Locality-specific peripheral interrupts (LPIs) are targeted peripheral interrupts that are routed to a specific processor in the MPCore. LPIs can only be non-secure group 1 interrupts and only with edge-triggered behavior. These interrupts are generated by a peripheral writing to a memory-mapped register in the GIC-500 interrupt controller and, consequently, are only message-based interrupts. The GIC-500 supports up to 56k LPIs. The cache size for frequently occurring MSI/MSI-x is 64 entries. The device ID is delivered to the GIC via the AWUSER bits.
- PPI#22 – DCC interrupt
- PPI#23 – PMC overflow
- PPI#24 – CTI interrupt
- PPI#25 – Virtual interface management
- PPI#26 – Hypervisor timer
- PPI#27 – Virtual timer
- PPI#28 – Legacy PL FIQ
- PPI#29 – Secure timer
- PPI#30 – Non-secure timer
- PPI#31 – Legacy PL IRQ

Processor Interrupt Groups

The APU interrupt controller exception level (EL) grouping is:

- Group 0: interrupt is expected to be handled at EL3
- Group 1: secure interrupt is expected to be handled at secure EL1
- Group 1: non-secure interrupt is expected to be handled at:
  - EL2 in systems using virtualization, or
  - EL1 in systems without virtualization

Virtual Interrupts

The Arm GIC v3 interrupt controller (GIC-500) provides hardware virtualization.

Interrupt Translation Services

The APU interrupt controller provides interrupt translation services (ITS) to isolate the device and provide ID translation for message-based interrupts. This enables virtual machines to program devices directly.

LPI and ITS Cache Updates

GIC-500 has a cache to store the settings for LPI interrupts and interrupt translation (ITS) for the message-based protocol. A cache miss results in up to three round trips to memory.

GIC Interrupt Controller

The APU processor includes a local interrupt controller for managing APU processor related interrupts. It is attached to the generic interrupt controller (GIC-500) to capture system interrupts.

To manage system interrupts, the APU includes the GIC interrupt controller, which is based on the Arm GIC-500 generic interrupt controller and is compatible with the Arm GIC v3 architecture.

Test and Debug

Debug Counter

TSGEN is the timestamp generator in the CoreSight debug module in the APU. The CNTCR register controls the counter operation by enabling, disabling, or halting the counter. Normally, it is 100 MHz after boot, but the frequency can be changed using the DBG_TSTMP_CTRL register.

The APU processors private timers clock is controlled by the CRL.DBG_TSTMP_CTRL register.

Register Reference

Processor Control and Status Registers

The following tables provide an overview of the AArch32 registers and the APU core private counters. The MPCore timers are defined by the AArch64 architecture specification.

Local Register Access

The CNTPCT register holds the current physical counter value. The CNTPCT counter operates in the LPD power domain to provide a reliable and uniform view of the system time to each of the APU cores. This counter is controlled by the TIMESTAMP_REF_CTRL register.

Table 59: **AArch32 Register Overview**

Function	Control Register
Timer frequency	CNTFRQ
Kernel control	CNTKCTL
Hypervisor control	CNTHCTL
Virtual offset	CNTVOFF

Table 60: **APU Core Private Counter (AArch64)**

Counter - Timer	Physical Counter	Virtual Counter	Physical Secure Counter	Hypervisor Physical Counter
Timer value	CNTP_TVAL_EL0	CNTV_TVAL_EL0	CNTPS_TVAL_EL1	CNTHP_TVAL_EL2
Timer control	CNTP_CTL_EL0	CNTV_CTL_EL0	CNTPS_CTL_EL1	CNTHP_CTL_EL2
Compare value	CNTP_CVAL_EL0	CNTV_CVAL_EL0	CNTPS_CVAL_EL1	CNTHP_CVAL_EL2
Timer count	CNTPCT_EL0	CNTVCT_EL0		

LPD DMA

The general purpose LPD DMA supports memory to memory and memory to I/O buffer transfers. The DMA performs burst transfers with its 128-bit AXI master interface. The DMA has eight separate channels. The channels are controlled by descriptor tables stored in system memory. The DMA also has simple programming modes.

The controller is an AXI4 master on the LPD main switch. Each channel can be independently enabled or disabled at any time. The DMA supports pause functionality per-channel, which allows software to pause the channel on a descriptor and allows software to program new sets of descriptors. Software can resume the channel after it has programmed a new set of descriptors.

The DMA implements a common buffer that is sized to allow the DMA to use the AXI4 bandwidth available. All channels share the common buffer. A common structure is automatically managed by hardware where software enables and disables the channel without concern for the allocation of buffers per channel. Each channel uses the available buffer on a first-come first-served basis. Buffer usage of each channel can be controlled by programming the issuing capability of each channel and rate control. The DMA supports two modes of operation, simple register-based DMA or descriptor-based scatter-gather DMA.

The DMA implements independent source (SRC) and destination (DST) descriptors that can transfer any size payload up to 1 GB. Descriptor payloads can start and end at any alignment. For some AXI slaves, over fetch of data is not allowed on the read channel. For these slaves, software can disable the over fetch feature. Software can independently enable/disable over fetch on a DMA channel basis. Over-fetch disable can significantly impact DMA efficiency (depends on payload alignment). Xilinx recommends only using this feature if it is supported by the target slave.

Functional Description

The controller is an AXI4 master on the LPD main switch.

The DMA is an AXI4 master. Each channel can be independently enabled or disabled at any time. The DMA supports pause functionality per-channel, which allows software to pause the channel on a descriptor and allows software to program new sets of descriptors. Software can resume the channel after it has programmed a new set of descriptors.

The DMA implements a common buffer that is sized to allow the DMA to use the AXI4 bandwidth available. All channels share the common buffer. A common structure is automatically managed where software enables and disables the channel without concern for the allocation of buffers per channel. Each channel uses the available buffer on a first-come first-served basis. Buffer usage of each channel can be controlled by programming the issuing capability of each channel and rate control. The DMA supports two modes of operation, simple register-based DMA or descriptor-based scatter-gather DMA.

The DMA implements independent SRC and DST descriptors that can transfer any size payload (up to 1 GB). Descriptor payloads can start and end at any alignment. For some AXI slaves, over fetch of data is not allowed on the read channel. For these slaves, software can disable over fetch. Software can independently enable/disable over fetch on each DMA channel. Over-fetch disable can significantly impact DMA efficiency (depends on payload alignment). Xilinx recommends only using this feature if it is required by an AXI slave.

Features

- Eight independent channels
- Scatter-gather descriptor driven mode
- Simple DMA mode

AXI DMA

- 128-bit AXI4 interface, 44/48-bit address
- Data transfers from one block of memory to another
- Descriptor driven with scatter-gather functionality
- Burst length of 16 data transfers
- Source (SRC) and destination (DST) payloads can start and end at any alignment, DMA takes care of 4 KB boundary crossing

Operations

- Over fetching can be enabled/disabled on per channel basis
- Each channel can be programmed as secure or non-secure
- Up to 32 outstanding source transactions per channel
- Periodic transaction scheduling — period can be independently programmed per channel
- Simple register-based DMA and scatter-gather DMA modes
- Hybrid descriptor option in scatter-gather DMA mode

- DMA START, STOP, and PAUSE features
- Interrupt accounting
- Descriptor prefetch to maximize efficiency, 100% efficiency with 128-bit aligned source and destination payloads
- Error recovery
- Incremental and fixed type bursts, fixed bursts only in simple DMA mode
- Independent AXI burst length on both the source and destination sides
- Flow control on a per channel basis via the PL EMIO interface

Read-only DMA mode

- Read data is discarded in this mode
- Available in simple DMA mode

Write-only DMA mode

- Data specified in the control register is written to destination address locations, no read command is issued
- Available in simple DMA mode

Comparison to Previous Generation Xilinx Devices

The LPD DMA is based on Xilinx® IP. The DMA version in the Versal™ ACAP is a slight update from Zynq® UltraScale+™ MPSoCs.

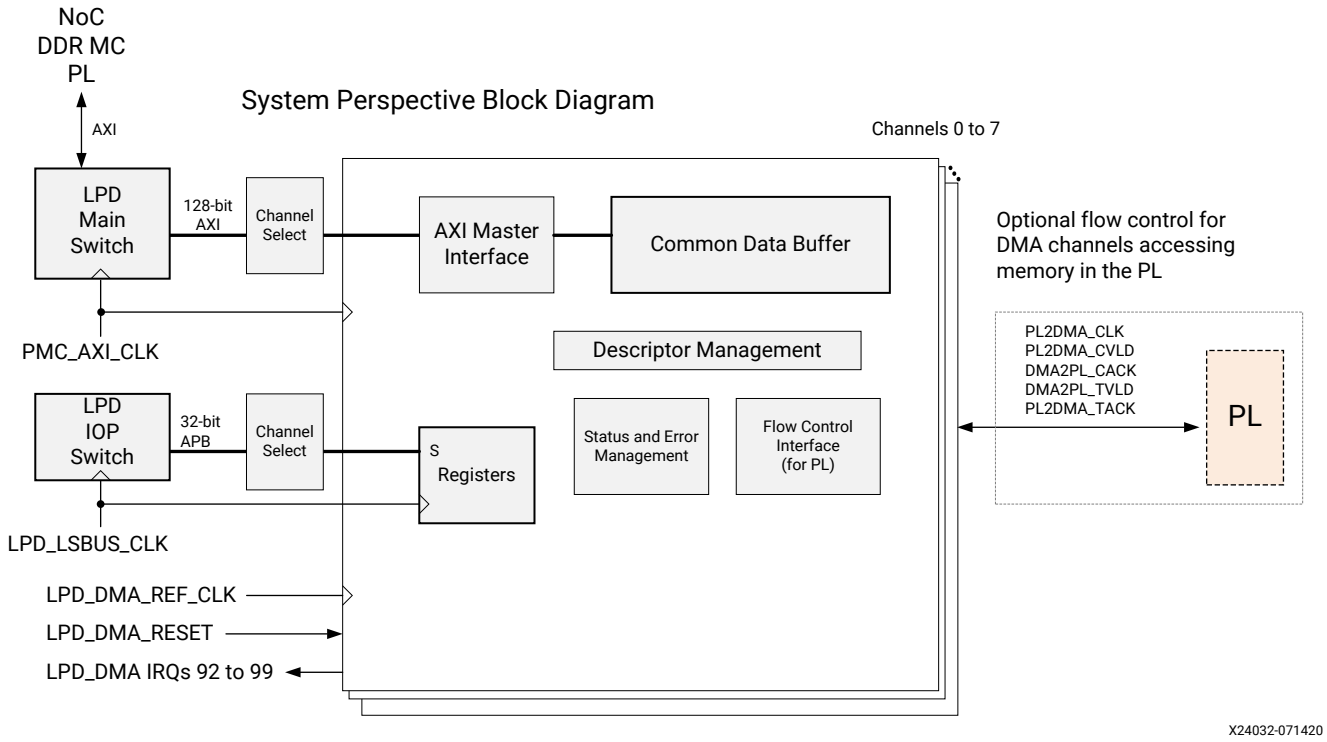
***Note*:** The LPD DMA flow control is done via the PL using the flow control interface (FCI). In the Versal device, the behavior of the DMA2PL_CACK signal is different from that of the Zynq UltraScale+ MPSoC. In Versal devices, the PL must not use DMA2PL_CACK in combinational logic to generate PL2DMA_CVLD. This could cause unpredictable operation.The signal is shown in Flow-Control Interface.

System Perspective

Block Diagram

The system perspective block diagram is shown in the following figure.

Figure 42: **System Perspective Block Diagram**



Functional Units

The major functional blocks include:

- Common buffer
- Arbiter
  - AXI read channel
  - AXI writer channel
- DMA engine channels

Common Buffer

A common buffer is shared between the DMA channels to hold the AXI read transaction data before it goes out on an AXI write channel. The common buffer is sized to allow usage of the full AXI bandwidth. The size of the LPD_DMA common buffer is 4 KB.

- Shares the full buffer space between enabled channels. When only one channel is enabled, it can use the full buffer memory space.
- Does not use/reserve any space in the memory if a channel is disabled (from a previous enable).

- In the event of an error, the DMA channel frees all occupied common buffer entries.
- Shared buffer on a first-come first-served basis.
- Software can limit the common buffer usage of a particular channel by programming read-issuing and rate-control registers. The design of the DMA ensures no starvation on any channel irrespective of their rate control and read issuing parameters.

System Interfaces

AXI Read Arbiter

Each DMA channel uses the AXI master interface to read data descriptor tables and read/write data buffers. The DMA implements round-robin arbitration. Arbitration is never granted to any request if the common buffer does not have enough space. Consequently, the DMA does not put back pressure on the AXI read channel.

If there is not enough space in the common buffer, the arbiter stays parked on the requesting channel until space is available.

AXI Writer Arbiter

The DMA channels share an AXI write channel. The features of the write arbiter are:

- Round-robin arbitration
- Common buffer flush in the event of an error

PL Flow Control Signals

DMA flow control signals are routed to the PL to manage accesses to PL memories. These signals are not used when accessing non-PL memories, which includes DDR, OCM, XRAM, etc.

Programming Guide

The DMA channel control and status registers provide individual channel controllers. In simple DMA mode, these registers are used to move data. In link-list mode, these channels process descriptor tables to move data in memory.

Performance Considerations

The DMA provides more optimal performance when the controller is programmed with the following considerations:

- Read and write descriptor payloads are 128-bit aligned (in scatter-gather mode)

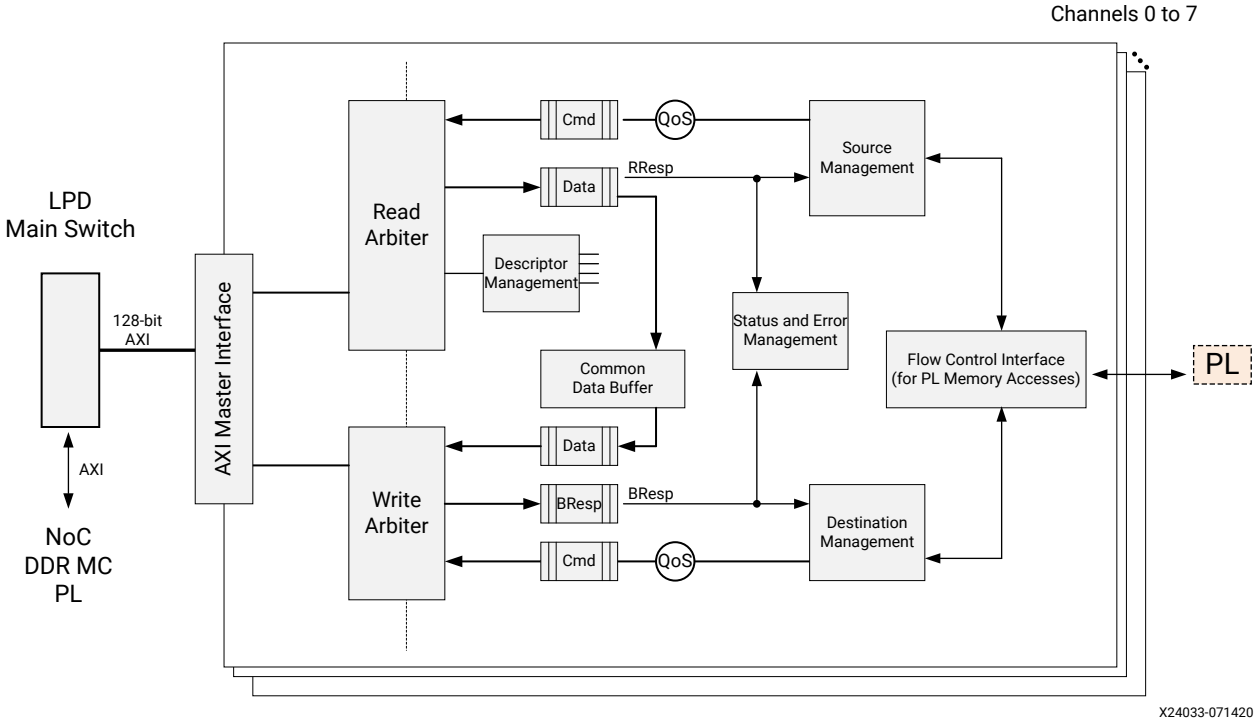
- SRC and DST descriptors are 256-bit aligned
- SRC and DST payload is >4 KB

The guideline is to match the capabilities of the read and write AXI channels and the DMA for the AXI read and write channels.

Channel Block Diagram

The DMA channel is responsible for the DMA operation and management.

Figure 43: **Channel Block Diagram**



Modes and States

The controller has simple and scatter-gather modes with an optional hybrid mode. The controller can be in an active or inactive state.

Modes

Each DMA channel is independently programmed in one of the following DMA modes:

- Simple mode
- Scatter-gather (standard and hybrid modes)

States

- Active
- Inactive

Simple Mode Programming

Simple DMA mode is also known as single-command mode because the DMA performs a data transfer upon receiving a command in a mono-shot manner. In simple DMA mode, the DMA transfer parameters are specified in the control registers. The DMA channel uses these parameters to transfer the data from the SRC to the DST side. This is the single command mode where the DMA channel operation is done after finishing the transfer. Subsequent transfers require the following steps:

1. Update the control registers with new transaction parameters.
2. Enable the DMA channel.

The DMA channel only looks at the SRC size of the transaction. It is always assumed that the transaction size of the DST side is the same as the SRC side's transaction size.

There are simple DMA sub-modes. The read-only and write-only modes are only supported in simple DMA mode. Each channel can be programmed in one of the following sub modes:

- In the read-only mode, the DMA channel reads the data (register specified location) but does not write the data anywhere. This feature can be used to scrub the memory.
- In the write-only mode, the DMA channel reads preloaded data from the control registers and writes it to memory. The DMA channel does not read data from a memory location. Software loads the source data into the registers that are used to write the DST locations. In write-only mode, both SRC and DST registers need to be configured.

Sequence Steps

The sequence steps for simple mode are outlined in this section.

Step 1

Wait until the DMA is in an idle state by reading the STATE field of LPD_DMA.CH_STATUS and ensuring it is either `00` or `11`. In the case where the DMA is in PAUSE state, follow the steps to bring the DMA out from PAUSE as described in Channel Paused.

Step 2

- Ensure that LPD_DMA.CH_CTRL0 [POINT_TYPE] is set = `0`.
- Program the data source buffer address LSB into register LPD_DMA.CH_SRC_DSCR_WORD0.
- Program the data source buffer address MSB into register LPD_DMA.CH_SRC_DSCR_WORD1.

Step 3

- Program the data destination buffer address LSB into register LPD_DMA.CH_DST_DSCR_WORD0.
- Program the data destination buffer address MSB into register LPD_DMA.CH_DST_DSCR_WORD1.

Step 4

- In simple DMA mode, both the SRC and DST transaction sizes must be programmed. The DMA uses the SRC transaction size but it also requires programming both registers. Program the source data size into the LPD_DMA.CH_SRC_DSCR_WORD2 register.
- Program the destination data transaction size into the LPD_DMA.CH_DST_DSCR_WORD2 register. Make sure that the SRC and DST transaction sizes are the same.

Step 5

Optionally, enable an interrupt by setting INTR as a `1` in the LPD_DMA.DST_DSCR_WORD3 and/or LPD_DMA.CH_SRC_DSCR_WORD3 registers.

Step 6

- If the source and destination buffer are allocated in non-cacheable memory or software flushes the caches, there is no need to set COHRNT.
- If the CCI-500 is used for hardware coherency, set COHRNT in the ZDMA_SC_SRC_DSCR_WORD3 and ZDMA_CH_DST_DSCR_WORD3 registers.
- If COHRNT is set, program ARCACHE and AWCACHE in the ZDMA_CH_DATA_ATTR register to indicate a cacheable transaction with a value like `0xF`.

Step 7

Enable the DMA channel to perform DMA transfers by setting the EN bit of LPD_DMA.CH_CTRL2. After enabling DMA, check for possible error conditions as described in Error Conditions.

Descriptor Mode Programming

Transfer parameters are specified in the buffer descriptors (BD). Software programs the SRC and DST BDs and enables a channel. The DMA channel uses the SRC and DST parameters for data transfer in a continuous fashion, as long as there are requests for data transfer in the source BDs and available destination buffers pointed by the destination BD. This can be viewed as memory-to-memory transfer. The channel fetches the first descriptor from the DSCR start address upon receiving an enable.

Data Flow

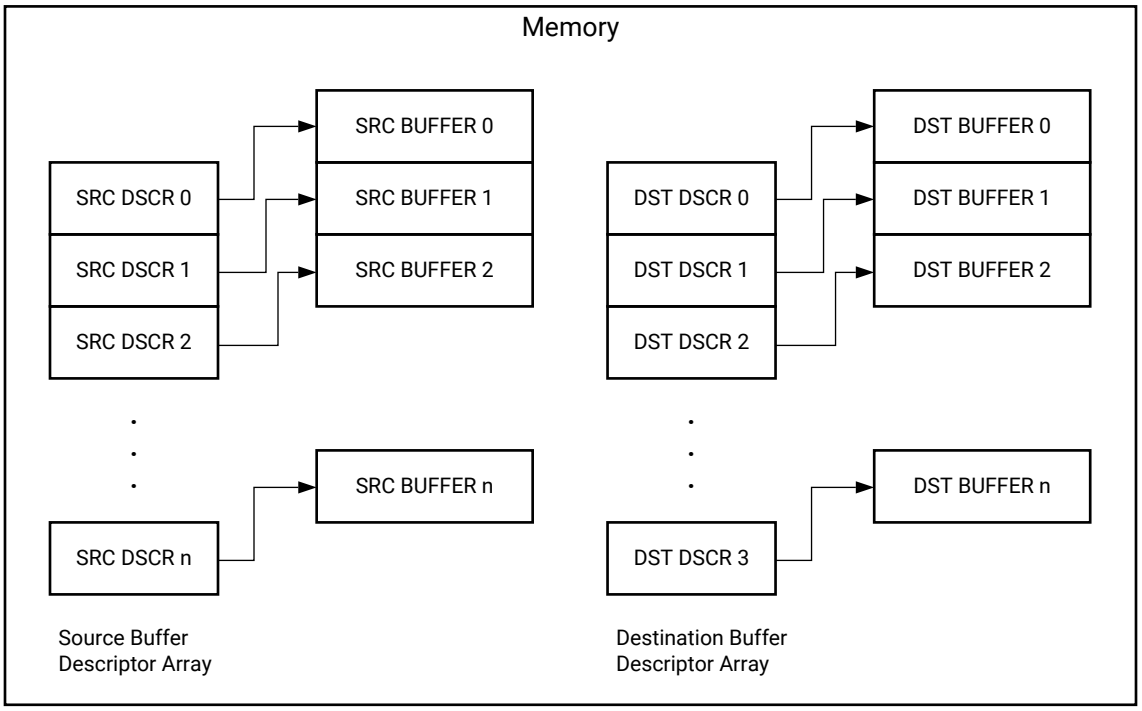
This section outlines the DMA model, modes, and the buffer descriptor (BD) format.

Model

The LPD DMA controller has eight DMA channels. Each channel is divided into two functional sides (in simple DMA mode) or two queues (in scatter-gather DMA mode), source (read) and destination (write).

The schematic in the following figure illustrates the source and destination side scatter-gather mode buffer descriptor arrays. The buffer descriptors (DSCR) point to their respective buffers. The DMA facilitates transfer of data from source (SRC) buffers to destination (DST) buffers. A source side descriptor can go to multiple destination side descriptors.

Figure 44: **SRC and DST Descriptors Pointing to Data Buffers**



Buffer Descriptor Format

The buffer descriptor (BD) format used in scatter-gather mode is shown in the following table. Both the SRC and DST implement the same format descriptor with a few exceptions. Similar words are implemented in the control registers, which can be used in simple DMA mode. By dividing the descriptor into 32-bit words and implementing them on the control registers, a consistent view is provided in both simple and scatter-gather mode.

Table 61: **Buffer Descriptor Format**

Word Number	Field Name	Size (bytes)	Bits	Description
0	ADDR LSB	4	[31:0]	Lower 32 bits of the address pointing to the data/ payload buffer.
1	ADDR MSB	4	[11:0]	Upper 12 bits of the address pointing to the data/ payload buffer.
			[31:12]	Reserved.
2	SIZE	4	[29:0]	Buffer size in bytes (1 G = 2^{30})
			[31:30]	Reserved.

Table 61: **Buffer Descriptor Format** *(cont'd)*

Word Number	Field Name	Size (bytes)	Bits	Description
3	CNTL	4	[0]	Coherency: Reserved
			[1]	DSCR element type: Each descriptor can be viewed as a 128/256-bit descriptor. 0: Current descriptor size is 128 bits (linear) 1: Current descriptor size is 256 bits (linked-list)
			[2]	INTR 0: Completion interrupt is not required 1 (SRC-side): Interrupt is set at the completion of this element. Completion indicates that data is read, but it could be in the DMA buffer (and not yet written to destination). 1 (DST-side): Interrupt is set at the completion of this element. Completion indicates that data is written to the destination location and BRESP is received.
			[4:3]	CMD This field is valid only on a SRC descriptor and is reserved on a DST descriptor. 00: Next DSCR is valid, the DMA channel continues with scatter-gather operation (in this case). Software must ensure that the next descriptor is valid. 01: Pause after completing this descriptor. Software can use this command to pause the DMA operation and update the descriptors. After the software is done updating the descriptors, it can resume the channel from where it paused. If software has updated a descriptor to new location, it can resume the channel and tell it to fetch the descriptor from the new location. Pause mode allows software to keep the state of the channel and avoid the enable sequence. 10: STOP after completing this descriptor. After the DMA channel detects STOP, it finishes the current descriptor payload transfer and goes to IDLE. Any subsequent transfer requires the software to follow an enable sequence. STOP does not preserve the state of the channel. 11: Reserved.
			[31:5]	Reserved.
4	NEXT ADDR LSB	4	[31:0]	Lower 32 bits of the NEXT descriptor address. This field exists only if the DSCR element type is set as 1.
5	NEXT ADDR MSB	4	[11:0]	Upper 12 bits of the NEXT descriptor address.
			[31:12]	Reserved. This field exists only if the DSCR element type is set as 1.
6	Reserved	4	[31:0]	Reserved. This field exists only if the DSCR element type is set as 1.

Table 61: **Buffer Descriptor Format** *(cont'd)*

Word Number	Field Name	Size (bytes)	Bits	Description
7	Reserved	4	[31:0]	Reserved. This field exists only if the DSCR element type is set as 1.

Decriptor Format

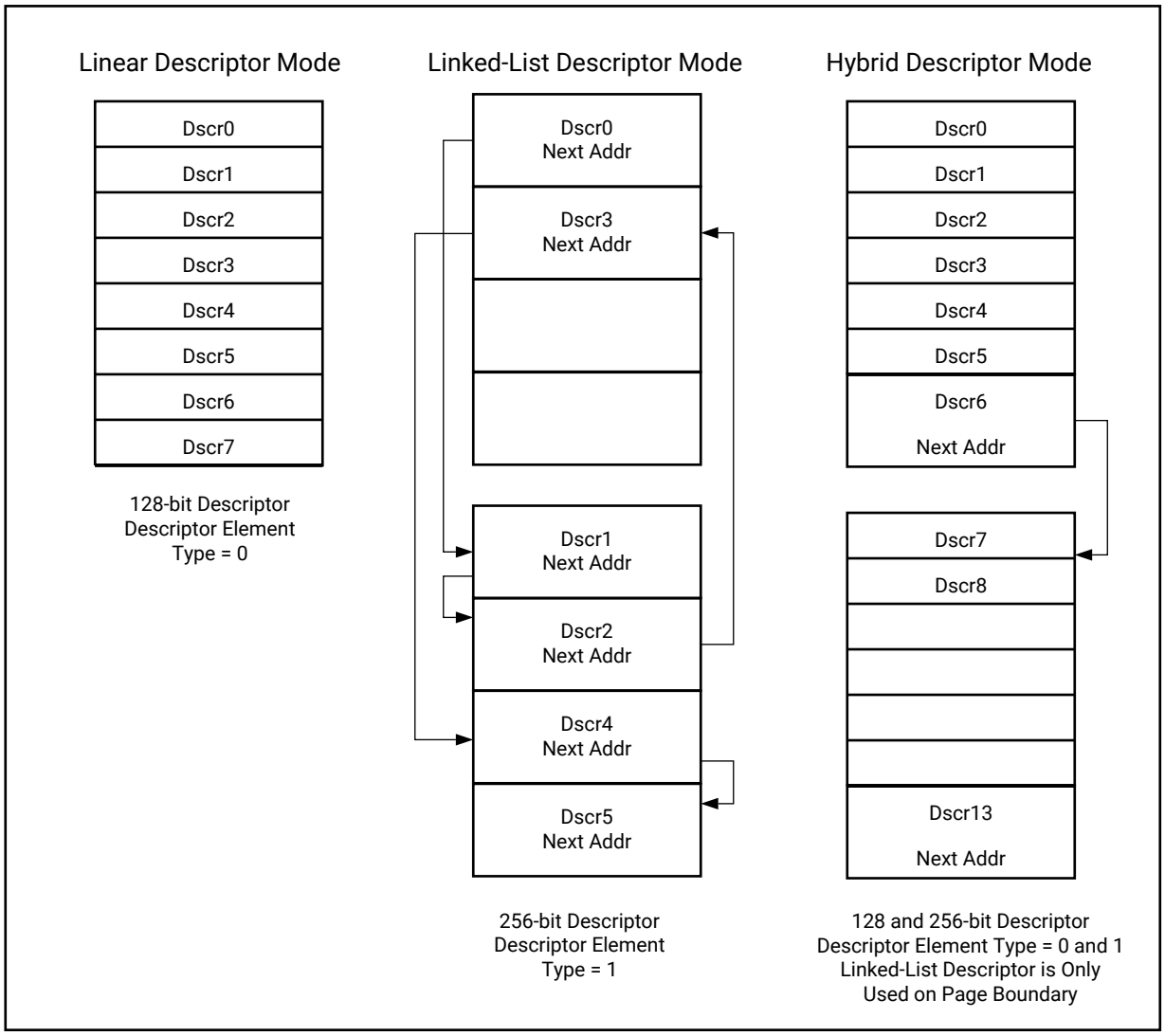
In scatter-gather DMA mode, the channel reads the data from the address specified in the SRC descriptor and writes to a location specified by the DST descriptor. The DMA implements a hybrid descriptor to support descriptor storage in two formats.

- Linear
- Linked-list
- Hybrid (multiple linear buffer descriptor arrays chained as a linked list)

Software can make use of a hybrid descriptor to dynamically switch between linear and linked-list mode. The hybrid descriptor approach allows the DMA driver software to arrange descriptors in a contiguous array of BDs, a linked list of BDs, or a mixed mode in which contiguous arrays of BDs can be chained together to create a linked list of BD arrays. This approach allows the driver software to be designed in a manner in which BDs can be allocated at initialization or in real time (and chained to a preceding BD). In applications where contiguous sets of memory are easily available, the software driver might not be able to manage a link list for descriptor storage. In this case, the descriptor can be stored in a linear array.

To support previously described cases, the DMA implements a hybrid descriptor. Each descriptor on the SRC and DST side implements a bit descriptor-element type, which indicates the type of the current descriptor. This allows software to switch between a linear and a link-list scheme dynamically. The following figure shows supported descriptor modes.

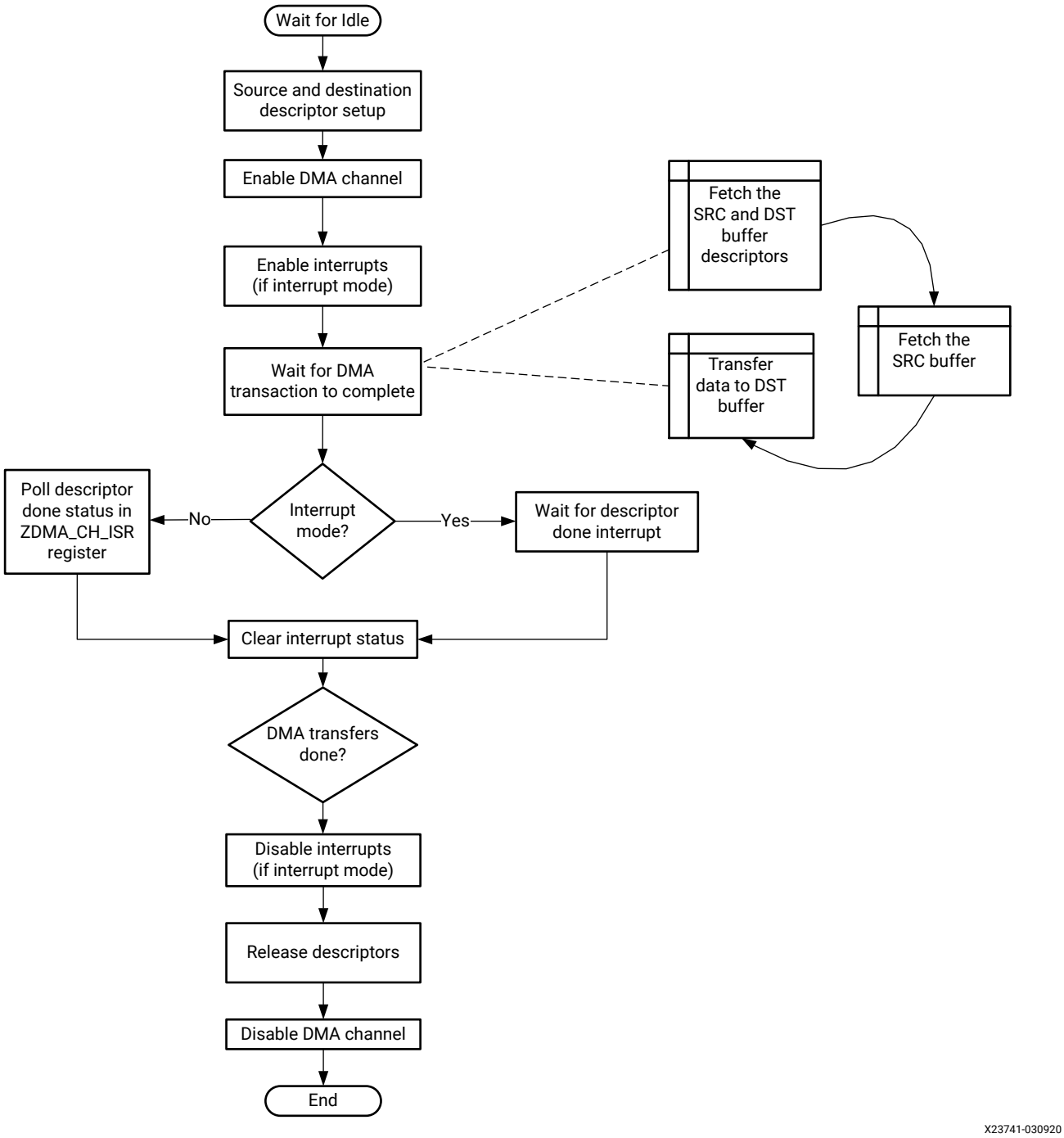
Figure 45: **DMA Supported Descriptor Mode Use-cases in Scatter-Gather Mode**



Linked List Mode Use Case

Linear mode is used when software can find a contiguous set of memory to accommodate all the buffer descriptors necessary (source and destination) as an array. The flowchart in the following figure captures the main steps.

Figure 46: **Linked List Flowchart**



Step 1

Ensure that the DMA is not in a busy state by reading the state field of ZDMA_CH_STATUS and ensuring that it is not `10`. If DMA is in the pause state, follow the steps to bring it out of the pause state as described in Channel Paused.

Step 2

1. Ensure the POINT_TYPE bit in ZDMA_CH_CTRL0 is set to `1`.
2. Allocate the source buffer descriptor objects in memory. Ensure that the buffer descriptor object address is 256-bit aligned.

   Note: If the buffer descriptors use the CCI-500 for hardware coherency, set AXCOHRNT to `1` in the ZDMA_CH_DSCR_ATTR register. Set the AXCACHE field to indicate a cacheable transaction with a value such as `1111b`.

   The address of the first buffer descriptor in a list is written to ZDMA_CH_SRC_START_LSB and ZDMA_CH_SRC_START_MSB.
3. Allocate the destination buffer descriptor objects in memory. Ensure that the buffer descriptor object address is 256-bit aligned.

   Note: If the buffer descriptors use the CCI-500 for hardware coherency, set AXCOHRNT to `1` in the ZDMA_CH_DSCR_ATTR register. Set the AXCACHE field to indicate a cacheable transaction with a value such as `1111b`.

   The address of the first buffer descriptor in a list is written to ZDMA_CH_DST_START_LSB and ZDMA_CH_DST_START_MSB.

TIP: *The buffer descriptors can also be pre-allocated during initialization time.*

Step 3

For each allocated source buffer descriptor object, program the following.

1. Program the source data fragment to transfer into the source buffer descriptor object. The ADDR LSB and ADDR MSB fields are programmed.
2. Program the size of each source data fragment to transfer into the source buffer descriptor object. The size field is programmed.
3. Set the coherency bit if the source data buffer uses the CCI-500 for hardware coherency. Set the ARCACHE field in the ZDMA_CH_DATA_ATTR register to indicate a cacheable transaction with a value such as `1111b`.
4. Set the DSCR element type to `0`.
5. Set the INTR field if an interrupt is required after the data is read for transfer. Typically, this can be set for the buffer descriptor object corresponding to the last source data fragment. Setting the last source descriptor for interrupt reduces the number of interrupts received.
6. The non-final buffer descriptor command field can be set to `00` for the next descriptor valid. For the final buffer descriptor, set the command field to `10` for STOP after completing this descriptor.

TIP: *If desired, set `01` to pause after completing the descriptor to put the DMA in a paused state after completing the final buffer descriptor. The steps to bring a channel out of pause into a enabled/disabled state are described in Channel Paused.*

7. Program the NEXT ADDR LSB and NEXT ADDR MSB to point to the next source buffer descriptor. If this is the last buffer descriptor in a linked list, these fields must be null.

Step 4

For each allocated destination buffer descriptor object, program the following.

1. Program the destination data fragment to transfer into the destination buffer descriptor object. The ADDR LSB and ADDR MSB fields are programmed.
2. Program the size of each destination data fragment to transfer into each respective destination buffer descriptor. The size field is programmed.
3. Set the coherency bit if the destination data buffer uses the CCI-500 for hardware coherency. Set the AWCACHE field in the ZDMA_CH_DATA_ATTR register to indicate a cacheable transaction with a value such as `1111b`.
4. Set the DSCR element type to `0`.
5. Setting the last source descriptor for interrupt reduces the number of interrupts received. Set the INTR field if an interrupt is required after the data is read for a transfer. Typically, this is set for the buffer descriptor corresponding to the last source data fragment. Setting the last destination descriptor for interrupt reduces the number of interrupts received.
6. The non-final buffer descriptor command field can be set to `00` for the *next descriptor valid*. For the final buffer descriptor, set the command field to `10` for *STOP after completing this descriptor*.

TIP: *If desired, set `01` to pause after completing the descriptor to put the DMA in a paused state after completing the final buffer descriptor. The steps to bring a channel out of pause into a enabled/disabled state are described in Channel Paused.*

7. Program the NEXT ADDR LSB and NEXT ADDR MSB to point to the next destination buffer descriptor. If this is the last buffer descriptor in a linked list, these fields must be null.

Step 5

Enable the DMA channel by writing into the control register ZDMA_CH_CTRL2. This initiates the DMA data transfer.

Step 6

Upon transfer completion, the DMA channel provides interrupts to the processor depending on how the INTR field of the buffer descriptors are set. For information on handling interrupts, see Interrupt Handling.

Software can use ZDMA_CH_IRQ_DST_ACCT and ZDMA_CH_IRQ_SRC_ACCT to decipher the number of processed buffer descriptors on the source and destination sides. Software can internally maintain counters of both the number of source and destination buffer descriptors configured for the data transfer. Upon updating the ZDMA_CH_IRQ_DST_ACCT and ZDMA_CH_IRQ_SRC_ACCT with an equal count, software can infer that the data transfer is complete. Software should count only those descriptors for which interrupts are enabled.

Step 7

After the DMA transfers are done, disable the DMA channel. See Channel Disabled for more information.

Linear Descriptor Use Case

In the linear descriptor use case mode, BDs are stored in a linear array. In Figure 45: DMA Supported Descriptor Mode Use-cases in Scatter-Gather Mode, the first block shows the linear descriptor mode. This can be considered as one 4K page. Each descriptor is 128 bits and the DMA channel can fetch 256 bits on every descriptor read. This allows the DMA to fetch two descriptors in a single AXI read and reduces the number of descriptor fetches.

- Each descriptor is 128 bits wide
- Each descriptor must be 128-bit aligned
- The descriptor element type is always `0` (in linear descriptor mode).

ADDR LSB [31:0]		WORD0
RSVD [31:12]	ADDR MSB [11:0]	WORD1
RSVD [31:29]	SIZE [29:0]	WORD2
RSVD [31:5]	CNTL [4:0]	WORD3

Linked-List Descriptor Use Case

Each descriptor is 256 bits wide, the first 128 bits store the descriptor information and the next 128 bits provide a pointer to the next descriptor. In this mode, the descriptor can be located anywhere in the memory (it might not be in the same 4K page).

- Each descriptor is 256 bits wide.
- Each descriptor must be 256-bit aligned.
- The descriptor element type is always 1 (in link-list descriptor mode).
- DMA channel can only fetch the next descriptor if it has read a current descriptor. Two descriptor fetches require two AXI reads.

ADDR LSB [31:0]		WORD0
RSVD [31:12]	ADDR MSB [11:0]	WORD1

RSVD [31:29]	SIZE [29:0]		WORD2
RSVD [31:5]		CTRL [4:0]	WORD3
NEXT DSCR ADDR LSB [31:0]			WORD4
RSVD [31:12]	NEXT DSCR ADDR MSB [11:0]		WORD5
RSVD [31:0]			WORD6
RSVD [31:0]			WORD7

Hybrid Descriptor Use Case

Linear and link-list descriptor types can be chained to reduce software and hardware overhead. For example, if software allocates two noncontiguous 4 KB pages to store descriptors, then it can contiguously store BDs in the first page and make the last BD of the first page point to the first BD of the next available page. Because the next address pointer in linear descriptor is not required, this scheme reduces both memory usage and software overhead. The descriptor mode diagram (Figure 45: DMA Supported Descriptor Mode Use-cases in Scatter-Gather Mode) details this use case.

- Each descriptor is aligned to its natural size.
  - Linear descriptor is 128-bit aligned.
  - Link-list descriptor is 256-bit aligned.

Buffer Descriptor Summary

- Both the SRC and DST descriptors must be aligned to their size.
- For efficiency, a DMA can prefetch a descriptor.
- The circular descriptor should always have at least one link-list element.
- Descriptors are not updated back to the memory. For instance, after a SRC/DST buffer descriptor is used by the DMA for data transfer, no updating of any field of SRC or DST buffer descriptor occurs to signal completion of a buffer descriptor to the software.
- A completion interrupt, along with status (interrupt accounting), is supported. The software can read the content of ZDMA_CH_IRQ_SRC_ACCT/ZDMA_CH_IRQ_DST_ACCT to find the number of buffer descriptors processed.

Interrupt Handling

Follow these steps to perform interrupt handling.

1. Read the status from the LPD_DMA.CH_ISR register.
2. If the [DMA_DONE] bit is set, mark the channel state as idle in the software context.

3. Check if the [DMA_PAUSE] bit is set. If yes, set the channel state to paused in the software context.
4. If any other error bit is set, set the channel as idle in the software context.
5. Clear the interrupt status from the LPD_DMA.CH_ISR register by writing back the value read in step 1.

Done Interrupt Accounting

***Note*:** The DMA channel does not update descriptors in memory.

When the controller is finished processing a descriptor table, it generates a SRC/DST done interrupt and updates the interrupt count register. Each channel includes the following scheme on both the SRC and DST sides.

- The software can selectively request a completion interrupt on descriptors. After a descriptor is processed, the DMA increments the interrupt count register.
- A SRC descriptor done interrupt is generated after the DMA is done reading all the data corresponding to the source buffer descriptor. The SRC descriptor done interrupt does not guarantee that data is written at a destination location. Data can still be in a shared common buffer.
- A DST descriptor done interrupt is generated after the DMA channel receives a response to the last AXI write of the buffer corresponding to the DMA buffer descriptor. The DST done interrupt ensures that data has been written to the memory location.

An interrupt is generated to the software as soon as the interrupt accounting's count transitions to non-zero. When the software takes this interrupt, it should also read the interrupt accountings register. Count provides the number of processed descriptors with interrupt enabled. This counter is cleared on read (due to coherency). This scheme eliminates the need for a timeout mechanism. It also provides flexibility to the software to enable an interrupt on a required descriptor.

The DMA channel implements a separate 32-bit interrupt account counter for the source and destination sides. If the software does not read/clear the counter for a long time, this counter can overflow. The DMA generates an interrupt to indicate the overflow condition on the interrupt accounting counter. If a counter over flows on the last descriptor of a DMA transfer (DMA DONE), the interrupt accounting counter overflow interrupt is generated.

Over Fetch

The DMA supports an AXI bus width of 128/64 bits. In the case where the source descriptor payload ends at a non-128/64 bit aligned boundary, the DMA channel fetches the last beat as the full-128/64 bit wide bus. This is considered an over fetch. The over fetch option can be disabled. If an over fetch is disabled and the SRC descriptor payload ends on a non-128/64 bit boundary, the DMA fetches any remaining bytes as a single byte AXI read.

The example in the following figure uses a source descriptor size of 8190 bytes (with a start address at `0x0000_0000` and end address at `0x0000_1FFD`), a 128-bit wide AXI bus, and a burst length of 16, the DMA can fetch 256 bytes in a single AXI burst. Two scenarios are provided in this section.

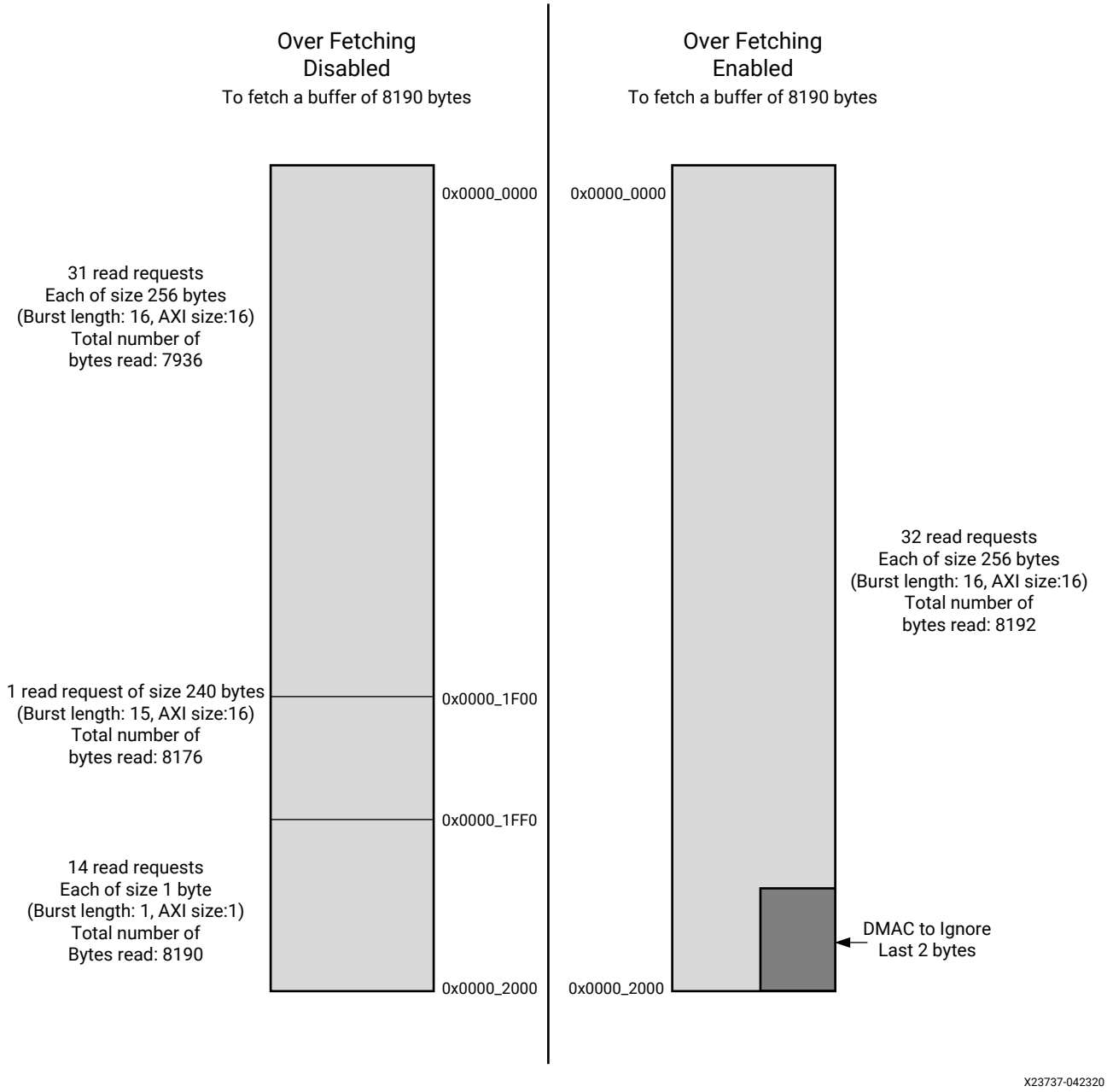
Scenario 1: Over Fetch is Disabled

- 31 AXI read command with burst length of 16 and AXI size of 16 bytes (7936 bytes fetched).
- One AXI read command with burst length of 15 and AXI size of 16 bytes (240 bytes fetched).
- To fetch the remaining 14 bytes, the DMA channel issues 14 single-beat AXI read commands with an AXI size of 1 byte.

Scenario 2: Over Fetch is Enabled

- 32 AXI burst length of 16 and AXI size of 16 bytes (8192 bytes fetched)

Figure 47: **Over Fetch Scenarios**



RECOMMENDED: *If the over fetch is disabled, it could significantly impact the performance of the DMA channel. Xilinx recommends only disabling the over fetch when absolutely necessary.*

Transaction Control

The transaction control mechanism is used to control the rate and number of read/write data transactions from a channel. The control parameters are applicable only to data transactions and not for descriptor read transactions.

TIP: *If the multiple rate control mechanism is enabled on a channel, a transaction is issued to arbitration when all enabled rate control mechanisms provide permission to issue that transaction.*

Data transactions on AXI read channels can be controlled per each channel using the following control mechanisms.

Outstanding Transactions

Each DMA channel provides a control register ZDMA_CH_CTRL1[SRC_ISSUE] where the software can program a maximum number of read outstanding transactions. The DMA channel uses this parameter to limit the number of outstanding read data transactions.

Rate Control

Each DMA channel can be independently programmed to issue transactions on a periodic basis. Higher priority channels can have a shorter interval between transactions. The lower priority channels can have a longer interval between transactions. The issue rate is independently controlled for each channel using an interval count that is programmed into the ZDMA_CH_RATE_CTRL [CNT] bit field. Rate control is enabled by setting ZDMA_CH_CTRL0 [RATE_CTRL] = 1. There are 16 pairs of registers for rate control (8 channels).

Enabling rate control causes the DMA channel to copy the interval count, ZDMA_CH_RATE_CTRL [CNT] bit field, into the channel's decrementing counter. This counter is decremented with every clock cycle. When the counter reaches 0, the DMA channel issues a transaction to the arbiter and again copies the interval count into the decrementing counter. The channel waits for the counter to reach 0 again, and then issues another transaction and reloads the counter. The cycle continues until disabled by setting [RATE_CTRL] = 0.

TIP: *When rate control is enabled, the read data transaction frequency is always equal to or less than the programmed rate control frequency (1/rate control count).*

Flow-Control Interface

Data transactions on the AXI write channel can only be controlled using the flow-control interface (FCI). The FCI is implemented per channel to provide read/write access flow control ability to the PL slave. The FCI can be independently controlled from each channel's control register. The software configures what accesses are flow controlled by the read/write FCI.

The PL slave provides credits to the DMA channel. Each credit is a permission for a single AXI transaction. When the FCI is attached to the SRC (read), there is a permission to generate one AXI data read transaction (write transaction when FCI is attached to write DST). The following table lists the FCI signals.

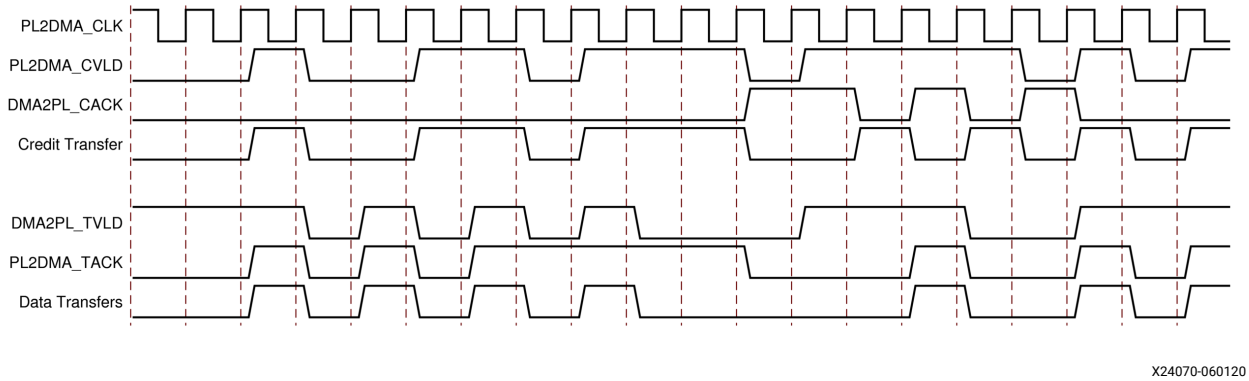
IMPORTANT! *The maximum number of credits accepted are 32.*

Table 64: **Flow Control Interface Signals**

Signal	Description
PL2DMA_CLK	PL clock: signals from/to PL are synchronous to PL2DMA_CLK. The DMA handles all clock domain crossing.
PL2DMA_CVLD	Credit valid signal to DMA.
DMA2PL_CACK	Credit acknowledgment from DMA: • Credits are accumulated when: ◦ PL2DMA_CVLD is High, and ◦ DMA2PL_CACK is Low • Each FCI can accumulate up to 32 credits. • If the FCI is not enabled, then the credits are cleared.
DMA2PL_TVLD	Transaction valid.
PL2DMA_TACK	Transaction acknowledgment: the DMA channel indicates that one write transaction is done (AXI write command was generated and a BRESP is received) when DMA2PL_TVLD and PL2DMA_TACK are True.

The timing diagram for the flow control interface is as shown in the following figure.

Figure 48: **Flow Control Interface**



Software can configure FCI to flow control either the SRC or DST based on whether the DMA channel is reading from or writing to the PL slave.

- FCI must be configured to flow control SRC if the DMA is reading from the PL slave
- FCI must be configured to flow control DST if the DMA is writing to the PL slave

Flow-Control Interface Considerations

- After reset, the FCI is configured to flow control the SRC read side
- ARLEN is used for all AXI transactions; both the SRC and DST sides
- Software configures the FCI to the correct side (SRC/DST)
- In case of an error, the DMA channel waits until the transaction valid FIFO is empty before going to DONE with an error state
- The DMA channel stops issuing write commands if the PL does not provide a PL2DMA_TACK in response to a DMA2_PL.TVLD for an extended time and the transaction FIFO goes full

When the FCI is attached to the DST side, the SRC transactions are limited by the threshold allowed in the common buffer. This threshold can be programmed by the ZDMA_CH_FCI [PROG_CELL_CNT] bit field in that channel. The DMA channel stops issuing data read commands after the number of occupied cells exceeds the programmed cell count threshold. If the write side of the channel is using FCI and the read side is not controlled, then the channel uses most of the common buffer. This limits the other channels. By using the threshold on common buffer usage, the channel's usage of the common buffer can be controlled.

After the channel is enabled with the FCI, the DMA channel accumulates incoming credits. Each channel can accumulate up to 32 credits. Each transaction consumes one credit. The channel does not issue a new transaction if a credit is not available. The credit is consumed upon generation of read/write commands based on the FCI configuration. If the FCI is not enabled, it does not affect the generation of AXI commands on the SRC/DST.

The FCI accepts credit from the PL slave as long as the credit FIFO is not full. Credits are cleared until the channel is enabled. After a channel is enabled, a DMA channel uses credits to flow control the SRC/DST AXI commands. In the event of an error, the DMA channel performs an error-recovery sequence. After error recovery is done, the channel clears both the FCI_EN and channel EN flags. After it clears the FCI_EN, the DMA channel clears all available and incoming credits until the next peripheral enable. The software provides channel state information to the PL slave (enable, pause, and error).

The DMA channel provides a transaction valid notification to the PL slave on every AXI write transaction completion. A transaction valid is always generated on receiving a valid BRESP. Irrespective of any read/write association, a transaction valid always indicates completion of a write transaction. The software can calculate and provide the total number of valid transactions expected to complete the current DMA transaction to the PL slave. The PL slave can use a transaction valid to find where a DMA channel is in a current DMA transaction.

Flow-Control Programming Model

The DMA implements one FCI per channel. An FCI interface can be independently controlled per channel. After each DMA transaction is done, the DMA channel clears both the channel EN and FCI_EN flags. The software must enable the FCI interface for each DMA transaction. If the FCI interface is not enabled (FCI_EN = 0), the DMA channel flushes all incoming credits.

Credits are only valid when the FCI interface is enabled (FCI_EN = 1).

- Setup channel mode (simple and scatter-gather mode).
- LPD_DMA.CH_DSCR_ATTR and LPD_DMA.CH_DATA_ATTR registers.
- Setup DMA mode:
  - Simple mode, program the DSCR registers.
  - SG mode, program the DSCR in memory and program the DSCR start address register.
- Set the FCI control parameters, LPD_DMA.CH_FCI [EN, SIDE].
- Set the enable bit, CH2_CTRL [EN]. This provides a trigger to the DMA channel.

The DMA channel provides transaction acknowledgment for all valid credits received after the LPD_DMA.CH_FCI [EN] bit is set. The DMA channel clears LPD_DMA.CH_FCI [EN] after it is done with the DMA transaction. The software must enable FCI along with the channel enable for subsequent DMA transfers.

The suggested use-model for applications is:

- SRC and DST payload addresses are aligned to programmed AXI burst length and an over fetch is enabled.
- Software provides the transfer size details to the flow control slave.

Implementation Notes

- If the suggested use-model requirements are satisfied, attaching FCI to SRC/DST is not required.
- When FCI is enabled, both the AXI read and write command use the same burst length SRC AXI length (ARLEN).
- When the SRC and DST descriptor payloads are not aligned to the bus width, the number of read and write transactions could be different.
- The size of the first and last transaction can be different based on the alignment of the read and write payload.
- One credit means one AXI read or write transaction. The size of the transaction can vary based on the 4k boundary crossing and over fetch disable. The DMA channel never generates a transaction larger than the programmed ARLEN.
- Read/write transactions can be controlled using more than one mechanism. A channel might not generate a transaction, even if it has credits, due to other channel control parameters.
  - Rate control counter.
  - Outstanding transaction count.

Attached to the SRC

Software can enable the FCI before enabling a channel. The DMA channel uses ARLEN on both the SRC and DST sides.

Number of SRC transaction = Number of DST transaction
SRC AXI transaction size/length = DST AXI transaction size/length

If a DMA channel is reading data from the flow controlling slave, each credit given to the DMA channel reads ARLEN x bus width (in bytes) worth of data. ARLEN x bus width (in bytes) worth of data is written to the FCI slave if the DMA channel is writing data to a slave.

The DMA channel can accept up to 32 credits. The slave can use this to pipeline credits to the DMA channel. Because of the aligned address requirement, each credit is the transfer size of ARLEN x bus width (bytes). A slave uses this to keep track of the number of bytes transferred. This information is used by slave to issue credits.

Channel Reading from a Flow Controlling the PL Slave

A DMA channel reading from a flow controlling the PL slave scenario is similar to the suggested use model except the one-to-one correlation between SRC and DST AXI commands does not exist. The number of commands generated on the SRC side can be different than the DST. In this case, the number of transaction valid responses can be less/more than the number of credits used. Unless the software calculates the number of valid transactions required for DMA transfer, the PL slave cannot use the valid transactions.

The DMA channel only generates read data transactions if credit is available. After it has enough data to generate a write transaction, it issues a write command. The slave can snoop on the AXI read channel to keep track of the number of beats/bytes read by the DMA channel.

Channel Writing to a Flow Controlling the PL Slave

In a DMA channel writing to a flow controlling the PL slave scenario, the software configures the FCI to flow control the DST. Each valid credit allows the DMA channel to perform one AXI write command. If the read/SRC is not flow controlled when the FCI is configured to the flow control DST, the channel can issue multiple read transactions and use the entire common buffer, which starves other channels. To resolve this issue, software configures the maximum number of entries used by the DMA channel. After the DMA channel exceeds the programmed value, it does not issue more read transactions. The PROG_CELL_CNT of the ZDMA_CH_FCI register can be programmed in the register.

Maximum number of occupied cells = (ARLEN + 1) << PROG_CELL_CNT

If the software programs PROG_CELL_CNT to zero, the maximum number of entries occupied by the DMA channel is the same as one full AXI burst.

Because the SRC and DST addresses are unaligned and over fetch can be disabled, the DMA channel might have to generate multiple read transactions to perform a single write transaction. Because of this, it is advised to program PROG_CELL_CNT to a `1`. As explained previously, the number of SRC and DST transfers can be different and unless the software calculates the number of valid transactions required for DMA transfer, the PL slave cannot use the valid transactions.

The DMA channel generates write data transactions only if credits are available. The write command is only generated when enough credit and enough data is available to generate one write transaction.

Error Conditions

DMA errors are isolated per channel and an error on one channel should not affect any other channel. There are multiple sources producing errors during DMA operation.

Software Programming Error

The DMA assumes that software programs registers and descriptors as expected. In case of incorrect software programming, a DMA channel does not take any action for error recovery and DMA channel behavior is unpredictable.

DMA Implements Interrupt Accounting Support

The software can selectively enable interrupt generation on each descriptor (independent on SRC and DST). On every descriptor done (which asked for an interrupt), the DMA increments a descriptor done counter. Each DMA channel implements an 8-bit interrupt accounting counter on the SRC and DST sides. An interrupt accounting counter overflow is indicated as an interrupt. Independent SRC and DST interrupts are generated. This is non-fatal error as it does not affect the channel functionality.

AXI Errors

In case of an AXI decode/slave error on data read/write or descriptor read, the DMA channel performs an error recovery sequence and recovers all occupied entries in the common buffer. After completing the error recovery sequence, it generates an interrupt to indicate the type of error and disables the channel.

AXI Errors

The DMA allows software to mark each channel secure/non-secure by programming the LPD_SLCR_SECURE.slcr_adma register. The secure bit field [tz] includes 8 bits to set the TrustZone security setting for the 8 DMA channels. If a channel is marked secure, only a secure master can access its DMA control and status registers. The DMA tags all the AXI transactions secure if a channel is marked secure.

Secure DMA channel characteristics include the following:

- Only secure masters can access their control and status registers.
- All AXI transactions from this channel are marked secure. They can access both secure and non-secure regions.

Non-secure DMA channel characteristics include the following:

- Both secure and non-secure masters can access their control and status registers.
- All AXI transactions from this channel are marked non-secure, and can access only non-secure regions.

Error Event

Space in the common buffer is allocated as each channel requests, one 128-bit entry at a time (from a total of 256 entries). A channel might hold many entries, and all of them must be cleared when an error is detected.

Security

The DMA allows software to mark each channel secure/non-secure by programming the LPD_SLCR_SECURE.slcr_adma register. The secure bit field [tz] includes 8 bits to set the TrustZone security setting for the 8 DMA channels. If a channel is marked secure, only a secure master can access its DMA control and status registers. The DMA tags all the AXI transactions secure if a channel is marked secure.

Secure DMA channel characteristics include the following:

- Only secure masters can access their control and status registers.
- All AXI transactions from this channel are marked secure. They can access both secure and non-secure regions.

Non-secure DMA channel characteristics include the following:

- Both secure and non-secure masters can access their control and status registers.
- All AXI transactions from this channel are marked non-secure, and can access only non-secure regions.

Channel Paused

The software can pause any channel by setting the scatter-gather descriptor command bits to pause. This feature is used to pause the DMA operation and program the next set of descriptors.

Current DSCR indicates CMD = Pause

If the current descriptor command bits indicates a pause, the DMA channel completes the current descriptor payload to the DST locations. After it is done with data transfer, the DMA channel goes into pause mode. The channel keeps the current operational state.

Coming Out of Pause

There are two ways to bring a channel out of pause and into active mode:

- Keep the current state and read the next descriptor continuously from the last descriptor before going into pause.

  CONT bit is set in the control register and [CONT_ADDR] = 0.
- Use the DSCR start address to fetch the first descriptor coming out of pause.

  CONT bit is set in the control register and [CONT_ADDR] = 1.

Software can also put the DMA channel in disable mode from pause mode: Mode = Pause, enable = 0, and [CONT] = 1.

Programming Model for Changing DMA Channel States

A DMA channel can be in one of the following states at any time. This section explains each state.

- Disabled
- Enabled
- Paused

Channel Enabled

The software can enable one or more channels at any time using the following enable sequence.

1. Setup channel mode (simple or scatter-gather mode).
2. Set the ZDMA_CH_{DATA, DSCR}_ATTR attribute registers.
3. Setup DMA mode.
   a. Simple mode, program the DSCR registers.
   b. In scatter-gather mode, program the DSCR in memory and program the DSCR start address register.
4. Set enable bit in the ZDMA_CH_CTRL2 register. This provides a trigger to the DMA channel.

Channel Disabled

The channel can go into a disabled state for these reasons:

- Current SRC descriptor indicates CMD = STOP.
  - DMA processes the current descriptor and goes into a disable state.
  - DMA channel ensures that all the data is transferred to the DST memory location before going into a disable state and updating the status register.
  - This mechanism can be used to indicate the end of an operation.
- DMA channel is in simple DMA mode and transfer is done.
  - After a channel is done transferring the data indicated into the SRC/DST DSCR register, the channel goes into a disable state.

- For subsequent transfers, the software must enable the channel.

- Software can put any paused channel into a disable state.
  - The current channel state is pause and it has received a CONT from the APB register.

    Mode = Pause & enable = 0 and CONT = 1

    The DMA channel goes into disable mode
- Any error detected on an AXI channel/descriptor programming puts the DMA channel into a disable state.

Register Reference

DMA Channel Registers

The LPD DMA registers are listed in the following table. The base address for each channel's registers are:

- Channel 0, `0xFFA8_0000`
- Channel 1, `0xFFA9_0000`
- Channel 2, `0xFFAA_0000`
- Channel 3, `0xFFAB_0000`
- Channel 4, `0xFFAC_0000`
- Channel 5, `0xFFAD_0000`
- Channel 6, `0xFFAE_0000`
- Channel 7, `0xFFAF_0000`

Table 65: **LPD DMA Channel Register Set**

Register Name	Address Offset	Access Type	Description
ERR_CTRL	0x000	RW	APB address decode error
CH_ISR CH_IMR CH_IER, CH_IDR	0x100 0x104 0x108 0x10C	WTC, R W W	Interrupt status, mask, enable, and disable
CH_CTRL0 CH_CTRL1 CH_CTRL2	0x110 0x114 0x200	RW	Controls
CH_FCI	0x118	RW	Flow control interface
CH_STATUS	0x11C	R	State of channel

Table 65: **LPD DMA Channel Register Set** *(cont'd)*

Register Name	Address Offset	Access Type	Description
CH_DATA_ATTR CH_DSCR_ATTR	0x120 0x124	RW RW	Data and descriptor AXI parameters
CH_SRC_DSCR_WD0 CH_SRC_DSCR_WD1 CH_SRC_DSCR_WD2 CH_SRC_DSCR_WD3	0x128 0x12C 0x130 0x134	RW	Source descriptor words
CH_DST_DSCR_WD0 CH_DST_DSCR_WD1 CH_DST_DSCR_WD2 CH_DST_DSCR_WD3	0x138 0x13C 0x140 0x144	RW	Destination descriptor words
CH_WR_ONLY_WD0 CH_WR_ONLY_WD1 CH_WR_ONLY_WD2 CH_WR_ONLY_WD3	0x148 0x14C 0x150 0x154	RW	Write only data words
CH_SRC_START_L CH_SRC_START_H	0x158 0x15C	RW	Source descriptor start address
CH_DST_START_L CH_DST_START_H	0x160 0x164	RW	Destination descriptor start address
CH_RATE_CTRL	0x18C	RW	Rate control count
CH_IRQ_SRC_CNT CH_IRQ_DST_CNT	0x190 0x194	RW RW	Source and destination interrupt account count

I/O Flow Control Signals

The controller has a set of flow control signals attached to the PL. These are optionally used to manage the flow of data between the controller and a slave memory in the PL.

Each channel has flow control signals routed to the PL.

- PL2DMA
- DMA2PL

XILINX

Section VII

Platform Processor, Configuration, and Security Units

This section includes chapters that describe the functionality of a core set of PMC blocks. These are listed below. This section does not include the flash controller chapters, JTAG, or system-level topics. Those chapters are located in other sections of the TRM.

All of the chapters related to platform management are organized into groups in the Overview chapter.

The remaining chapters are assigned to platform management-specific functional units and modules:

- Platform Processing Unit (PPU)
- Processing System Manager (PSM)
- PL Configuration Units
  - Configuration Frame Unit (CFU)
  - Configuration Frame Interface (CFI)
- Slave Boot Interface (SBI)
- Streaming Interconnect Module
  - Secure Stream Switch
  - PMC DMAs
  - AES-GCM
  - SHA3-384
- RSA/ECDSA
- True Random Number Generator (TRNG)
- Physically Unclonable Function
- Battery-Backed RAM (BBRAM)
- eFUSE Controller
- PMC Register Reference

Overview

The platform management hardware includes processors, configuration units, and security units to support hardware and software boot operations. The PMC hardware architecture is illustrated in PMC Interconnect Diagram.

The PMC includes the ROM code unit (RCU) processor running the BootROM code for the hardware boot process. The PMC also includes the platform processing unit (PPU) running the platform loader and manager (PLM) software. The platform management operations are described in the Platform Management Controller chapter.

The many chapters for the blocks that are used for platform management are divided into the following groups. Many of these links go to other sections of the TRM.

Boot Interface Hardware

The boot modes are summarized in Boot Modes. The controllers for these boot modes are described in these TRM sections:

- Section XIII: Flash Memory Controllers
- JTAG Boot Mode
- Slave Boot Interface

Platform Management Processors

There are three platform management processor units:

- ROM code unit (RCU runs BootROM code)
- Platform Processing Unit (PPU runs PLM software, initial download by BootROM)
- Processing System Manager (PSM runs firmware downloaded by PLM)

Inter-processor Interrupts

The inter-processor interrupts (IPI) enable the PPU, PSM, and other processors in the PS to signal and send short private messages between each IPI agent.

- Inter-Processor Interrupts for processor-to-processor communications

Global Register Request Interrupts

The requests for platform management (from system software) are directed to the PLM software in the PMC or to the PSM firmware in the LPD.

- PMC Global Register Set
- PSM Global Register Set

System Error Signals

System errors are reported via the error status accumulator registers:

- PMC Error Status Accumulator Registers
- PSM Error Status Accumulator Registers

Streaming Module

The streaming switch interfaces to the encrypted bit streams and security units are generally only accessed by the PLM to manage authentication and encryption/decryption:

- AES-GCM
- SHA3-384
- PMC DMAs, DMA0 and DMA1
- Secure Stream Switch (SSS)

Additional PMC Units

Additional units include:

- RSA/ECDSA
- True Random Number Generator (TRNG)
- Battery-Backed RAM (BBRAM)
- eFUSE Controller

PMC Programming Interfaces

There are three programming/configuration interfaces.

- APB programming interface (memory mapped registers for PMC and PS)
- NPI programming interface (memory mapped registers for NoC, DDRMC, AI Engine and others)
- CFU slave interface consumes PL configuration files (configuration frames)

These are described in Programming Interfaces chapter.

XILINX

Chapter 32

Platform Processing Unit

The platform processing unit (PPU) normally runs the platform loader and manager (PLM) firmware. The PLM configures the system and downloads boot image files. The PLM monitors the system and provides platform services. The resources available to the PPU firmware include security, power control, error detection, and functional safety features.

The PPU is a master on the PMC main switch and implements the MicroBlaze™ architecture.

Features

The PPU includes the following features:

- 32-bit MicroBlaze processor with triple modular redundancy (TMR)
- PPU RAM with 384 KB storage
  - Local PPU memory for instructions and data
  - ECC protected

The PPU can also access the PMC system RAM (128 KB) attached to the PMC main switch.

System Perspective

Interrupts

Each PPU generates two system interrupts:

- Memory, FPU, access errors (IRQ 43, 44)
- Performance monitor (IRQ 40, 41)

The PPU receives the system interrupts via its GIC proxy registers.

APB Address Decode Error

The PPU generates a system interrupt when the APB programming interface detects an address decode error.

GIC Proxy Interrupt Controller

The GIC proxy clients PPU and PSM receive all system interrupts. The masking is controlled by the global registers.

Inter-processor Interrupts

The PPU processor is a hardwired IPI agent. This enables the PLM firmware to receive and establish communication channels between itself and other IPI processor agents. The IPI is described in Inter-Processor Interrupts.

GIC Proxy System Interrupts

The PPU receives all of the system interrupts into its register module.

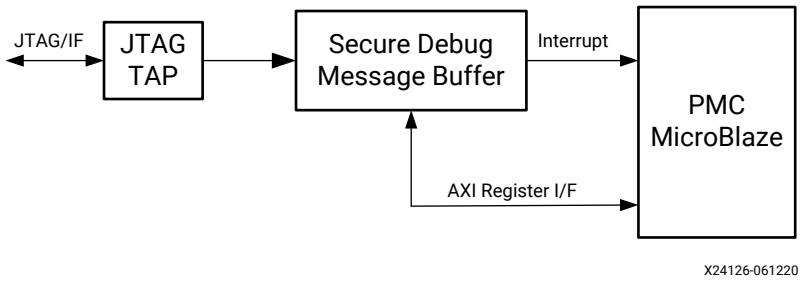
Tamper Event Monitoring and Response System

The Versal device has a robust tamper monitoring and response system. This system allows for the detection of, with programmable responses to, voltage glitches, voltage and temperature deviations, and activity on external debug interfaces (e.g., JTAG). Supporting extensibility to a larger tamper system, the Versal device allows for the enablement of an external tamper trigger with a programmable response.

Secure Debug

The Versal device allows for debug enablement of a secure system via an external interface (i.e., JTAG). JTAG is blocked by default in a secure system. If enabled by the user, the securely disabled JTAG port will listen for a cryptographically secure message (RSA or ECDSA signed). If such a message is received, the JTAG port is enabled by the PMC. Improperly authenticated messages can be either ignored or used as a trigger to the Versal device tamper response system. The secure debug feature is disabled by default and can be permanently disabled using eFUSEs. A high-level overview of this method is shown in the following figure.

Figure 49: **Secure Enablement of JTAG Interface**



Programming Model

The PMC processing unit (PPU) is based on a MicroBlaze triple module redundant processor. The platform loader and management (PLM) software is downloaded into the PPU RAM memory during the BootROM process. The PLM controls the system during normal and abnormal operations. The PLM software programming uses the PPU processor programming model.

The programming model includes:

- MicroBlaze processor
- Global control and status registers
- PPU and PMC RAM memories
- Memory-mapped access to NoC, PL, and LPD resources
- Memory-mapped programming interfaces: APB and NPI
- System-level interrupts, errors, and other signals

System Masters

There are two large interconnect switches in the PMC, see the PMC Interconnect Diagram figure.

- PMC main switch
- PMC IOP switch

The PMC slaves are listed in PMC Address Map.

XILINX

Chapter 33

Processing System Manager

The processing system manager (PSM) is a processor in the LPD that controls the power islands in the PS. The PLM downloads firmware into the PSM RAM for the processor to execute.

The PSM includes global registers that are written by system software to request platform services. This functionality is explained in PSM Global Register Set.

The PSM is a triple module redundant (TMR) MicroBlaze™ processor.

Features

The PSM includes the following features:

- 32-bit MicroBlaze processor with triple modular redundancy
- Local RAM for instruction and data
- GIC proxy system interrupt controller
- Register controls for PS power islands

System Perspective

The PSM is located in the LPD. It has a 32-bit slave interface accessible from the LPD slave switch and a 32-bit master interface with access to the LPD main switch.

Reset

The PSM can be reset in several ways, including:

- POR reset
- PS reset
- LPD reset

Processor State After Reset

After a reset, the PSM can come up executing or in a sleep state. The control registers are shown in the table.

Table 66: **PSM Reset Mode and Wake-up Register Control**

Register Name	Bit Field	Address	Access Type	Description
CRL.PSM_RST_MODE	[rst_mode]	0xFF5E_0370	RW	PSM processor after reset: 00: Execution mode 01: Sleep mode
	[wakeup]			PSM processor wake-up: 0: sleep 1: execute

Interrupts

The PSM generates a core interrupt and a dedicated inter-processor interrupt (IPI).

- PSM core interrupt: IRQ 82
- PSM dedicated IPI: IRQ 61

The PSM also receives all the system interrupts via its GIC proxy registers.

APB Address Decode Error

The PPU generates a system interrupt when the APB programming interface detects an address decode error.

Inter-processor Interrupts

The PPU processor is a hardwired IPI agent. This enables the PLM firmware to receive and establish communication channels between itself and other IPI processor agents. The IPI is described in Inter-Processor Interrupts.

GIC Proxy Interrupt Controller

The GIC proxy clients PPU and PSM receive all system interrupts. The masking is controlled by the global registers.

XILINX

Chapter 34

PL Configuration Units

The PL is configured by sending programmable device image (PDI) partition information to the configuration frame unit (CFU). The CFU interprets the data received via the PMC main AXI switch and generates commands to the configuration frame interface (CFI). The CFI bus transactions are sent to the CFRAMEs scattered throughout the programmable logic (PL). This configures the DSPs, CLBs, block RAM, UltraRAM, distributed RAM, clocking structures, and more in the PL.

Configuration Frame Unit

The configuration frame unit (CFU) translates AXI traffic into the CFI bus protocol, performs data integrity checks, and manages the CFI traffic. The CFU also performs several functions including frame write rate matching, frame read rate matching, frame read/write transition, row switching, decompression, and putting the CFI into idle.

The CFU is the only master for the CFI bus. The input CFU reference clock runs at the same rate as the output CFI clock frequency. The CFU AXI interfaces support 32-bit, 64-bit, or 128-bit transactions. The CFU cannot support multiple AXI masters at the same time while streaming or while a configuration frame (CFRAME) read/write is in progress so it uses a round robin arbitration method to handle the AXI input for write and read.

The programmable device image (PDI) created with the Vivado® or Vitis™ tools contains PL partition data that the PMC CFU manages. The CFU handles the transfer of configuration data through the CFI to the CFRAMEs in the PL. The CFU ensures that the CFRAMEs are ready to accept the CFI bus transactions sent to them. The CFU checks the packet data to determine if the packet rate sent needs to be slower and throttles the AXI interface as necessary. The CFU also performs data decompression when PDI compression is used.

The CFU includes an APB programming interface to the CFU_CSR and CFU_CFRAME register modules. These are used to configure the functionality, poll status, and generate PL global signal sequences.

Configuration Frame Interface

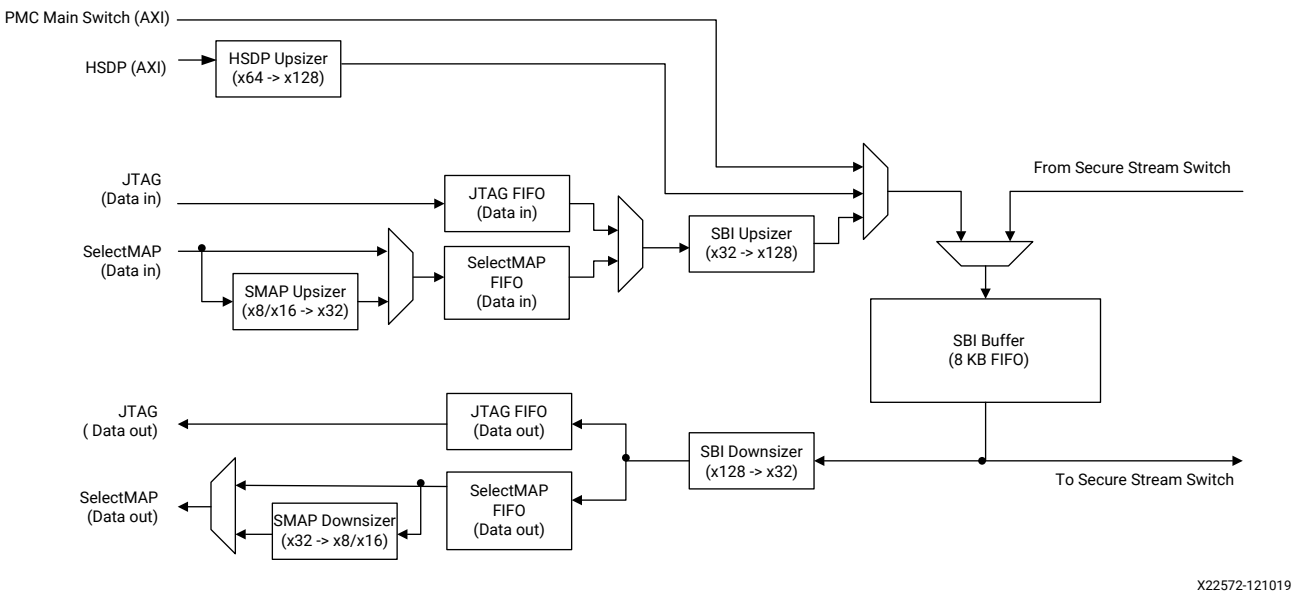
The configuration frame interface (CFI) is a dedicated 128-bit programming bus used to configure the PL. The CFU is the only master for the CFI bus. The CFI bus protocol is translated by the CFU from AXI. The CFI bus transactions are sent to all CFRAMEs. Each CFRAME has a logic controller that decodes the row address to determine whether or not it should receive the CFI packet.

The CFI is pipelined as it travels through the CFRAMEs. The CFU monitors the CFI bandwidth and pending operations to ensure that the CFRAME can accept the data being sent.

Slave Boot Interface

The slave boot interface (SBI) is a Xilinx proprietary interface that buffers incoming configuration data from the slave boot modes (JTAG and SelectMAP). The incoming data is then fetched by the RCU or PPU using one of the PMC DMAs for further processing and loading to the configuration interfaces. The SBI requires flow control to prevent overflow of the internal buffer using a BUSY pin. For SelectMAP, when BUSY is asserted, the external controller must stop sending data within 24 cycles. The SBI datapath and interface options are shown in the following figure. The HSDP, SelectMAP, and JTAG data are adjusted to the 128-bit bus for processing.

Figure 50: **SBI Datapath**



Streaming Interconnect Module

Secure Stream Switch

The secure stream switch (SSS) in the PMC routes the data between the two DMAs (PMC DMA0 and PMC DMA1), AES-GCM, SHA3-384, SBI, and the PMC global signals. The data sources for this switch are AES-GCM, SBI, and the two PMC DMAs. The data sources can only broadcast to a set of specific destinations as shown in the following table.

Table 67: **Secure Stream Switch**

Data Source	Data Destination (Valid Broadcast Option)				
	PMC DMA0	PMC DMA1	AES-GCM	SHA3-384	SBI
PMC DMA0	X		X	X	X
PMC DMA1		X	X	X	X
AES-GCM	X	X			
SBI	X	X			

Any invalid configuration takes on the default configuration that essentially ties all destination ports to `0x0`. The switch generates an error interrupt if data is broadcast into the system when the switch is in default configuration or when data is received from an unintended interfacing module during a particular configuration.

The configuration is set in the register, PMC_SSS_CFG that resides in the PMC_GLOBAL register space. A particular destination module listens to only one of the source modules in the system at a particular instant in time.

PMC DMAs

The PMC has two 128-bit DMAs. The PMC DMA0 and PMC DMA1 are 2-channel simple DMAs that allow separate control of the read and write channel.

The PMC DMAs primary responsibility is to move data efficiently between the memory-mapped 128-bit AXI interface and the PMC secure stream switch domain. The PMC DMAs move data to and from the cryptographic accelerators (AES, SHA) and SelectMAP through the secure stream switch. The PMC DMAs are not bound to the PMC address space. For example, they can be used to fetch a reconfiguration image from DDR memory. Both PMC DMAs are independent and can be used simultaneously. In the SelectMAP boot mode, the PMC DMA1 is dedicated for the data loading.

The features of the PMC DMAs include:

- Separate read channel (SRC) and write channel (DST) DMA
- Read channel fetches data from the PS-side(memory) and delivers it to the PMC stream switch (SS) interface
- Write channel receives data from the PMC stream switch (SS) interface and delivers it to the PS-side (memory)
- 128-bit AXI 3.0 interface on the PS-side
- Deep 128x128-bit data FIFOs for both the SRC and DST datapaths
- Single thread (single AXI-ID) operation for both read and write channels
- PMC DMA operates synchronously in the pmc_sec_clk clock domain
- SRC DMA only issues a read AXI command if there is enough space in the read data FIFO for the entire burst
- Start address is 32-bit aligned
- PMC DMA hardware manages alignment between the SS-side and the AXI domain
- Transfer length is in units of 4-byte (32-bit) words
- Can accept two commands per channel via a 2-deep command FIFO
- Timeout mechanisms for both SRC (read) and DST (write) channels
- Dedicated APB interface for PMC DMA register access

AES-GCM

The Versal™ device AES accelerator operates in GCM mode offering symmetric authentication, as well as decryption and encryption. Available at both boot and run time, this AES accelerator offers built-in protection against differential power attacks (DPA) and supports protocol protections (i.e., key rolling).

The AES-GCM supports a 256-bit key for boot and either a 128-bit or a 256-bit key afterward and uses a 128-bit data interface (broken into 32-bit words).

The following key sources are supported:

- Battery-backed RAM (BBRAM)
- eFUSE
- Boot header
- User key register
- Black key (PUF encrypted key storage)

Initialization Vector Register

There are four initialization vector (IV) registers. These registers store both the AES-GCM IV, as well as the data size of the next block (in support of key rolling).

SHA3-384

Versal™ ACAPs support the latest secure hash algorithm SHA3-384 standardized by NIST (FIPS-202).

The SHA3-384 hardware accelerator included in the Versal ACAP implements the SHA-3 algorithm. It is used together with the RSA accelerator to provide image authentication. It is also used to perform an integrity check of the RCU ROM prior to execution. The SHA-3 block generates a 384-bit digest value. If a design requires a 256-bit digest, the least significant 256 bits of the digest should be used (see Recommendation for Applications Using Approved Hash Algorithms NIST Special Publication 800-107).

The hash function is calculated on blocks that are 832-bits long (104 bytes). Only whole blocks can be processed through the SHA. All messages processed by the SHA-3 accelerator must be appropriately padded. See SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, NIST FIPS PUB 202 for padding requirements. SHA3-384 padding should be M || 01 || 10 * 1.

RSA/ECDSA

The public key cryptographic algorithms ECDSA and RSA are used to verify the authenticity of the programmable device image. Boot images can be authenticated using either RSA-4096 or EDCSA (NIST P-384 curve). After boot, the RSA key length or ECDSA curve is user-selectable.

The Versal™ ACAP includes an accelerator for both RSA and ECDSA math, and it is available to the user. The accelerator supports the following:

- RSA
  - Implements a modular exponentiation engine
  - R*R mod M precalculation
  - 2048, 3072, and 4096 key sizes
- ECDSA
  - Implements a point multiplier engine for elliptic curve cryptography
  - P-384 curve for initial boot
  - Support for a wide variety of NIST, SECG, SM2, and Brainpool curves for user images and data beyond initial boot

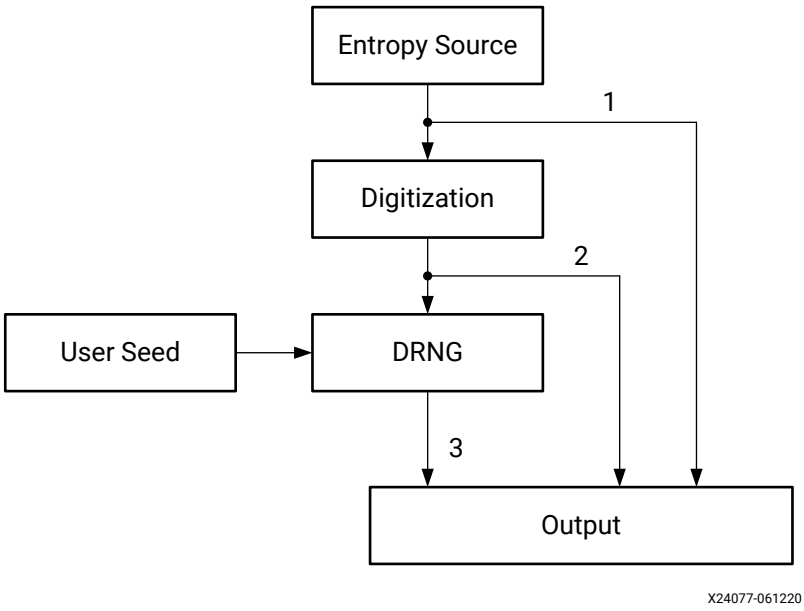
Chapter 38

True Random Number Generator

The Versal™ device contains a true random number generator (TRNG). The TRNG enables applications to be compliant with AIS-20/31 and NIST-800-90A/B/C standards. To support these standards, the TRNG operates in three modes as shown in the following figure.

1. Entropy source output
2. Internal seed + DRNG output
3. External seed + DRNG output

Figure 51: **TRNG Modes of Operation**



Features include:

- Generates cryptographically secure random numbers
- Generates data blocks with a 32-bit wide interface
- Provides security strength of 256 bits
- Ring oscillator and PLL random sources

Physically Unclonable Function

The Versal™ device contains a physically unclonable function (PUF). The PUF creates a signature (or fingerprint) of each device that is unique to that device. Its value is not “knowable” by Xilinx or the user enabling usage as a key encryption key (KEK). This KEK is 256 bits in length with 256 bits of entropy and is used to encrypt the users red key allowing its storage in black (encrypted) form. The black key can be stored in either eFUSEs, BBRAM, or external storage.

Enhanced from the previous generation, the Versal device PUF also outputs a user accessible unique ID that is cryptographically isolated from the PUF KEK itself despite using the same entropy source. While unique to each device, it is not considered a “secret” and does not have the same access protections of the KEK itself.

Battery-Backed RAM

The battery-backed RAM (BBRAM) includes 288 bits of memory for a 256-bit AES security key and 32 bits for additional information. The additional 32 bits can be left unused, used for configuration counting by the PLM, or used for user-defined purposes.

Software writes the AES key into the lower 256 bits using the eight write-only BBRAM_[0:7] registers. After writing the AES key, software writes 32 ECC bits to the BBRAM_AES_CRC register.

The write to the BBRAM_AES_CRC register causes the CRC engine to read back the AES key from memory and calculate its own CRC. The BBRAM controller CRC engine calculated CRC is then compared to the software CRC to verify the AES key was written correctly to the BBRAM memory. The BBRAM_STATUS register indicates when the verification is complete and indicates the result of the CRC using the [AES_CRC_DONE] and [AES_CRC_PASS] bits, respectively.

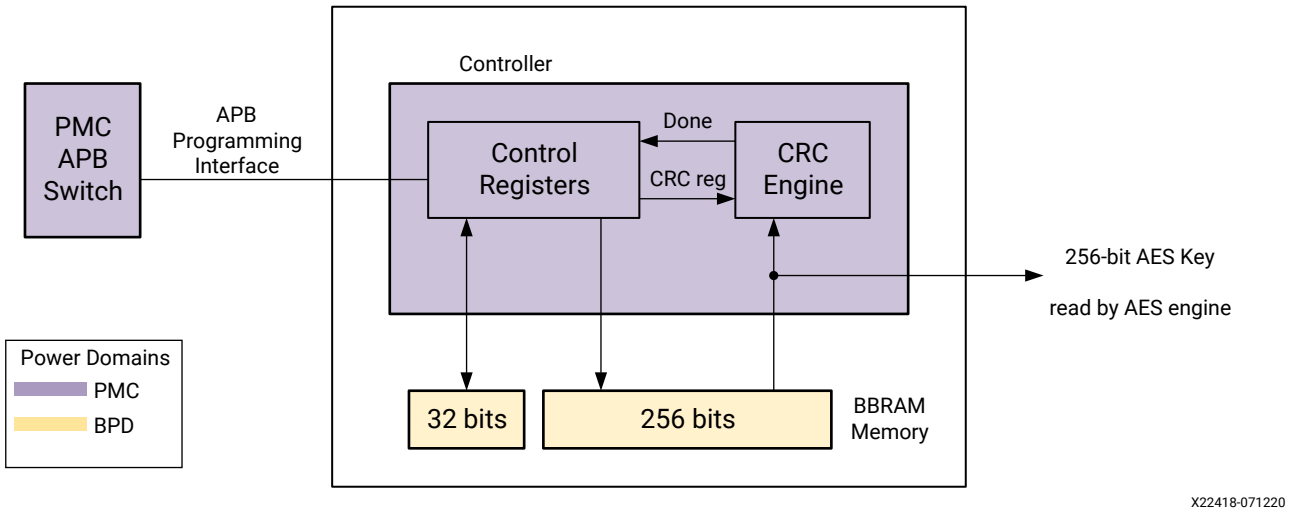
The 256-bit AES key can only be read by the AES engine. Software cannot read the 256-bit AES security key in the BBRAM.

The upper 32 bits are read/write using the BBRAM_8 register. These 32 bits can be used to store the configuration count information for the PLM. When configuration count information is not required, the 32 bits are available for user applications. The BBRAM_8 register is write protected by writing a 1 to the BBRAM_MSW_LOCK [VAL] bit. Once the bit is set, it remains set and the register becomes read-only. The PMC must be reset to clear the [VAL] lock bit.

The 256-bit AES key can be securely updated by writing to the BBRAM APB programming interface. After the key is updated, subsequent boots of the device will use the new key. Access to the BBRAM can be protected by the PMC XPPU protection unit.

The BBRAM block diagram is shown in the following figure.

Figure 52: **BBRAM Block Diagram**



eFUSE Controller

The device and data security block includes one-time programmable non-volatile eFUSE memory elements. The eFUSE memory stores design information, such as the user-defined values and security keys. The eFUSE memory can be addressed from the eFUSE controller, which is accessed via the APB. In Versal™ ACAPs, the eFUSE controller requires a separate programming voltage rail (VCC_FUSE).

The PMC device and data security block eFUSE controller enables the Vivado® development environment or system software using the XilSKey macro library to program the eFUSE. The XilSKey library is the structure to program the eFUSEs.

Versal ACAPs do not have a readback path for the eFUSE key. Instead, when the key is written, a CRC32 value of that key is provided. After the key has been written, the device verifies that the key in storage matches the provided CRC32. The device then provides a pass or fail result. The read disable eFUSE now prevents the CRC32 validation from occurring.

PMC Register Reference

The PMC includes the global register module for system software to request run-time services from the PLM.

Platform Global Registers

The global registers serve several purposes:

- Register set control and status
- Persistent and non-persistent 32-bit storage registers
- Miscellaneous status, state, and errors
- Platform service requests including power, isolation, reset, and wake-up
- Safety check

The PMC global register set is accessed via a 32-bit APB programming interface that can be accessed by any permitted system processor.

The platform service request registers allow system software to make power-up, power-down, isolation, and software reset requests by setting bits in the trigger registers. These are explained in PMC Service Requests.

PMC Global Register Set

The entire PMC global register set is summarized in the following table.

Table 68: **PMC Global Register Set Overview**

Register Name	Offset Address	Access Type	Description
Miscellaneous			
GLOBAL_CTRL	0x0000	RW, R	APB error, PLM firmware is loaded flag, MB status, and MB clock control
PMC_MULTI_BOOT	0x0004	RW	Multiboot offset address
PPU_TMR_CTRL	0x0008	RW	LMB ECC error propagation select

Table 68: **PMC Global Register Set Overview** *(cont'd)*

Register Name	Offset Address	Access Type	Description
PMC_GLOBAL_ISR PMC_GLOBAL_IMR PMC_GLOBAL_IER PMC_GLOBAL_IDR	0x0010+	W1C	APB address decode error, secure stream config error, and PUF access error
32-bit Storage Registers			
GLOBAL_GEN_STORAGE0 GLOBAL_GEN_STORAGE1 GLOBAL_GEN_STORAGE2 GLOBAL_GEN_STORAGE3 GLOBAL_GEN_STORAGE4	0x0030+	RW	General 32-bit storage registers
PERS_GLOB_GEN_STORAGE0 PERS_GLOB_GEN_STORAGE1 PERS_GLOB_GEN_STORAGE2 PERS_GLOB_GEN_STORAGE3 PERS_GLOB_GEN_STORAGE4	0x0050+	RW	General 32-bit storage registers Reset only by a POR
PMC Software Service Errors			
PMC_GSW_ERR	0x0064	RW	
PWR_SUPPLY_STATUS	0x010C	R	
Power, Isolation, Reset, and Wake-up Requests			
REQ_PWRUP_ISR REQ_PWRUP_IMR REQ_PWRUP_IER REQ_PWRUP_IDR REQ_PWRUP_TRIG	0x0110+	R, W1C R W W W	System software power down requests: - LPD - SoC power domain (SPD) - PL The SPD includes NoC and DDR power
REQ_PWRDWN_ISR REQ_PWRDWN_IMR REQ_PWRDWN_IER REQ_PWRDWN_IDR REQ_PWRDWN_TRIG	0x0210+	R, W1C R W W W	Power-up requests
REQ_ISO_ISR REQ_ISO_IMR REQ_ISO_IER REQ_ISO_IDR REQ_ISO_TRIG	0x0310+	R, W1C R W W W	Isolation requests
REQ_SWRST_ISR REQ_SWRST_IMR REQ_SWRST_IER REQ_SWRST_IDR REQ_SWRST_TRIG	0x0410+	R, W1C R W W W	System software reset requests: - PS - LPD - SPD - PL
REQ_WAKEUP_ISR REQ_WAKEUP_IMR REQ_WAKEUP_IEB REQ_WAKEUP_IDR REQ_WAKEUP_TRIG	0x0430+	R, W1C R W W W	CoreSight™ wake-up GPR for LPD and CPM
Miscellaneous			

Table 68: **PMC Global Register Set Overview** *(cont'd)*

Register Name	Offset Address	Access Type	Description
DDR_RETENTION_CTRL	0x0324	RW	Hold the XPIO output latched values to support DRAM self-refresh mode so the DDRMC power (SPD) can be shut down
DBG_PWR_ACK	0x0444	RW	CoreSight power-up acknowledge for LPD and CPM
PMC_SSS_CFG	0x0500	RW	Secure stream switch interface
PPU_MB_FATAL PPU_MBx_FT_STATUS	0x0610+	R	
PPU_RST	0x0620	RW	
PPU_RST_MODE	0x0624	RW	
PPU_AXI	0x0634	RW	PPU AXI QoS value
SAFETY_CHK	0x0800	RW	
SAFETY_CTRL	0x0804	RW	
PL_STATUS	0x0880	R	
DONE	0x0884	RW	
SEM			
SEM_STATUS SEM_ERROR	0x1018 0x1020	RW	
SEM_CMD_REG	0x1020	RW	
SEM_CRAMERR_ADDRL0 SEM_CRAMERR_ADDRH0 Etc.	0x1040+	RW	Four error addresses, eight registers
Software Mutexes			
PMC_MUTEX_[31:0]	0x1100+	RW	Software mutex registers
Register Write Locks			
PPU_RST_LOCK	0x1200	RW	Control locking of PPU_RST resettable registers
POR_LOCK	0x1204	RW	Control locking of POR resettable registers

PMC Service Requests

When system software makes a platform service request (via the PMC global registers), the PLM needs to analyze the system state and take action on the request. If the request can be granted, it accesses its PMC local registers to gracefully satisfy the request. If the request cannot be granted or there is an issue, it needs to communicate back to the requester.

The system software can make the following service requests to the PMC using the PMC Global Register Set:

- Major subsystem power up and down in Power and Isolation
- Subsystem and controller resets in Service Requests to PSM

- Test and debug CoreSight Wake-Up

Power and Isolation

The power-up, power-down, and isolation service requests for the major subsystems are triggered by setting bits in the PPU global register set. The PPU PLM software receives an interrupt request unless the interrupt is masked. The PLM manages the service request interrupts with status and mask registers.

Three power request types:

- Power up: REQ_PWRUP_TRIG
- Power down: REQ_PWRDWN_TRIG
- Isolation: REQ_ISO_TRIG

For three power domains:

- LPD
- SPD (includes NoC and DDRMC)
- PL

CoreSight Wake-Up

The CoreSight wake-up request is made by accessing the PMC_GLOBAL.WAKEUP_IRQ_TRIG register:

- Bit [0] for Arm GPR in LPD
- Bit [0] for Arm GPR in CPM

PSM Global Register Set

The entire PMC global register set is summarized in the following table.

Table 69: **PSM Global Register Set Overview**

Register Name	Offset Address	Access Type	Description
Miscellaneous			
GLOBAL_CTRL	0x0000	RW, R	APB slave error enable, FW loaded, PSM master R/W QoS, PSM sleep, wake status, and clock control
APU_PWR_STATUS_INIT	0x0008	RW	APU 0 and 1 power state value
APB_ERR_ISR APB_ERR_IMR APB_ERR_IER APB_ERR_IDR	0x0010+	R, W1C R W W	APB programming interface address decode error

Table 69: **PSM Global Register Set Overview** *(cont'd)*

Register Name	Offset Address	Access Type	Description
PS_SW_ERR	0x0020	RW	Software errors detected by PSM
PSM_BOOT_SERV_ERR	0x0024	RW	Boot and service errors detected by PSM
32-bit Storage Registers			
GLOBAL_GEN_STORAGE0 GLOBAL_GEN_STORAGE1 GLOBAL_GEN_STORAGE2 GLOBAL_GEN_STORAGE3 GLOBAL_GEN_STORAGE4 GLOBAL_GEN_STORAGE5 GLOBAL_GEN_STORAGE6 GLOBAL_GEN_STORAGE7	0x0030+	RW	General 32-bit storage registers
PERS_GLOB_GEN_STORAGE0 PERS_GLOB_GEN_STORAGE1 PERS_GLOB_GEN_STORAGE2 PERS_GLOB_GEN_STORAGE3 PERS_GLOB_GEN_STORAGE4 PERS_GLOB_GEN_STORAGE5 PERS_GLOB_GEN_STORAGE6 PERS_GLOB_GEN_STORAGE7	0x0050+	RW	General 32-bit storage registers Reset only by a POR
Power State Status			
PWR_STATE AUX_PWR_STATE	0x0100 0x0104	R	Power and retention states for power islands and memories
Power, Isolation, Reset, and Wake-up Requests			
REQ_PWRUP_ISR REQ_PWRUP_IMR REQ_PWRUP_IER REQ_PWRUP_IDR REQ_PWRUP_TRIG	0x0110+	R, W1C R W W W	System software power down requests: - APU 0, APU 1, APU L2 cache - RPU cores, TCM banks - OCM banks, GEM 0, GEM 1, FPD
REQ_PWRDWN_ISR REQ_PWRDWN_IMR REQ_PWRDWN_IER REQ_PWRDWN_IDR REQ_PWRDWN_TRIG	0x0210+	R, W1C R W W W	System software power up requests
REQ_ISO_ISR REQ_ISO_IMR REQ_ISO_IER REQ_ISO_IDR REQ_ISO_TRIG	0x0310+	R, W1C R W W W	FPD isolation request
REQ_SWRST_ISR REQ_SWRST_IMR REQ_SWRST_IER REQ_SWRST_IDR REQ_SWRST_TRIG	0x0410+	R, W1C R W W W	Subsystems and Power Islands: - APU 0, APU 1, APU MP, RPU - GEM 0, Gem 1, USB 2.0 - I/O peripherals, PS, LPD, FPD Refer to Service Requests to PSM for bit assignments.

Table 69: **PSM Global Register Set Overview** *(cont'd)*

Register Name	Offset Address	Access Type	Description
REQ_WAKEUP_ISR REQ_WAKEUP_IMR REQ_WAKEUP_IER REQ_WAKEUP_IDR REQ_WAKEUP_TRIG	0x0700+	R, W1C R W W W	Wake-up requests: - APU 0, APU 1 from APU GIC - RPU 0, RPU 1 from RPU GIC - USB 2.0 CoreSight wake-up requests: - GPR for APU 0, APU 1 Debug power for: - APU 0, APU 1, RPU, FPD
REQ_CTRL_ISR REQ_CTRL_IMR REQ_CTRL_IER REQ_CTRL_IDR REQ_CTRL_TRIG	0x0714+	R, W1C R W W W	Power-down requests: - From APU 0, APU 1 - From RPU 0, RPU 1 Reset request from RPU 0 Warm reset request: - For APU 0, APU 1 from APU MP - For APU 0, APU 1 from APU debug FPD power alarm
Miscellaneous			
DBG_PWR_ACK	0x0808	RW	Debug power-up acknowledge: - APU 0, APU 1 - RPU - FPD
SAFETY_CHK	0x0A00	RW	Safety check register

PSM Service Requests

This section includes the global registers used by system software to request several platform services:

- Power up and down of domains and islands
- Isolation of interfaces and signals between power domains
- Reset of subsystems and power domains

The service requests are serviced by the PLM software and PSM firmware. System software writes to the PSM and PMC global registers to request a service.

Power Islands

The power-up, power-down, and isolation service requests for the functional units are triggered by setting bits in the PSM global register set. The PSM firmware receives an interrupt request unless the interrupt is masked. The firmware manages the service request interrupts with status and mask registers.

The power and isolation requests for the major subsystems are listed in the following table.

Table 70: **PSM Global Power and Isolation Service Requests**

Power Island	Power-up and -down Request Register Bits	Power State Status Bits
	REQ_PWRUP_TRIG REQ_PWRDWN_TRIG	**PWR_STATE**
APU core 0 APU core 1	0 1	0 1
APU L2 cache	7	7
RPU core 0 RPU core 1	10 11	10 11
RPU 0 TCM A RPU 0 TCM B RPU 1 TCM A RPU 1 TCM B	12 13 14 15	12 13 14 15
OCM Bank 0 OCM Bank 1 OCM Bank 2 OCM Bank 3	16 17 18 19	16 17 18 19
GEM1 GEM0	20 21	20 21
FPD	22	22

Wake-Up Service Requests

The blocks that the PSM can wake up are listed in the following table.

Table 71: **PSM Wake-up Service Requests**

Block Name	Wake-up Request
	PSM_GLOBAL.REQ_WAKEUP_TRIG Register Bits
APU core 0 APU core 1	0 1
RPU core 0 RPU core 1	4 5
USB 2.0	6
APU core 0 CoreSight APU core 1 CoreSight	16 17

XILINX

Section VIII

Interconnect

This section includes these chapters:

Overview

At the chip level, the network on chip (NoC) interconnect provides access to the entire SoC for the processor software and the other masters. In the PMC and PS, the masters and slaves are clustered around three main Arm® advanced microcontroller bus architecture (AMBA®) switches. For the PMC and LPD, an I/O peripheral (IOP) switch is attached to the AMBA® switch.

All masters use the NoC to access the DDR memory, the integrated hardware, and the PL.

The TRM describes the details of the PMC-PS interconnect shown in the PS Interconnect Diagram. The NoC is described in the *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* (PG313).

Features

Transactions from masters include several attributes that are used by the interconnect and the destination, including:

- Single or burst data transfers
- Secure or non-secure TrustZone declaration
- System master identification (SMID)
- Cacheability
- Quality of service (QoS) traffic types
- Transactions include 44 or 48 bits of meaningful address
  - 44-bit addresses are received by the DDR memory and other memory-mapped destinations
  - 48-bit addresses are received by the SMMU for translation to a 44-bit physical address

The attributes are explained in Attribute Types. The parameter values used by each master are listed in PS-PL Interfaces.

Comparison to Previous Generation Xilinx Devices

This section contrasts the interconnect in the Zynq® UltraScale+™ MPSoC and the Versal™ ACAP.

The NoC and NPI interconnects are new to the Versal devices. The local AXI interconnect within the PS and the PS-PL AXI interfaces have similar functionality to the previous Zynq UltraScale+ MPSoC devices with different implementations; timeout, isolation, and parity functionality are integrated into each AMBA switch.

MMUs

The MMUs in the APU cores and the FPD SMMU are similar to previous generation Xilinx® devices.

AMBA Switches

AMBA® switches exist in each of the major subsystems including the PMC, LPD, and FPD. The contrasting functionality is shown in the following table.

Table 72: **AMBA Switch Functionality**

Function	Zynq UltraScale + MPSoC	Versal ACAP	Description
Bus isolation	AIB	Isolation	For security and power management.
Timeout	ATB	Timeout	The timeout monitors the AMBA interconnect slave connectors.
Transaction integrity error checking	~	Parity	A parity value is attached to certain signals coming into the switch. The signals are checked as they exit the switch.
Performance module probe	APM	Performance probe	Performance statistics are captured in the AMBA switches and made available to software.
Event		Event detection probe	
Trace capture		Trace capture probe	

AXI Timeout

The AXI timeout function in the Versal ACAP is based on new IP. The function is similar to Zynq UltraScale+ MPSoC, but the instances and programming model are different.

AXI and APB Isolation

Isolation is done at the AXI switch ports instead of individual AXI isolation blocks (AIB).

Xilinx Memory Protection Unit

The Xilinx memory protection unit (XMPU) is similar to the one in the Zynq UltraScale+ MPSoC. The functional differences are shown in the following table.

Table 73: **XMPU Functional Differences**

Function	Zynq UltraScale+ MPSoC	Versal ACAP	Comment
Interface	Inserted into AXI channels	Inserted into interconnect switches, XRAM ports, and DDRMC	
Error handling	Poison the base address	Issue a fail message on the interconnect	Fail message received by master
Response to address in secure range but master ID match fails	Allow or deny based on default read/write configuration	By default, transaction is denied	

Xilinx Peripheral Protection Unit

The Xilinx peripheral protection unit (XPPU) is similar to the one in the Zynq UltraScale+ MPSoC except in the way an error is signaled. The default setting is to deny a transaction. Also, the XPPU does not protect the inter-processor interrupt (IPI) controller and it does not have 32-byte apertures. The XPPU can be reprogrammed as needed during system operation. The XPPU functional differences are shown in the following table.

Table 74: **XPPU Functional Differences**

Function	Zynq UltraScale+ MPSoC	Versal	Comment
Interface	Inserted into AXI	Integrated with interconnect and interface protection	Updated interfaces
Error handling	Poison the base address	Issue a fail message on the interconnect	Fail message received by master
Total number of apertures	128 x 32 B 256 x 64 KB 16 x 1 MB 1 x 512 MB	~ 256 x 64 KB 16 x 1 MB 1 x 512 MB	Not all XPPU instances include all apertures
IPI controller	Control registers and message buffers protected by the XPPU	Control registers and message buffers protected by the IPI controller	New functionality in the IPI
IPI message buffer location	In XPPU	In IPI controller	

System Perspective

PMC and PS Interconnect

The PMC and PS are coupled together with interfaces to the PL and CPM. Several 128-bit AMBA® interfaces exist between the PS and the PL:

- LPD and FPD each have master ports to the PL
- PL has three slave ports to the PS with two-way or I/O coherency with the APU L2 cache
- PL has two slave ports to the LPD and FPD main switches
- The PS has pathways to the CPM
- The CPM has a channel to the SMMU TCU3 that connects to the FPD CCI AXI4-Lite port

The main switches in the PMC, LPD, and FPD are shown in the PS Interconnect Diagram.

Network On Chip

The TRM shows the network on chip (NoC) interconnect in high-level subsystem block diagrams, but does not explain its implementation or behavior. Refer to the NoC product guide for its descriptions, guidance on configuration, and performance tuning, *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* (PG313).

Configuration

The NoC is configured using the Vivado® IP integrator. Configuration data is written to the NoC units via the NPI programming interface.

Programming Interfaces

The APB and NPI programming interfaces enable software to access registers that control the interconnect and all the other system units. The APB programming interfaces are located throughout the PMC and PS on the local interconnects as slaves. The NPI programming interface is controlled by a single master located in the PMC. The NPI master is controlled by software using register accesses. Both interfaces provide error reporting. The programming interface summary includes:

- APB: 32-bit RW, multiple interface locations
- NPI: 32-bit RW with burst, single bus master located in the PMC

Memory Virtualization

In some designs, multiple operating systems are required to run on the APU MPCore. Running multiple guest operating systems on a CPU cluster requires hardware support to virtualize the processor system into multiple virtual machines (VM) to allow each guest operating system to run on its VM.

Operating systems are generally designed to run on native hardware. The system expects to be executing in the most privileged mode and assumes total control over the whole system. In a virtualized environment, it is the VM that runs in privileged mode, while the operating system is executing at a lower privilege level.

When booting, a typical operating system configures the processor, memories, I/O devices, and peripherals. When executing, it expects exclusive access to such devices, including changing peripherals' configuration dynamically, directly managing the interrupt controller, replacing MMU page table entries (PTE), and initiating DMA transfers.

When running de-privileged inside a virtual machine, the guest operating system is not able to execute the privileged instructions necessary to configure and drive the hardware directly.

The VM must manage these functions. In addition, the VM could be hosting multiple guest operating systems. Therefore, direct modification of shared devices and memory requires cautious arbitration schemes.

The level of abstraction required to address this, and the inherent software complexity and performance overhead, are specific to the characteristics of the architecture, the hardware, and the guest operating systems. The main approaches can be broadly categorized in two groups.

- Full virtualization
- Paravirtualization

In full virtualization, the guest operating system is not aware that it is virtualized, and it does not require any modification. The VM traps and handles all privileged and sensitive instruction sequences, while user-level instructions run unmodified at native speed.

In paravirtualization, the guest operating system is modified to have direct access to the VM through hyper-calls or hypervisor calls. A special API is exposed by the VM to allow guest operating systems to execute privileged and sensitive instruction sequences.

The Arm® Cortex™-A72 exception level-2 (EL2) provides processor virtualization. The Arm® v8 supports virtualization extension to achieve full virtualization with near native guest operating system performance.

System Perspective

The system memory management unit (SMMU) enables system masters to share sections of the APU memory map. The CCI provides the option to interact with the APU's L2 cache.

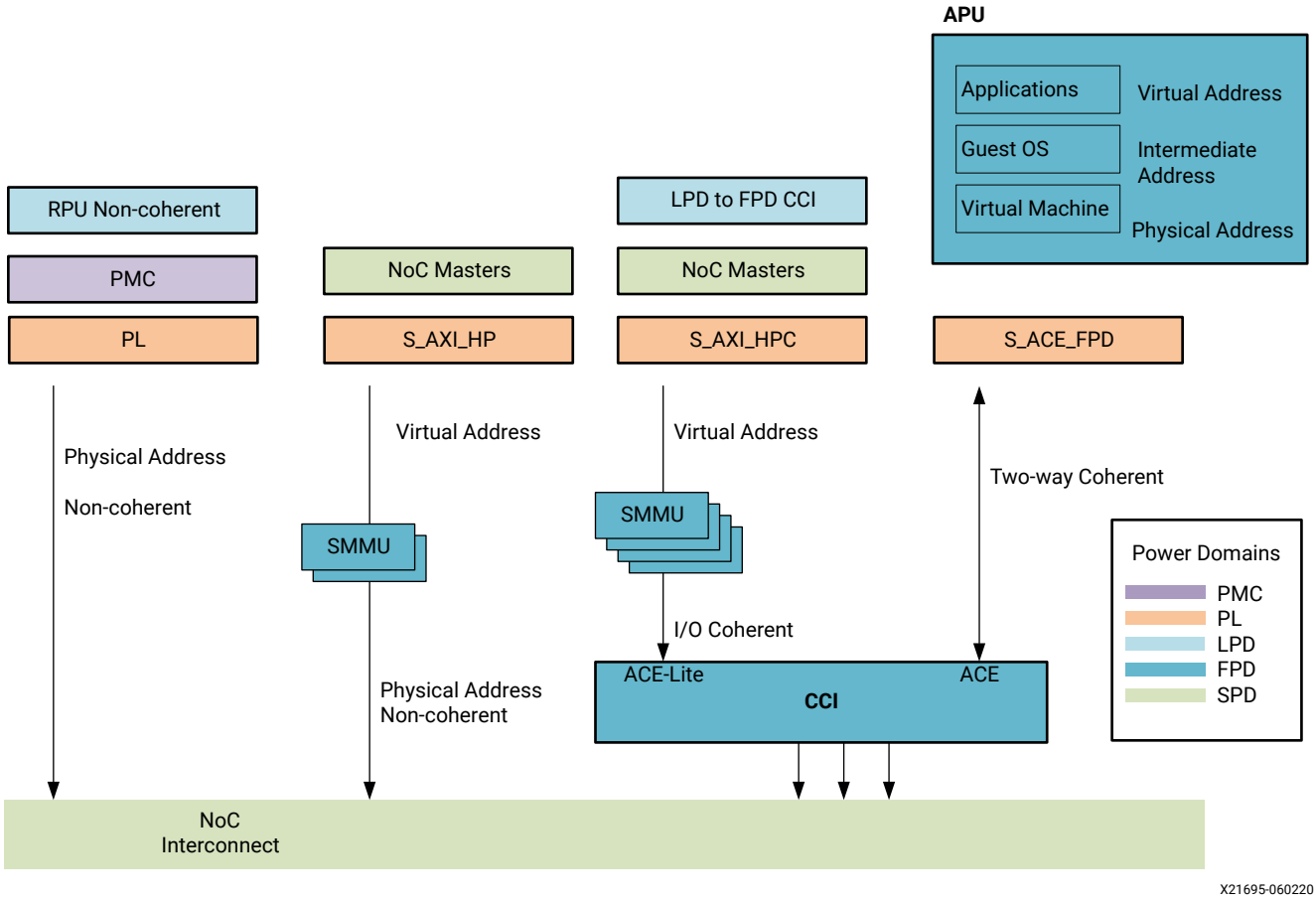
- The address from masters are interpreted by the SMMU as a virtual address that is mapped to a physical address. The physical address is used by the CCI and system memories.

The PS includes memory management units (MMUs) to support multi-OS and multi-threaded kernels.

- APU MMU includes two-stage address translation (with natural page table look-up memory protection).
- FPD SMMU includes two-stage address translation (with natural page table look-up memory protection).

The PS memory address translation is shown in the following figure.

Figure 53: **PS Memory Address Translation**



APU Virtualization

A processor element is in hypervisor mode when it is executing at EL2 in the AArch64 state. An exception return from hypervisor mode to software running at EL1 or EL0 is performed using the ERET instruction.

EL2 provides a set of features that support virtualizing the non-secure state of an Arm v8-A implementation. See the Arm® Architecture Reference Manual Arm® v8 for more information.

The basic model of a virtualized system involves the following:

- A hypervisor software, running in EL2, is responsible for switching between virtual machines. A virtual machine is comprised of non-secure EL1 and non-secure EL0.
- A number of guest operating systems, that each run in non-secure EL1, on a virtual machine.

- For each guest operating system, there are applications that usually run in non-secure EL0, on a virtual machine.

The hypervisor assigns a virtual machine identifier (VMID) to each virtual machine. EL2 is implemented only in a non-secure state, to support guest operating system management.

EL2 provides information in the following areas:

- Provides virtual values for the contents of a small number of identification registers. A read of one of these registers by a guest operating system or the applications for a guest operating system returns the virtual value.
- Traps various operations, including memory management operations and accesses to many other registers. A trapped operation generates an exception that is taken to EL2.
- Routes interrupts to the appropriate area:
  - The current guest operating system.
  - A guest operating system that is not currently running.
  - The hypervisor.

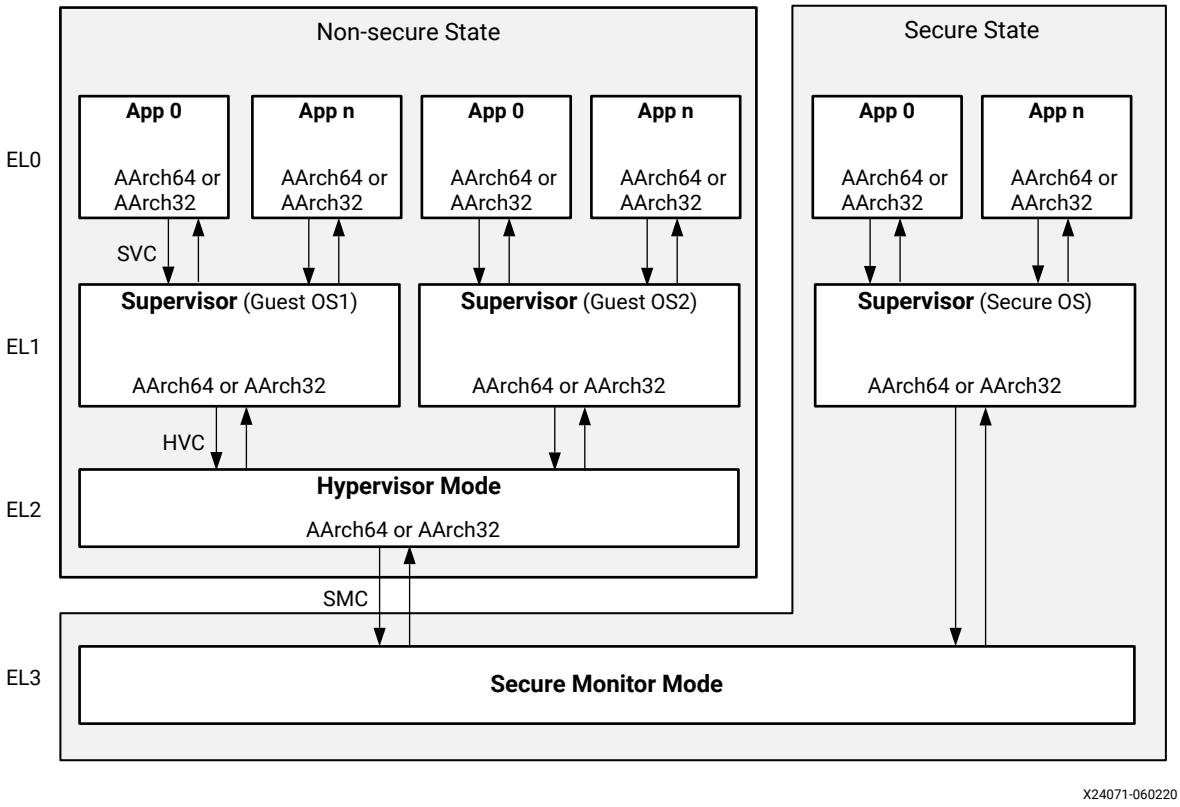
In a non-secure state, the following occurs:

- The implementation provides an independent translation regime for memory accesses from EL2.
- For the EL1 and EL0 translation regime, address translation occurs in two stages.
  - Stage 1 maps the virtual address (VA) to an intermediate physical address (IPA). This is managed at EL1, usually by a guest operating system. The guest operating system believes that the IPA is the physical address (PA).
  - Stage 2 maps the IPA to the PA. This is managed at EL2. The guest operating system might be completely unaware of this stage. Hypervisor creates the stage 2 translation table.

Execution Modes

The following figure shows the Arm v8 execution modes discussed in this section.

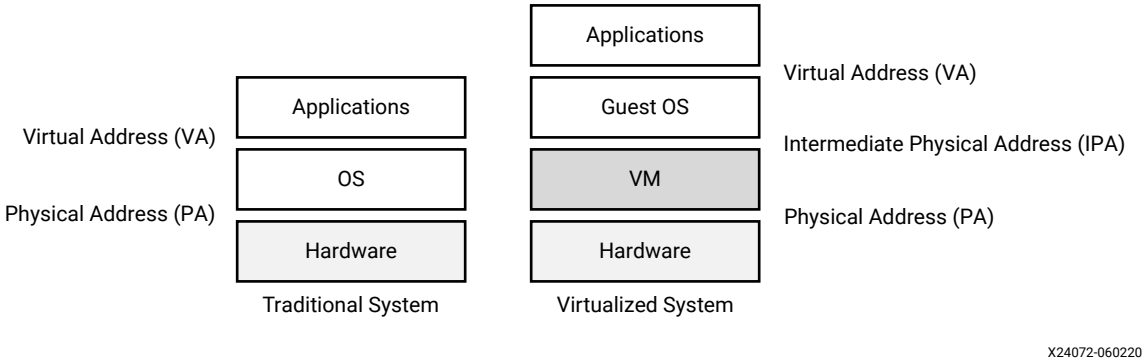
Figure 54: **Arm v8 Execution Modes**



Virtualized System

The hypervisor directly controls the allocation of the actual physical memory, which fulfills its role as arbiter of the shared physical resources. This requires two stages (VA→IPA, and IPA→PA) of address translation. The following figure shows the traditional versus virtualized systems addresses in the translation stage.

Figure 55: **Traditional Versus Virtualized Systems Address Translation Stage**



Interrupt Virtualization

The APU GIC v3 interrupt virtualization is a mechanism to aid interrupt handling, with native distinction of interrupt destined to secure-monitor, hypervisors, currently active guest operating systems, or non-currently-active guest operating systems. This reduces the complexity of handling interrupts using software emulation techniques in the hypervisor.

System Memory Management Unit

The system memory management unit (SMMU) includes these functions:

- Address translation
- Transaction security state control
- Memory protection using page table look-ups

These functions are performed with a combination of the seven translation buffer units (TBU 0 to 6). Four of these are in the path of incoming AXI interfaces to the CCI. The translation and protection tables that are cached in the TBU are updated by the SMMU translation control unit (TCU).

The functions of the SMMU are performed in the TBUs and include:

- Translates 48-bit virtual address from processors and other masters into a 44-bit physical address for DDR memory and other memory-mapped destinations
- Memory protection from unauthorized or errant accesses

TBU Instances

There are seven TBUs supported by the SMMU TCU. These are listed in the following table with their system masters and destinations.

Table 75: **System Masters**

SMMU Unit	System Masters	Destination
TBU0	LPD AXI protocol	CCI S3 ACE-lite
TBU1	NoC NSU3	CCI S2 ACE-lite
TBU2	Switch from S_AXI_HPC ACE_lite, or NoC NSU2	CCI S1 ACE-lite
TBU3	CPM	CCI S0 ACE-lite, MSI
TBU4	Noc NSU1	Switch to FPD main switch
TBU5	NoC NSU0, S_AXI_HP, CoreSight	FPD main switch or NoC NMU
TBU6	Noc NSU	NoC NMU

Address Translation Examples

The SMMU provides address translation for an I/O device to identify more than its actual addressing capability. In the absence of memory isolation, I/O devices can corrupt system memory. The SMMU provides device isolation to prevent DMA attacks. To offer isolation and memory protection, it restricts device access for DMA-capable I/O to a pre-assigned physical space.

Native, Non-Virtual

As an example, consider the AXI interfaces from the programmable logic to the PS that pass through the SMMU in the PS. When enabled, the SMMU also offers protection from DMA masters in the PL restricted access to the PS memory region. This is protection in the context of a symmetric multiprocessing system running an OS. The OS on an APU can isolate the DMA from interfering with other devices under the APU. In a similar way, the SMMU can also be enabled to restrict DMA units or other PS masters from accessing the PS memory region.

Virtual

The SMMU enables address translation in a virtualized system. An SMMU provides isolation among different guest operating systems by setting appropriate translation regimes and context. This isolation among guest operating systems prevents malfunction, faults, or hacks in one domain from impacting other domains. An SMMU provides system integrity in a virtualized environment.

Additionally, the SMMU supports two security states. In a system with secure and non-secure domains, SMMU resources can be shared between secure and non-secure domains. For details on two security states in the SMMU, see the Arm System Memory Management Unit Architecture Specification.

Stream IDs

The SMMU uses a 15-bit stream ID to perform address translations. This information is part of a transaction and indicates which master originated the request.

Stream ID bits [9:0] are the same as the system master ID (SMID) bits (see APU SMID [3:0]). Stream bits [14:10] are assigned by the TBU that the transaction passes through. There are seven TBUs:

- TBU 0 appends `000_00b`
- TBU 1 appends `000_01b`
- Etc.

- TBU 6 appends `001_10b`

Memory Protection Functionality

When an address space is not mapped to valid address translation entry, then the transaction generates a slave error back to the master.

Cache Coherent Interconnect

The cache coherent interconnect (CCI) enables system processors and other masters to share data with the APU MPCore L2 cache using hardware-managed coherency. The coherency is done by routing the system master bus transaction through to the CCI. The CCI is implemented using the Arm® CCI-500 IP.

There are two types of coherency ports:

- Two-way coherency (ACE ports to APU and PL)
- I/O coherency (four ACE-light ports)

System-level coherency enables the sharing of memory by system components without the software requirement to perform cache maintenance to maintain coherency between caches. Regions of memory are coherent if writes to the same memory location by two components are observable in the same order by all components.

***Note*:** Processors in the PL can be I/O coherent with the APU L2 cache by accessing the ACP interface to the snoop control unit (SCU) in the APU MPCore. This PL to PS interface is described in ACP Interface. The ACP interface does not connect through the CCI.

The CCI has four masters that connect to the HNoC structure at the south side of the device. The CCI provides interconnectivity to the LPD, CPM, and PL, but it cannot access the PMC directly. The connections are shown in PS Interconnect Diagram.

Slave Port Interfaces

There are two types of CCI slave interface ports.

Two-way Coherency

The 128-bit S_ACE_FPD interface from the PL to PS provides full two-way hardware coherency between the APU MPCore and a system master in the PL. The two-way coherency allows a PL master to snoop the APU L2 cache and the APU MPCore to snoop the PL. The S_ACE_FPD interface is an extension to the AXI protocol and provides hardware cache coherency.

I/O Coherency

I/O coherency (one-way) enables system masters to snoop the APU L2 cache. If there is a read hit in the L2 cache, then the L2 cache sources the data. If there is a write hit in the L2 cache, then the action depends on the coherency policy requested by the master. I/O coherency with the CCI is supported by the S_AXI_HPC interface attached to an ACE-Lite port of the CCI. Master can also access the SMMU and CCI via NoC.

The ACE coherency protocol ensures that all masters observe the correct data value at any given address location by enforcing that only one copy exists whenever a store occurs to the location. After each store to a location, other masters can obtain a new copy of the data for their own local cache, allowing multiple copies to exist. See the Arm® AMBA® AXI and ACE protocol specification for a detailed overview.

Master Port Interfaces

All master and slave interfaces are 44-bit address and 128-bit data.

Destinations

The CCI master port destinations are listed in the following table.

Table 76: **CCI Master Port Destinations**

CCI Master Port	Destination				
	PL Interfaces: M_AXI_FPD M_AXI_LPD	OCM and XRAM	FPD/LPD/PMC slaves	CPM	NoC includes access to: DDR, PL, AI Engine
M0	X	X	X	~	~
M1	~	~	~	X	~
M2 M3 M4 M5	~	~	~	~	X

CCI Routed Traffic

Transactions coherent to the APU L2 cache must be routed through the CCI. However, traffic routed through the CCI does not necessarily need to be tagged for coherency. In this case, the transaction goes through the CCI without coherency checks.

Memory Protection

The interconnect has several features that protect the system slaves from erroneous application software and misbehaving hardware interfaces. Erroneous software includes malicious and unintentional code that corrupts system memory or causes system failures. Misbehaving hardware includes incorrect device configuration, malicious functionality in the PL, or an unintentional design.

Each bus master is assigned a master ID number. Each master specifies a read/write access type and address for each transaction. In addition, the Arm TrustZone technology tags the security level of each AXI transaction. The access type, address, and security level are checked by protection mechanisms before reaching the destination to determine if the master has the authority to access the requested memory (this includes memory locations and memory-mapped registers).

Functional Units

The MMUs restrict access using page table faults. The protection units compare the credentials of the master with the current access controls for that address location.

- APU MPCore MMU page mapping
- SMMU page mapping
- XMPU protection units
- XPPU protection units

The following table summarizes the system protection units terminology.

Table 77: **Memory Protection**

Access Unit	Description
APU MPCore MMU	Monitors the transactions from the APU processors.
FPD SMMU	The SMMU includes one translation control unit (TCU) and six translation buffer units (TBU). The SMMU provides protection (and address translation) for all non-APU transactions targeting the PS address space. The protection functionality is applied to the physical address that occurs after the address translations. The SMMU registers are accessible only from the APU.

Table 77: **Memory Protection** *(cont'd)*

Access Unit	Description
XMPU	The XMPUs provides memory partitioning and TrustZone protection for memories: - PMC RAM and SBI - OCM memory port - XRAM memory ports - DDR memory controller
XPPU	Several XPPUs: - PMC_XPPU to PMC CFU and IOP programming interfaces - NPI_XPPU to NPI programming interfaces - LPD_XPPU to LPD programming interfaces and PSM slave access

Use Case Examples

In this system protection use case, the RPU runs a safety application where a certain region of the OCM might be required to be protected and dedicated for use by the RPU. Some peripherals like the UART controller and the QSPI controller might also require protection and be dedicated for use by the RPU. The following are needed to accomplish these requirements:

- RPU generates transactions
- XMPU protects the region of the OCM for access only by the RPU master ID; other regions of the OCM can be accessed by other masters
- XPPU protects the UART controller and QSPI controller for use by the RPU master ID

TrustZone Security

The TrustZone technology provides a foundation for system-wide security. TrustZone technology is a software-controlled, hardware-enforced system for separating secure and non-secure AXI transactions. Masters are assigned a security profile that is either statically controlled (always secure or always non-secure), or dynamically controlled using a configuration register. Similarly, software processes are assigned a secure or non-secure state.

A non-secure application can only access non-secure system resources, whereas, a secure application can see all resources. Resource access is extended to bus accesses using the non-secure, NS flag, which is mapped to the AxPROT[1] signal in the AXI protocol. Any part of the system can be designed to be part of the secure world including debug, peripherals, interrupts, and memory. By creating a secure subsystem, assets can be protected from software and hardware attacks.

Xilinx Memory Protection Unit

The Xilinx memory protection unit (XMPU) verifies that a system master is explicitly allowed to access a memory address. The XMPU is a region-based memory protection unit. This section describes the XMPU in detail, including configuration and functionality.

System protection is applied to incoming AXI transaction requests from masters. The transaction is checked for:

- Master ID (unique for each master)
- TrustZone secure or non-secure (NS)
- Read or write access type

XMPUs appear on the slave interconnect ports:

- PMC_XMPU on IOP flash and CFU
- OCM_XMPU for the OCM memory
- FPD_XMPU on slaves
- Accelerator RAM (when present), with an XMPU on each of the four AXI ports
- DDR memory controller ports; one per controller

The XMPUs are shown in the system block diagram PS Interconnect Diagram.

Use Case Example

In this system protection use case, the RPU runs a safety application where a certain region of the OCM might be required to be protected and dedicated for use by the RPU. To accomplish these requirements, the following are required:

- RPU generates secure transactions
- XMPU can protect a region of the OCM for exclusive use by the RPU and makes the rest of memory available for use by allowed masters

Features

General Features

The XMPU includes these features:

- Slave interconnect port to receive a transaction
- Master interconnect port to addressed slave
- Interconnect clock is the same for master and slave ports
- Programming interface to access control registers
- Lock register
  - Configure with TrustZone secure transactions prior to registers being locked
  - When the lock is set, writes to interrupt registers are the only ones allowed
  - Once the lock is set, the lock is only unlocked by a POR reset
- Error and status reporting
  - Failed message generated and passed to addressed slave
  - Interrupt for interconnect transaction permission violation
  - Interrupt for register access violations
- Memory partitioning and protection
  - Isolate a master
  - Give set of masters access to address ranges

Region Features

- 16 regions defined
- Each region with start address and end address
- Define address range with 4 KB or 1 MB granularity
- Regions can overlap
  - In case of overlap, the higher region# has higher priority
- Each region can be independently enabled or disabled
- Disabled region is not used for protection checking
- Read and write permission for each region can be independently enabled or disabled
- Each region can be independently set to secure or non-secure

- If permission or secure check violations are detected, then the transaction is blocked and an error is generated

System Perspective

PMC and PS Instances

The following table lists the system protection units. The region size is fixed at 4 KB.

Table 78: **XMPU PMC and PS Instances**

Name	Granularity	Upstream Master	Downstream Slave	Control Registers Base Address
PMC_XMPU	4 KB	PMC main switch	IOP, flash, CFU	0xF12F_0000
OCM_XMPU	4 KB	OCM switch	OCM memory	0xFF98_0000
FPD_XMPU	4 KB	FPD main switch	APU GIC and FPD slaves	0xFD39_0000

XRAM and DDRMC Instances

The following table lists the XRAM and DDRMC instances.

Table 79: **XMPU XRAM and DDRMC Instances**

Name	Granularity	Upstream Master	Downstream Slave	Programming Interface
Accelerator RAM				
XRAM_LPD_XMPU	4 KB	OCM switch	Accelerator RAM	APB
XRAM_PL0_XMPU XRAM_PL1_XMPU XRAM_PL2_XMPU		PL fabric		
DDR Memory Controller (one per controller)				
DDRMCx_XMPU	1 MB	Four-port NoC interface	DRAM controller	NPI

Memory Regions

Each XMPU has 16 regions, numbered from 0 to 15. Each region is defined by a start address and an end address. There are two region address alignment types. The 4 KB granularity is used for all XMPUs except the one in the DDRMC.

When a memory space is included in more than one XMPU region configuration, the higher region number has higher priority (that is, region 0 has lowest priority). Each region can be independently enabled or disabled. If a region is disabled, it does not include protection checking.

If none of the regions are enabled or the request does not match any of the regions, then a subtractive decode determines whether or not the request is allowed. That is, the XMPU takes the default action as specified in the XMPU control register.

Access Checking Operations

An incoming read or write request on a port is checked against each XMPU region as described in this section.

TIP: *When a memory space is included in more than one XMPU region, the higher region number has higher priority (that is, region 0 has the lowest priority). This determines the set of permissions used for the checks described in this section.*

For the enabled region, two basic checks are completed first.

- Check if the address of the transaction (AXI_ADDR) is within the region. That is, START_ADDR ≤ AXI_ADDR ≤ END_ADDR.
- Check whether the master ID of the incoming transaction is allowed. That is, incoming_MID & MID_Mask == MID_Value & MID_Mask.

If these checks are true, then the region configuration is checked with regards to security and read and write permissions.

***Note*:** Disabled regions do not grant permissions.

Master ID Validation

Each XMPU uses the inbound Master ID in each AXI transaction to validate the transfer. The Master ID is masked by the [MASK] bit field and then compared against the [ID] bit field of the Rxx_MASTER region registers. If the following equation is satisfied (along with security and read/write checks), the transaction is allowed. In this equation, these are [10-bit parameters] in the Rxx_MASTER region register:

$$[ID] \ \& \ [MASK] == AXI\_MasterID \ \& \ [MASK]$$

Security Validation

- If the region is configured as secure, then only the secure request can access this region.

- If the region is configured as secure, then the read and write permissions are independently checked to determine whether or not the transactions are allowed.
- If the transaction is non-secure and the region is configured as secure, then the check fails, and the transaction is handled as described in Error Handling.
- If the region is configured as non-secure and the transaction is non-secure, then read and write permissions are independently checked to determine whether or not the transaction is allowed. If the check fails, the transaction is handled as described in Error Handling.

Error Handling

Errors can occur from security or read/write violations. When an error occurs, the XMPU records the type of violation, the transaction address, and the master ID of the first transaction that failed. The protection unit flags the violation and can generate an interrupt. When a security violation occurs, there is an additional logging to indicate that the error was a security violation. Only one error and the first error is recorded for both read/write AXI channels.

Permission and Security Violations

When a permission or security violation is detected, the XMPU asserts the error flag in the transaction header. This header is read to determine what action to take. The XMPU also records the address, error type, and master ID number. An interrupt can be generated. Only the first occurrence of an error is recorded. For simultaneous read and write errors, only the write error is recorded.

The transaction is stored in these registers:

- XMPU.ERR_STATUS1_LO, ERR_STATUS1_HI (address)
- XMPU.ERROR_STATUS2 (master ID number)

The type of violation is recorded in the XMPU.ISR register:

- [SecurityVIO]
- [WrPermVIO]
- [RdPermVIO]

Transaction Signals

The format of the transaction signals is shown in the following table. There are two sets of signals, one for reads and one for writes.

Table 80: **Transaction Signals**

Entry	Description
Master offset address	Transaction address
User flags	Master ID
Security flag	Protection (PROT)
Valid	Valid indicator
Hold	
Error flag	Pass/fail result
Read or write indicator	0 = write, and 1 = read
Hide	Hide control from XMPU.CTRL [HideAllowed] bit output from XMPU to protection wrapper
Sideband	Interrupt

Configuration

The XMPU is configurable either one time or through a secure master. At boot time, the XMPU can be configured and its configuration is locked. If an XMPU register set is locked, the XMPU can only be reconfigured after the next system reset. If the configuration is not locked, the XMPU can be reconfigured any number of times by trusted software (using a secure master).

RECOMMENDED: *Xilinx recommends only configuring each XMPU one time. When an XMPU is programmed, all of its settings must be programmed to ensure that only the allowed transactions go through.*

XILINX

Chapter 49

Xilinx Peripheral Protection Unit

The Xilinx peripheral protection unit (XPPU) protects the system addressable programming registers from erroneous application software and misbehaving hardware interfaces. Erroneous software includes malicious and unintentional code that corrupts system register settings or causes system failures. Misbehaving hardware includes incorrect device configuration, malicious functionality, or unintentional design.

There are several XPPU in the LPD and PMC for register programming:

- PMC main switch to APB slave interfaces (PMC_XPPU)
- PMC main switch to NPI master unit that accesses the NPI slave interfaces (PMC_XPPU_NPI)
- LPD main switch to APB slave interfaces (LPD_XPPU)

The XPPUs are identified in the system PS Interconnect Diagram.

The XPPU looks at several transaction attributes to determine if the transaction should be allowed to proceed normally. The attributes include the 44-bit physical address, the AxPROT[1] security bit, and the system master ID (SMID) bits that are encoded in the AxUSER command signals. These attributes are used to restrict the access to memory mapped peripheral interfaces.

Features

The XPPU has several features:

- Access control for a specified set of address apertures on a per-master basis
- Access control granularity on a per-peripheral or a per-message buffer basis
- Up to 20 simultaneous sets of one or more masters
- Several sets of programmable apertures, including:
  - 256 x 64 KB for peripheral slave ports
  - 16 x 1 MB for peripheral slave ports
  - Single 512 MB for flash memory controller
- AXI transaction permission violation interrupt

- APB slave interface address decode error interrupt

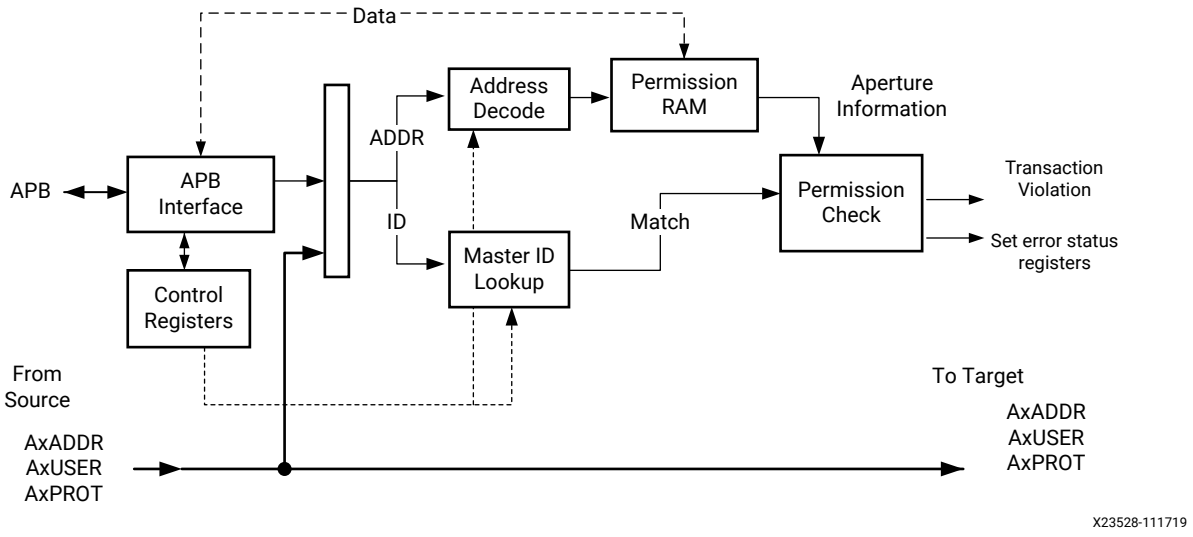
The XPPU interfaces consist of the following:

- Slave AXI port where master ID is carried on lower bits of AxUSER
- Master AXI port where master ID is carried on lower bits of AxUSER
- AXI clock (same for master and slave ports)
- APB slave programming interface (requires secure transactions to access)
- Level-sensitive, asynchronous interrupt output

System Perspective

The basic protection operations and their system interfacing to the system is shown in the following figure.

Figure 56: **XPPU Match and Permission Diagram**



Instances

The locations of the various system protection modules in the PS are shown in the PS Interconnect Diagram.

The following table lists the system protection units.

Table 81: **XPPU Instances**

Name	Upstream Master	Downstream Slave	Control Registers Base Address
LPD_XPPU	LPD main switch	LPD IOP slave switch	0x00_FF99_0000
PMC_XPPU	PMC main switch	PMC IOP slave switch	0x00_F131_0000
PMC_XPPU_NPI	PMC main switch	NPI bus master	0x00_F130_0000

Access Checking Operation

For every read and write transaction, the XPPU determines if the transaction is allowed to proceed with fine grain control of specific addresses. If the transaction is allowed, then it proceeds normally. If the transaction is not allowed, it asserts an error flag that is detected downstream.

An AXI transaction request is allowed to access the address space defined by an APERPERM_xxx register if these conditions are satisfied:

- The requesting master fits one or more of the profiles of a MASTER_IDxx register.
- The bit for the that profile is set in the [PERMISSION] bit field. For example, if the master satisfies the MasterID and read/write permissions of the MASTER_ID00 register and bit 0 of the [PERMISSION] bit field = 1, then the transaction is allowed to proceed.
- The transaction request satisfies the APERPERM_xxx [TRUSTZONE] bit setting.

IMPORTANT! *XPPU is used to configure the device control address space to be TZ or non-TZ. Devices (peripherals) are configured to be TZ or non-TZ by separate registers—this control is not provided by XPPU.*

Aperture Permissions

Aperture List

The following table shows the four sets of apertures and the address protected for each aperture.

Table 82: **XPPU Aperture List**

Aperture Size	Address Interval	Number of Apertures	Base Address of Aperture Registers		
			PMC_XPPU	PMC_XPPU_NPI	LPD_XPPU
64 KB	0x0001_0000	256	0xF100_0000	0xF600_0000	0xFF00_0000

Table 82: **XPPU Aperture List** *(cont'd)*

Aperture Size	Address Interval	Number of Apertures	Base Address of Aperture Registers		
			PMC_XPPU	PMC_XPPU_NPI	LPD_XPPU
1 MB	0x0010_0000	16	0xF000_0000	0xF700_0000	0xFE00_0000
512 MB	0x2000_0000	1	0xC000_0000	Does not exist	0xE000_0000

Master ID Entry

When an AXI transaction is received, the master ID (MID) that is available as part of AxUSER is compared against all entries of the MID list, together with parity checks (if enabled).

An AXI MID matches the n^{th} entry if the following is true:

(MASTER_IDnn.MASTER_ID_MASK & MID == MASTER_IDnn.MASTER_ID_MASK & MASTER_IDnn.MASTER_ID) && (~CTRL.MID_PARITY_EN || CTRL.MID_PARITY_EN & (MASTER_IDnn.MASTER_ID_PARITY == Computed parity))

An entry in the master ID list consists of the fields shown in the following table.

Table 83: **XPPU Master ID Entry**

Name	Bit Field	Bit Field	Description
Master ID	MID	[9:0]	The master ID to match
ID mask	MIDM	[25:16]	The ID mask
Read-only	MIDR	[30]	If set, only read transactions are allowed
Register parity	MIDP	[31]	Parity of bits [30, 25:16, 9:0]

For a matched entry, if it is enabled by the corresponding bit of the PERMISSIONS field (as defined by the PERM field shown in Table 84: Aperture Permissions Register Format) and if the read only (MASTER_IDnn.MIDR) bit is set, only read transactions are allowed and write transactions are not allowed.

The bitwise result of matching against each entry of the master ID list is stored in the match vector (MATCH [m–1:0]. The parity bit is computed and written by the software if the parity option is enabled.

Register Format

The XPPU aperture register structure enumerates the permission settings on each protected peripheral. Each APERPERM_xxx register entry contains the information listed in the following table.

Table 84: **Aperture Permissions Register Format**

Field Name	Bit Field	Description
PERMISSION	[19:0]	Master ID profile permission. Each of the 20 [PERMISSION] bits correspond to the MASTER_ID{0:19} registers. The [PERMISSION] field helps to determine if the transaction request of the master characterized by a MASTER_ID register is permitted. 0: Not allowed 1: Allowed A 1 in bit position n (n < m) indicates that the nth entry in the master ID list has permission to access the aperture. This check is further qualified by parity and TrustZone checks.
TRUSTZONE	[27]	0: Only secure transactions are allowed 1: Secure or non-secure transactions are allowed
PARITY	[31:28]	The hardware checks the parity bits for the [PERMISSION] and [TRUSTZONE] bit fields. Software must generate and load the parity bits before the protection unit uses the register.

Four parity bits are added to protect the (TrustZone and permission) fields, which are equally divided into four protected fields. Parity must be computed by software when writing an entry in the aperture permission list. If the controller detects a parity error, then a status bit is set.

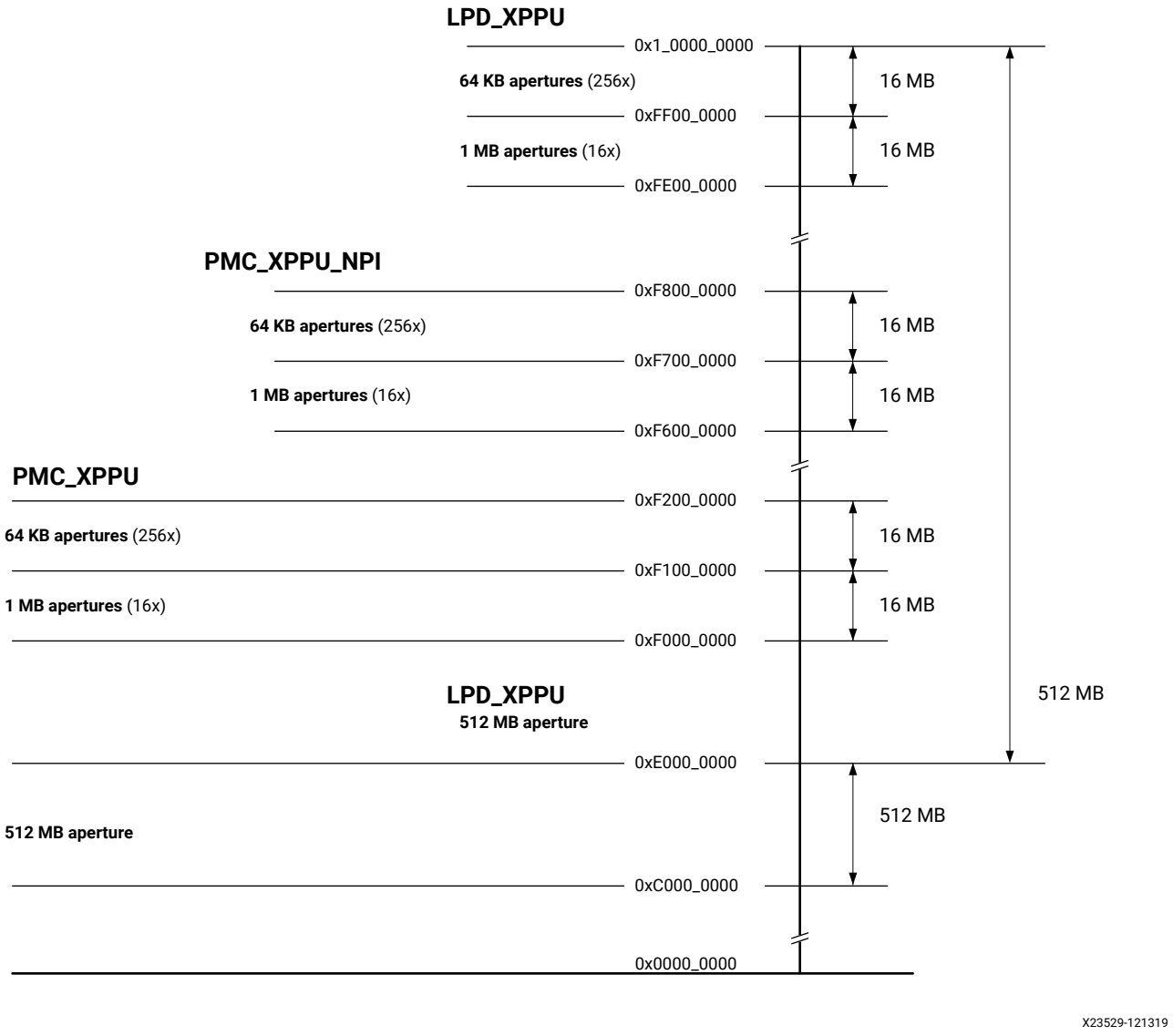
- Bit [31] is parity for bit [27] and bits [19:15]
- Bit [30] is parity for bits [14:10]
- Bit [29] is parity for bits [9:5]
- Bit [28] is parity for bits [4:0]

The aperture permission list must be completely initialized by software to 0 before the XPPU can be enabled. The software is also required to compute and write parity. For unprotected apertures, all supported master match bits in the permission RAM should be set to 1.

Protected Addresses

The XPPU protects the address ranges shown in the following figure.

Figure 57: **XPPU Aperture Memory Map**



Permission Checking

Permission checking is performed using the master ID and TZ security settings of the transaction. The MasterID sets one or more of the 20 local MATCH bits that are compared against the address-selected aperture permission register, APERPERM_xxx. The XPPU also tests the AxPROT[1] and R/W signals with the APERPERM_xxx [TRUSTZONE] bit. The following equation is for read transactions.

$$\text{Transaction\_OK} = (\text{MATCH} \ \& \ \text{PERMISSION} \ != 0)$$

AND { (TRUSTZONE == 1) OR {(AxPROT[1] == 0) && (TRUSTZONE == 0) }}

- The first term means that the incoming AXI master ID, after the mask is applied, should be listed in the master ID list, and it should also be listed as an allowed master in the aperture permission list, APERPERM_xxx registers.
- The second term means that the incoming AXI TrustZone (on AxPROT [1]) should meet the aperture (slave) TrustZone setting.

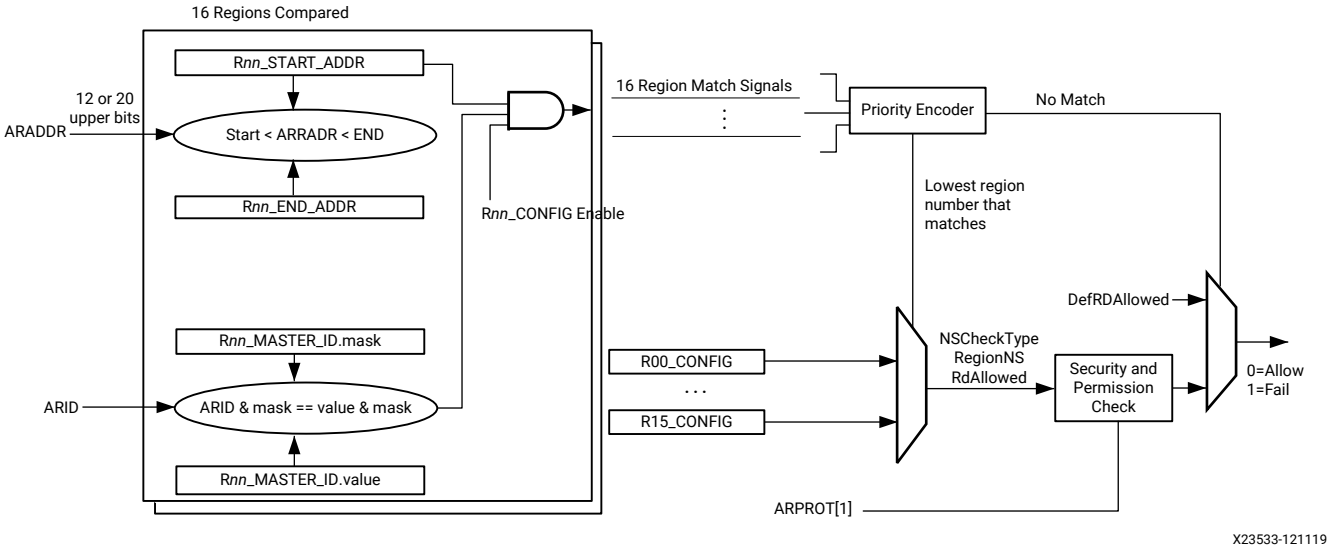
The result from this equation is further qualified with the parity check on the selected register from the aperture permission list if the parity check is enabled.

If all of the these checks pass, the transaction is allowed.

Block Diagram

The XPPU functional block diagram is shown in the following figure.

Figure 58: **XPPU Functional Block Diagram**



Error Handling

The following table lists the possible errors that can be encountered by the XPPU and how they are handled.

Table 85: **Error Handling in XPPU**

Error[1]	Actions
Master ID list parity error	The MASTER_IDnn register associated with the parity error is disabled and cannot enable a match, that is, MATCH [nn] is forced to 0. The MID_PARITY bit of the ISR register is set and an interrupt can optionally be signaled.
Master ID list read only error	A master ID read-only error occurs when any matched MASTER_IDnn register is enabled by the corresponding bit of the PERM field from the selected entry for the addressed peripheral, its MIDR bit is set, and the transaction is a write. When multiple master IDs are both matched and enabled and one or more have MIDR bits set, a master ID read-only error is still flagged. The MID_RO bit of the ISR register is set.
Master ID list miss error	When all MATCH vector bits are zero, a master ID miss error occurs. The MID_MISS bit of the ISR register is set.
Aperture permission list parity error	The transaction is disallowed and APER_PARITY bit of the ISR register is set. An interrupt can optionally be signaled.
Transaction TrustZone error[2]	When a non-secure transaction attempts to access a secure slave, a transaction TrustZone error occurs. This error is flagged only when there is no MID_MISS error and no APER_PARITY error. This error is not flagged when there is a MID_MISS error or an APER_PARITY error. The transaction fails and an interrupt can optionally be signaled.
Transaction permission error[2]	When a master ID is not allowed to access a slave, a transaction permission error occurs. An access to an address not covered by the XPPU causes this type of error. A burst length/size error (when accessing 32B buffers) also causes this type of error to occur. This error is flagged only when there is no MID_MISS error and no APER_PARITY error. This error is not flagged when there is a MID_MISS error or an APER_PARITY error. The transaction fails. An interrupt can optionally be signaled.

Notes:

1. Access to an address not covered by the aperture permission registers goes through the XPPU intact.
2. The first transaction address, master ID, and read/write mode are captured for debugging. When there are simultaneous read/write errors, only the write error is recorded. Only the first error is recorded. To record further errors, the ISR (interrupt status register) must be cleared first.

Configuration

The XMPU is configurable either one time or through TZ accesses. At boot time, the XMPU can be configured and its configuration is locked so that it can only be reconfigured at the next system reset. If the configuration is not locked, the XMPU can be reconfigured any number of times by TZ accesses.

For dynamic reconfiguration, the software can directly configure the registers.

Security Lock

Before the lock bit is set, configuration can be changed by TZ secure accesses. After the lock bit is set, the configuration cannot be changed, even by secure accesses.

The software can chose to lockdown the values in the XMPU registers so they cannot be changed. Access can be reenabled after a POR.

For dynamic reconfiguration, the XPPU lock must not be asserted.

The lock bit does not prevent the interrupt status registers from being cleared, i.e., the ISR registers are not included in the lock. The accesses to clear the interrupts still need to be TZ secure. All other interrupt registers (IEN, IDS) are protected under LOCK. Only the ISR can be modified when the lock is set.

Master ID Validation

Each XPPU also uses the Master ID in each AXI transaction to validate the transaction. The Master ID is masked by the [MIDM] bit field and then compared against the [MID] bit field in the MASTER_IDxx registers. If the following equation is satisfied (along with [TRUSTZONE] and [PERMISSION] checks in the APERPERM_xxx register), then the transaction is allowed. In this equation, these are [10-bit parameters] in the MASTER_IDxx register.

$$[\text{MID}] \ \& \ [\text{MIDM}] == \text{AXI\_MasterID} \ \& \ [\text{MIDM}]$$

PS-PL Interfaces

There are several AMBA® interface channels between the PL and PS.

Slave interface ports in the PL connect to masters in the PS:

- S_ACE_FPD: AXI coherency extension (ACE)
- S_ACP_FPD: accelerator coherency port (ACP)
- S_AXI_HPC: AXI4-Lite coherent port to SMMU and CCI
- S_AXI_HP: AXI port to FPD main switch
- S_AXI_LPD: AXI port to LPD main switch

Master interface ports in the PL are driven by port slaves in the PMC:

- M_AXI_FPD: AXI port from FPD main switch to PL
- M_AXI_LPD: AXI port from LPD main switch to PL

PL to PS AMBA Interfaces

These interfaces are shown in PS Interconnect Diagram.

Summary Table

Address Translation and Coherency

The memory protection and coherency features of the interfaces are shown in the following table.

Table 86: **PL to PS AMBA Interfaces**

Interface Name	Alternate Name	Address Translation Unit	APU L2 Cache Coherency	Description
S_ACE_FPD		~	Two-way	PL to FPD CCI with two-way coherency
S_ACP_FPD	Summary TableAddress Translation and Coherency	~		PL to APU MPCore coherency
S_AXI_HPC	AFIFM2	SMMU TBU2	I/O coherency	PL to FPD AXI
S_AXI_HP	AFIFM0, S_AXI_FPD	SMMU TBU5	~	PL to FPD AXI
S_AXI_LPD	AFIFM4	Done in PL	~	PL to LPD AXI

ACE Interface

The APU cache-coherent interconnect (CCI) includes an AXI coherency extension (ACE) to the PL (S_ACE_FPD). This interface is an extension to the AXI protocol with two-way coherency between the APU L2-cache and a cache subsystem in the PL.

Coherency is further discussed in the Cache Coherent Interconnect chapter.

ACP Interface

The accelerator coherency port (ACP) is a 128-bit AXI interface to enable coherent transactions from the PL to snoop the APU L2 cache that includes write allocation into the APU L2 cache. The ACP does this by connecting to the snoop control unit (SCU) inside the APU MPCore.

The ACP interface is generally used for legacy applications but has other, general applications. An alternative is the full ACE interface (S_ACE_FPD) with two-way coherency. Or, the ACE-lite interface (S_HPC_FPD) with I/O coherency. The ACE and ACE-lite interfaces are routed to the SMMU and then CCI.

PL coherent masters can use the ACP interface to access the APU L2 cache and the memory subsystem in the same way the APU processors do. This hardware coherency mechanism simplifies software and can increase overall system performance in appropriate applications. From a system perspective, the ACP interface competes with APU CPUs for access to the L2 cache memory.

Note: All ACP transactions are considered coherent to the APU L1 data cache and L2 unified cache. There is no option to mark a transaction as non-coherent through the side band signals (AxUSER and AxCACHE).

Any read transactions through the ACP to a coherent region of memory interact with the SCU to check whether the required information is stored within the processor L1 data caches. If it is, the data is returned directly to the requesting component. If it misses in the L1 data cache, then there is also the opportunity to hit in L2 cache before finally being forwarded to the DDR memory system. For write transactions to any coherent memory region, the SCU enforces coherency before the write is forwarded to the DDR memory system. The transaction can also optionally allocate into the L2 cache, removing the power and performance impact of writing through to the DDR memory.

The ACP interface cannot be routed through either the APU MMU or the SMMU. Consequently, the ACP interface drives a 44-bit physical address.

ACP Limitations

The ACP accepts only the following (cache-line friendly) transactions.

- 64-byte aligned (of 64-byte) read/write INCR transactions. All write-byte strobes must be the same for all beats (either enabled or disabled). AxLEN must be `0x03` (four beats).
- 16-byte aligned (of 16-byte) read/write INCR transactions. Write-byte strobes can have any value. AxLEN must be `0x00` (one beat).
- ARCACHE and AWCACHE are restricted to the values `0b0111`, `0b1011`, and `0b1111`.
- The value of `0b11` for AxUSER[1:0] is not allowed, other values (`0b00`, `0b01`, `0b10`) are allowed.

For further details, see the Arm® *Cortex™-A72 MPCore Processor Technical Reference Manual*.

The ACP interface supports up to four outstanding transactions. These can be any combination of reads and writes. However, there can only be one outstanding transaction per AXI ID. The master must avoid sending more than one outstanding transaction on the same AXI ID to prevent the second transaction from stalling the interface until the first is complete.

ACP Usage

The ACP provides a low-latency path between the PS and the accelerators implemented in the PL when compared with a legacy cache flushing and loading scheme. The steps that must take place in an example of a PL-based accelerator are as follows:

1. The CPU prepares input data for the accelerator within its local cache space.
2. The CPU sends a message to the accelerator using one of the HPM AXI master interfaces to the PL.
3. The accelerator fetches the data through the ACP, processes the data, and returns the result through the ACP.

4. The accelerator sets a flag by writing to a known location to indicate that the data processing is complete. The status of this flag can be polled by the processor or can generate an interrupt.

When compared to a tightly-coupled coprocessor, ACP access latencies are relatively long. Consequently, the ACP is not recommended for fine-grained instruction level acceleration. Instead, for coarse-grain acceleration, such as video frame-level processing, the ACP does not have a clear advantage over traditional memory-mapped PL acceleration because the transaction overhead is small relative to the transaction time, and can potentially cause undesirable cache thrashing. Consequently, the ACP is optimal for medium-grain acceleration, such as a block-level crypto accelerator and video macro-block level processing.

The ACP supports limited throughput (four outstanding transactions), two transaction burst lengths (64-byte and 16-byte), and adversely affects CPU cluster performance (by treating all ACP transactions as coherent).

***Note*:** For the best power and performance, use either an S_AXI_HPC port or the ACE port to provide I/O coherency as the preferred approach over the ACP.

PS Traffic Architecture

The PS interconnect includes the LPD, FPD, and their connections to the PL. See the PS Interconnect Diagram for the hardware architecture overview.

Multiple CCI Master to NoC

There are four CCI masters connected to the NoC. There are four channels in the NoC structure along the southern part of the device. For a device perspective, see Physical Layout. Each CCI master is connected to one of the NoC channels. The southern NoC structure has four channels and services the PMC, PS, and DDRMC.

PS-PL Interfaces

The PS interconnect includes direct ways to connect the PS and PL, including:

- Direct AXI and ACE channels
- NoC pathways

AXI Transaction Attributes

The AXI transaction attributes include the data and address widths, and the AxCACHE, AxUSER, and AxPROT command signals that accompany an AXI interconnect transaction request as defined by the Arm AMBA AXI and ACE Protocol Specification.

The transaction attribute settings and options for each master are described in the following sections.

Attribute Types

System masters issue bus transactions with several types of attributes in addition to an address, and a read/write access type. These include the following:

***Note*:** Not all parameters apply to all masters. Some masters have fixed attributes for some bits while others have bits that can be configured statically or dynamically.

Burst Size

See Data Widths and Bursts for more information.

Cacheability

The cacheability of a transaction is specified in the AxCache bits and are driven by the master. These signal bits are used by the CCI to determine if a cache look-up is required and if there is a hit, what to do with the write data.

System Master IDs

Each PMC and PS master issues one or more unique system master IDs (SMID) to the interconnect. The SMID is transported on the AxUser[9:0] signals.

For some sources, a portion of the master ID bits are derived from an ID, a register setting, or by other means. The SMID[9] bit is always = 1 for PMC/PS masters. The bit is always = 0 for sources outside the PMC/PS. This flexibility and restriction allow for the enforcement of system-level protection on a per channel, per processor, or per non-PMC/PS basis.

The SMIDs for the PMC and PS are listed in System Master ID.

TrustZone Security Protection

Each transaction is marked as secure or non-secure (NS) using the AxProt[1] bit and are driven by the master. See TrustZone Security for more information.

Memory Space

The address spaces are described in:

- Memory Space

The address map tables are listed in Address Maps.

Data Widths and Bursts

The smallest data width is 32 bits. The largest transaction is a 128-bit data word with a burst of 256 data cycles (4 KB burst size).

- APB: Single 32-bit data word
- NPI: Burst of 32-bit data words
- AXI: Master can burst up to 256 words
- AXI: Slaves can typically receive up to 16 words

When an AXI master sends a 256-word burst to a slave that only supports 16-word transfers, the interconnect breaks-down the transaction into smaller bursts for the slave.

Quality of Service

The quality of service (QoS) signals provide information about the priority of the transaction.

The QoS signals flow through the system from a source attached to an AXI switch to various destinations.

QoS Sources

The source of the QoS signals.

PL to PS Interfaces

The QoS signals for three PL to PS interface can be statically configured using the register settings or dynamically set by input signals in the PL:

- S_HPC_FPD
- S_HP_FPD

- S_AXI_LPD

Transaction Traffic Types

Three different traffic types are defined:

- Low latency: typically CPU to DDR memory transactions
- Isochronous: real-time deadlines
- Best effort: bulk transfers and not time critical

Low Latency Transactions

For high-priority transactions, low latency is the key for meeting performance requirements. The APU and RPU MPCores can specify low latency transactions so their memory access requests are serviced in a timely manner to avoid or minimize CPU pipeline stalls.

Low latency transactions are usually associated with cache fill and replacement.

Video Isochronous Transactions

Another category of masters that can live with longer latency in typical conditions is isochronous (or video class) transactions. However, there is critical moment (maximum latency) in which data must be available without causing a system degradation. The key requirement is a guaranteed maximum latency. The example masters are video encoder, camera sensor, or display device.

Best Effort Transactions

High-throughput and best effort transactions are allowed with long latency but need high throughput to achieve architectural performance goals. These types of masters include DMAs, PCIe® with interconnect for a CCIX PCIe module (CPM), and PL functional units.

TrustZone Security

The Arm TrustZone technology tags the security level of each transaction. These are used by the protection units (XPPU and XMPU) and SMMU to restrict transaction access. The protection units work to support safety and security applications.

All AXI transactions are tagged to indicate their security level, and the tags are propagated throughout the interconnect using the ARPROT[1] and AWPROT[1] AXI signals.

Because TrustZone defines the security level of each AXI transaction, the system protection units can be used to allow or disallow a transaction based on its security level. Secure transactions can optionally access non-secure slaves, if allowed. Non-secure transactions cannot access secure locations.

Features

TrustZone technology provides a foundation for system-wide security and the creation of a trusted platform. The basic principle behind TrustZone technology is the isolation of all software and hardware states and resources into two worlds, trusted and not trusted.

A non-secure virtual processor can only access non-secure system resources, whereas, a secure virtual processor can see all resources. Resource access is extended to bus accesses using the NS flag which is mapped to the AxPROT[1] attribute on the AXI interconnect.

Any part of the system can be designed to be part of the secure world including debug, peripherals, interrupts, and memory. By creating a security subsystem, assets can be protected from software attacks and common hardware attacks.

Typical example TrustZone technology use cases include firmware protection, security management, and peripheral/IO protection. The TrustZone functionality is further described in TrustZone Security.

Architecture

The PS AXI interconnect propagates the AxPROT[1] bit to define the security of a transactions for secure or non-secure accesses. Strictly speaking, secure transactions are not allowed to access non-secure memory. Because the secure mode can issue secure or non-secure transactions, a secure transaction is allowed to access both secure and non-secure memory. The downside includes the potential of cross-contamination of software bugs because secure software can unintentionally corrupt non-secure memory.

In accordance with the recommendations of Arm's Trusted Base System Architecture specification, devices developed with TrustZone technology enable the delivery of platforms capable of supporting a full trusted execution environment (TEE) and security-aware applications and secure services, or trusted applications (TA). A TEE is a small secure kernel that is normally developed with standard APIs and developed to the TEE specification evolved by the Global Platform industry forum. See Arm References for more information.

TrustZone technology enables the development of a separate rich operating system and trusted environments by creating additional operating modes to the normal domain, known as the secure domain and the monitor mode. The secure domain has the same capabilities as the normal domain while operating in a separate memory space. The secure monitor acts as a virtual gatekeeper controlling migration between the domains.

The TrustZone technology forms the basis of a trusted secure environment for Arm systems. It enables a secure world (secure operating system) to be separated from a non-secure world (main operating system). TrustZone technology enables isolation between a secure and a non-secure world, which is enforced by hardware such that a non-secure world cannot access the resources in a secure world, but a secure world can access both secure and non-secure resources.

Master and Slave Security Profiles

Each system master provides a security setting with each AXI transaction. The AXI transactions pass through a protection unit to help maintain system integrity for security and safety applications. Profiles types include secure, non-secure (NS), programmable, and dynamic.

- Secure slaves prevent unauthorized access by non-secure masters
  - Slave security profiles for most peripherals are implemented by the XPPU and XMPUs
  - Access to several system control register sets must always be done by a secure master
- DDRMC, OCM, and XRAM memory can include secure and non-secure regions
  - Programmable on a per region basis (1 MB for DDRMC, 4 KB for OCM and XRAM)
  - Configurable using the respective XMPU protection units
- Several types of masters
  - Fixed type: secure or non-secure
  - Programmable: a register selects between secure and non-secure
  - Dynamic: master can change security levels on a per transaction basis, e.g., PS-PL AXI interfaces
- System boot assumes secure mode until FSBL reads the BootROM header
  - The processor system boots in secure mode
- RPU does not use TrustZone technology. Transactions from the RPU to the TrustZone environment of the APU can be configured as secure or non-secure
- The boot-time security level of the RPU is configurable, the default is to issue secure transactions

TrustZone Profile

The security profile for master and slaves are listed in the following table.

Table 87: **TrustZone Profile**

PS Entity	Slave Port	Master Port	Notes
APU			

Table 87: **TrustZone Profile** *(cont'd)*

PS Entity	Slave Port	Master Port	Notes
APU MPCore/L2	~	Both	
GIC	Both	~	Global interrupt controller (GIC)
APU system counter	Secure	~	System counter uses two APB ports (secure and non-secure)
APU system counter	Non-secure	~	
CCI			
CCI control registers	Both (internal)	~	Cache Coherent Interconnect (CCI) control registers can be configured to be secure or non-secure
SMMU			
TCU APB	Secure	~	SMMU_REG
TBU AXI	Both	Both	Programmable
XPPU, XMPU			
APB interface	Secure	~	
AXI interface	Both	Both	Programmable
LPD DMA Unit			
DMA channels	SLCR configurable	SLCR configurable	Programmable on a per channel basis
RPU			
RPU Cores	~	SLCR configurable	
RPU TCMs	XPPU configurable	~	External AXI slave port
LPD Peripherals and Slaves			
Secure SLCR	Secure	~	
PMC	Secure	Secure	
eFUSE/BPD/ PS_SYSMON	Secure	~	Fuses, battery power unit, SYSMON unit
CoreSight	Secure	Secure	
IOP peripherals	XPPU configurable	SLCR configurable	I2C, GPIO, SPI, GEM Ethernet, SD/eMMC, CAN, USB, UART, QSPI, OSPI
LP slave interfaces on APB	XPPU configurable	~	Potential secure slaves: reset-controller
TTC0 to 3	Configurable	~	
LPD_SWDT, FPD_SWDT		~	Watchdog timers
FPD Peripherals and Slaves			
Secure SLCR	Secure	~	
FP slaves APB	XPPU configurable	~	Potential secure slaves: reset-controller and PCIe
DDR Memory, OCM, and XRAM			
DDR Memory Controller	XMPU configurable	~	Secure/non-secure per region with 1 MB granularity
OCM	XMPU configurable	~	Secure/non-secure per region with 4 KB granularity

Table 87: **TrustZone Profile** *(cont'd)*

PS Entity	Slave Port	Master Port	Notes
Accelerator RAM (XRAM)		~	

Notes:

1. Secure: peripheral or memory device is always secure, independent of the condition.
2. Non-secure: peripheral or memory device is always non-secure, independent of the condition.
3. Configurable: peripheral or memory device could be configured as secure or non-secure but only one mode is allowed at any given time.
4. Both: part of the peripheral or memory device is secure while the other part is non-secure.

System Master ID

Each system master generates one or more 10-bit system master ID (SMID) number. The SMID is used by the SMMU, XMPU, XPPU, and IPI to identify a master to ensure it has the authority to access a slave.

Every PMC and PS master is assigned one or more unique SMID value starting with bit [9] = 1. Each non-PMC/PS master is assigned one or more unique number with bit [9] = 0. The SMID value is driven on the AxUSER channel. See PMC and PS SMID Table for an assignment list.

When a transaction is routed through a TBU in the SMMU, the 10-bit SMID value is concatenated with a 5-bit stream field that is unique to each TBU in the SMMU. See Stream IDs for assignments.

Features

Each unique SMID (one or more) per master is applied to both read and write transactions. The SMIDs provide the following characteristics and features:

- A master can have more than one SMID if it has different memory access threads that should be treated differently for system management purposes (e.g., multi-channel DMA).
- A master can have more than one SMID if it can support dynamic switching between two or more contexts.
- SMIDs traverse end-to-end over the NoC and AXI interconnect (soft and hard) via the AxUSER bits.
- PS masters are assigned a fixed set of SMIDs. The assignments for the PL masters is configurable.
- SMID [9] bit is hardwired and splits the masters into two groups:
  - 1: PS masters.

    - 0: Non-PS masters (including NoC, AI Engine, CPM, and PS-PL AXI slave interfaces).
- SMID values for PL IP are programmed by the PMC CFU using PL configuration frames.
- The non-PS masters routed to the NoC are assigned SMIDs by the PLM via the NPI programming interface.
- The non-PS masters NoC (including the PL to PS AXI master interfaces) are assigned SMIDs by CFRAMEs.
- A device can have dynamic context. In this case, the device is assigned a set of SMID values, and the appropriate device driver software has access permission to change the SMID within the allocated SMID set, but does not have access permission to change the SMID outside of the assigned set.
- SMID values are tied to permissions and security.
- SMID values are ECC protected as they traverse the interconnect.

Comparison to Previous Generation Xilinx Devices

In the Zynq® UltraScale+™ MPSoC, the master ID was either a fixed 10-bit value per master such as USB, SD, etc., or a combination of a fixed value and a subset of the master's AXI ID.

In the Versal device, a new type of unique device ID is defined, known as system master ID (SMID). The SMID replaces the Zynq UltraScale+ MPSoC master/stream ID. This provides several advantages:

- Flexibility
- Scalability
- ID assignment is not tied to the choice of PS input port
- ID is unique regardless of the path it takes via hard/soft interconnect or NoC
- ID is unique regardless of the physical location of the soft IP attachment to the NoC (i.e., support for relocatable partial reconfiguration)
- ID is preserved in cases of PL device virtualization/coherency (PL to NoC to PS to NoC to DDRMC)

System Perspective

At the top-level, the SMIDs are assigned as shown in the following table.

Table 88: **Global SMID Ranges**

Master	SMID		
	[9]	[8:7]	[6:0]
PMC masters	1	Reserved	See PMC and PS SMID Table
LPD masters	1		
FPD masters	1		
PL to PS AXI interfaces	0		Programmable
Non-PMC, non-PS masters	0	Programmable	Programmable

PMC and PS SMID Table

The system master IDs (SMID) are encoded in the AxUSER bits [9:0]:

- [9] always = 1 for PMC/PS masters
- [8:7] always = 00 as reserved for PMC/PS
- [6:0] bit values are assigned individual PMC/PS masters

Table 89: **System Management IDs**

Name	SMID [9:0]	Notes and Configuration Registers
PMC Masters		
RCU (BootROM)	10_0100_0110	
PPU (PLM software)	10_0100_0111	
DAP controller	10_0100_0000	
SYSMON	10_0100_0001	
SD/eMMC0 SD/eMMC1	10_0100_0010 10_0100_0011	
QSPI flash	10_0100_0100	
OSPI flash	10_0100_0101	
PMC_DMA0 PMC_DMA1	10_0100_1000 10_0100_1011	
HSDP_DPC	10_0100_1001	
LPD Masters		
RPU0 processor	10_0000_00xx	LPD_SLCR.RPU0_SMID_CFG
RPU1 processor	10_0000_01xx	LPD_SLCR.RPU1_SMID_CFG

Table 89: **System Management IDs** *(cont'd)*

Name		SMID [9:0]	Notes and Configuration Registers
LPD_DMA	Ch 0 Ch 1 Ch 2 Ch 3 Ch 4 Ch 5 Ch 6 Ch 7	10_0001_000x 10_0001_001x 10_0001_010x 10_0001_011x 10_0001_100x 10_0001_101x 10_0001_110x 10_0001_111x	LPD_SLCR.LPD_DMA_SMID_CFG
USB 2.0		10_0011_000x	LPD_IOU_SLCR.USB_SMID_CFG
GEM 0 Ethernet MAC GEM 1		10_0011_0100 10_0011_0101	
PSM		10_0011_1000	
HSDP_DMA		10_0011_1001	
FPD Masters			
APU MPCore		10_0110_xxxx	Bits [3:0] are determined by AXI_ID, see APU SMID [3:0]
SMMU		10_0111_0100	
APU GIC interrupt controller		10_0111_0010	
CoreSight		10_0111_0011	
PCIe		10_0101_0xxx	Bits [2:0] are determined by PCIe device ID
PL Masters			
Miscellaneous		0x_xxxx_xxxx	Bit [9] is always 0.

APU SMID [3:0]

APU SMID [3:0] bits are derived from the APU AXI ID, and are listed in the following table.

Table 90: **APU SIMD Bits [3:0]**

AXI_ID[6:5]	SIMD [3:0]
00	1000
01	1000
10	1000
11	0 | AXI_ID[2:0]

XILINX

Section IX

Interrupts and Errors

This section includes these chapters:

System Interrupts

The system interrupt sources provide flexibility for platform control, targeted actions and signaling events.

Most system blocks generate an interrupt to signal the completion of a task or to alert that an event occurred. For example, when a direct memory access (DMA) unit completes its transfer or an advanced peripheral bus (APB) programming interface detects an address decode error, a system interrupt can be generated.

In many cases, multiple interrupts are generated within a controller. The enabled interrupts are OR'd together to create a single system interrupt.

There are over 150 system interrupts. The system interrupts are routed to the PMC, PSM, RPU_GIC, APU_GIC, and PL.

Some common system interrupts include:

- I/O peripheral control interrupts
- Inter-processor interrupts (IPI)
- Timer interrupts
- Correctable and uncorrectable errors
- APB programming interface address decode errors

System Interrupt Controllers

There are several system interrupt controllers:

- RPU: GIC-390, v2 architecture
- APU: GIC-500, v3 architecture
- PMC: MicroBlaze™
- PSM: MicroBlaze
- PL fabric: controller instantiated for MicroBlaze or other PL-based processor

Interrupt Source Accumulators

The subsystem generates an interrupt when something occurs that software should know about. There are usually multiple reasons why an interrupt is generated. The local interrupt sources are controlled by a local interrupt accumulator. The OR'd result becomes a system interrupt.

IRQ System Interrupts

The system interrupts are generated by various subsystem units and are routed to the system interrupt controllers. The system interrupts are listed in the following table.

Table 91: **System Interrupts**

IRQ Name	IRQ Number (RPU, APU GIC)	GICPx_IRQ Bit (GIC Proxy)	Description
IRQ Status Register 0			
reserved	32:39	GICP0 [0:7]	reserved
RPU0_PERF_MON	40	GICP0 [8]	Performance monitor
RPU1_PERF_MON	41	GICP0 [9]	
OCM	42	GICP0 [10]	OCM error
RPU0_ERR	43	GICP0 [11]	Combined errors: FPU, memory ECC, and AXI slave access
RPU1_ERR	44	GICP0 [12]	
LPD_GPIO	45	GICP0 [13]	LPD GPIO controller
LPD_I2C0	46	GICP0 [14]	LPD I2C 0 controller
LPD_I2C1	47	GICP0 [15]	LPD I2C 1 controller
SPI0	48	GICP0 [16]	SPI 0 controller
SPI1	49	GICP0 [17]	SPI 1 controller
UART0	50	GICP0 [18]	UART 0 controller
UART1	51	GICP0 [19]	UART 1 controller
CANFD0	52	GICP0 [20]	CANFD 0 controller
CANFD1	53	GICP0 [21]	CANFD 1 controller
USB_2_XFER	54:57	GICP0 [22:25]	USB 2.0 controller bulk transfer, isochronous transfer, controller interrupt, control transfer
USB_2_CORE	58	GICP0 [26]	USB 2.0 controller
PMC_BUF_IPI	59	GICP0 [27]	OR of all IPIs targeted to PMC with message buffer
PMC_NOBUF_IPI	60	GICP0 [28]	OR of all IPIs targeted to PMC without message buffer
PSM_IPI	61	GICP0 [29]	OR of all IPIs targeted to PSM
IPI0	62	GICP0 [30]	IPI 0 interrupt
IPI1	63	GICP0 [31]	IPI 1 interrupt
IRQ Status Register 1			
IPI2	64	GICP1 [0]	IPI 2 interrupt

Table 91: **System Interrupts** *(cont'd)*

IRQ Name	IRQ Number (RPU, APU GIC)	GICPx_IRQ Bit (GIC Proxy)	Description
IPI3	65	GICP1 [1]	IPI 3 interrupt
IPI4	66	GICP1 [2]	IPI 4 interrupt
IPI5	67	GICP1 [3]	IPI 5 interrupt
IPI6	68	GICP1 [4]	IPI 6 interrupt
TTC0_CLK[0:2]	69:71	GICP1 [5:7]	TTC controller 0, timer/clocks 0 to 2
TTC1_CLK[0:2]	72:74	GICP1 [8:10]	TTC controller 1, timer/clocks 0 to 2
TTC2_CLK[0:2]	75:77	GICP1 [11:13]	TTC controller 2, timer/clocks 0 to 2
TTC3_CLK[0:2]	78:80	GICP1 [14:16]	TTC controller 3, timer/clocks 0 to 2
LPD_SWDT	81	GICP1 [17]	SWDT in LPD
PSM	82	GICP1 [18]	PSM interrupt
LPD_XPPU	83	GICP1 [19]	XPPU in LPD
LPD_INT	84	GICP1 [20]	OR of LPD interconnect masters and slaves
SYSMON	85	GICP1 [21]	SYSMON unit
reserved	86:87	GICP1 [22:23]	reserved
GEM0	88	GICP1 [24]	GEM controller 0
GEM0_Wakeup	89	GICP1 [25]	GEM controller 0 wake-up
GEM1	90	GICP1 [26]	GEM controller 1
GEM1_Wakeup	91	GICP1 [27]	GEM controller 1 wake-up
LPD_DMA[0:3]	92:95	GICP1 [28:31]	LPD DMA channels 0 to 3
IRQ Status Register 2			
LPD_DMA[4:7]	96:99	GICP2 [0:3]	LPD DMA channels 4 to 7
LPD_XMPU	100	GICP2 [4]	XMPU in LPD
LPD_SWDT_RSTPEND	101	GICP2 [5]	SWDT in LPD reset pending
LPD_SWDT_WS[0:1]	102:103	GICP2 [6:7]	SWDT in LPD WS 0 and 1
CPM	104	GICP2 [8]	OR of CPM interrupts and events
CPM_CE	105	GICP2 [9]	CPM interrupt 1, correctable error
USB_2_PME	106	GICP2 [10]	USB power management unit (PME) located in the PMC power domain
CPM_UE	107	GICP2 [11]	CPM interrupt 2, uncorrectable error
reserved	108:109	GICP2 [12:13]	reserved
XRAM	110	GICP2 [14]	Accelerator RAM controller
XRAM_CE	111	GICP2 [15]	Accelerator RAM correctable error
XRAM_UE	112	GICP2 [16]	Accelerator RAM uncorrectable error
reserved	113:115	GICP2 [17:19]	reserved
PL_PS_Group0_[0:7]	116:123	GICP2 [20:27]	PL_IRQ[0:7] to LPD
PL_PS_Group1_[0:3]	124:127	GICP2 [28:31]	PL_IRQ[8:11] to FPD
IRQ Status Register 3			
PL_PS_Group1_[4:7]	128:131	GICP3 [0:3]	PL_IRQ[12:15] to FPD

Table 91: **System Interrupts** *(cont'd)*

IRQ Name	IRQ Number (RPU, APU GIC)	GICPx_IRQ Bit (GIC Proxy)	Description
FPD_SWDT	132	GICP3 [4]	SWDT in FPD
reserved	133	GICP3 [5]	reserved
FPD_XMPU	134	GICP3 [6]	XMPU in FPD
APU_L2	135	GICP3 [7]	APU L2-cache double bit ECC error
EXT_ERR	136	GICP3 [8]	External error
APU processor	137	GICP3 [9]	APU interrupts
CCI	138	GICP3 [10]	FPD CCI
SMMU	139	GICP3 [11]	FPD SMMU
FPD_SWDT_WS0	140	GICP3 [12]	SWDT controller in FPD, WS0
FPD_SWDT_RSTPEND	141	GICP3 [13]	FPD_SWDT reset pending
FPD_SWDT_WS1	142	GICP3 [14]	SWDT controller in FPD, WS1
reserved	143:151	GICP3 [15:23]	reserved
CFU	152	GICP3 [24]	Configuration frames unit
reserved	153	GICP3 [25]	reserved
PMC_GPIO	154	GICP3 [26]	PMC GPIO controller
PMC_I2C	155	GICP3 [27]	PMC I2C controller
OSPI	156	GICP3 [28]	OSPI controller
QSPI	157	GICP3 [29]	QSPI controller
SD/eMMC0	158	GICP3 [30]	SD/eMMC controller 0
SD/eMMC0_Wakeup	159	GICP3 [31]	SD controller 0 wake-up
IRQ Status Register 4			
SD/eMMC1	160	GICP4[0]	SD/eMMC controller 1
SD/eMMC1_Wakeup	161	GICP4[1]	SD controller 1 wake-up
reserved	162	GICP4[2]	reserved
PMC_DMA0	163	GICP4[3]	PMC DMA 0
PMC_DMA1	164	GICP4[4]	PMC DMA 1
PMC_AXI	165	GICP4[5]	OR of PMC interconnect masters and slaves
PMC_XPPU	166	GICP4[6]	PMC XPPU
PMC_XMPU	167	GICP4[7]	PMC XMPU
SBI	168	GICP4[8]	SMAP bus interface
AES	169	GICP4[9]	AES
RSA	170	GICP4[10]	ECDSA RSA
EFUSE	171	GICP4[11]	eFUSE
SHA	172	GICP4[12]	SHA
TRNG	173	GICP4[13]	True random number generator
RTC_Alarm	174	GICP4[14]	RTC alarm
RTC_Seconds	175	GICP4[15]	RTC seconds
SYSMON	176	GICP4[16]	Voltage and temperature system monitor

Table 91: **System Interrupts** *(cont'd)*

IRQ Name	IRQ Number (RPU, APU GIC)	GICPx_IRQ Bit (GIC Proxy)	Description
reserved	177	GICP4[17]	reserved
NPI_IRQ0	178	GICP4[18]	NPI interrupt 0, DDRMC_MB all correctable software errors and interrupts
NPI_IRQ2	179	GICP4[19]	NPI interrupt 2, DDRMC_MC all correctable errors
NPI_IRQ5	180	GICP4[20]	NPI interrupt 5, AI Engine all correctable errors and miscellaneous events
NPI_IRQ6	181	GICP4[21]	NPI interrupt 6, AI Engine debug events and miscellaneous events
NPI_IRQ7	182	GICP4[22]	NPI interrupt 7, AI Engine miscellaneous events
NPI_IRQ8	183	GICP4[23]	NPI interrupt 8, GT interrupts and requests
NPI_IRQ9	184	GICP4[24]	NPI interrupt 9, GT all correctable errors
reserved	185	GICP4[25]	reserved
NPI_IRQ20	186	GICP4[26]	NPI interrupt 20, NoC user interrupts and errors
NPI_IRQ21	187	GICP4[27]	NPI interrupt 21, NoC user interrupts and errors
NPI_IRQ22	188	GICP4[28]	NPI interrupt 22, NoC user interrupts and errors
NPI_IRQ23	189	GICP4[29]	NPI interrupt 23, NoC user interrupts and errors
PMC RAM	190	GICP4[30]	PMC RAM
reserved	191	GICP4[31]	reserved

Register Reference

There are several sets of system interrupt masking registers.

There are multiple interrupt controller types receiving the system interrupts.

Interrupt Masking Registers

The system interrupts are distributed to the destinations listed in the table.

Table 92: **System Interrupt Masking Registers**

Destination	Controller	ISR and IMR	Programming Model
PMC	GIC proxy	PMC_GLOBAL	PPU MicroBlaze™
PSM	GIC proxy	PSM_GLOBAL	PSM MicroBlaze
RPU	Arm® GIC-390		Arm v2 architecture
APU	Arm GIC-500		Arm v3 architecture
PL	Output signal	None	~

Inter-Processor Interrupts

The inter-processor interrupts (IPI) enable one processor (source agent) to interrupt another processor (destination agent). The source agent optionally writes to a request message buffer and the destination agent optionally writes to a response message buffer. The communications process uses the IPI interrupt register structure, the system interrupt structure, and the IPI message buffers.

There are a maximum of ten agents and eight sets of message buffers; two of the agents do not have message passing buffers. Three of the agents are hardwired: PSM, PMC, and PMC_NOBUF. The other agents are assigned by their system master IDs (SMID).

In a typical situation, the source agent writes a 32-byte request message and then triggers an interrupt to the destination agent. The destination agent reads the request message and, optionally writes a response message. There are eight sets of 32-byte message/response buffers for each agent (16 total buffers per agent) for a total of 128 IPI message buffers. Each source-destination pair must establish their own message-passing communication protocols. These message buffers are access protected by the LPD_XPPU protection unit and IPI logic.

When the interrupt is serviced, the destination agent clears its status interrupt bit. This bit is observed by the source agent's observation register. This is an accumulation of the status interrupt bits from each of the destination agents. The source agent processor can have more than one active outstanding interrupt and message passing activity. The IPI interrupt registers are access protected by the LPD_XPPU protection unit and IPI registers.

Features

The IPI features include:

- Cross-processor communication interrupts with message passing
  - Source agent (a processor)
  - Destination agent (a processor)
- Up to 10 agents:
  - Three hardwired assignments: PSM, PMC, and PMC_NoBuf
  - Seven programmable agents assigned to a system processor

- Agent assignments:
  - Master ID tags in IPI registers for message and interrupt register access
  - System interrupt routing
- IPI agent interrupt registers:
  - Protected by 64 KB apertures in LPD_XPPU
- Message buffers are protected by hardware, and aperture controls using SMID
  - 64 request message buffers, 32 bytes each
  - 64 response message buffers, 32 bytes each

Comparison to Previous Generation Xilinx Devices

The Versal™ ACAP inter-processor interrupt mechanism is similar to the one in the Zynq® UltraScale+™ MPSoC with several differences as shown in the following table.

Table 93: **IPI Comparison to Previous Generation Xilinx Devices**

Feature	Zynq UltraScale+ MPSoC	Versal ACAP
IPI register set access control	IPI provides protection to interrupt registers	XPPU controls the read/write access of different masters
Message buffer protection	IPI provides protection (0xFF3F_0000)	XPPU provides protection (0xFF99_0000)
Message buffer programming	Read/write access hardcoded	Fully programmable and in software control
Permission setting	Permissions are hardcoded in hardware	XPPU had permission RAM entries for 128 32B apertures
Message buffer	Within the IPI	In XPPU
Lock feature	Lock feature added	Not present

System Perspective

System Management IDs

The IPI uses a processor's SMID to match it with an IPI agent. This includes three hardwired agent slots (PSM, PMC, and PMC_NOBUF) and the seven programmed slots (IPI 0 to 6). The SMID provides information to the LPD_XPPU protection unit and IPI controller.

System Interrupts

In addition to using SMIDs, the processors must be able to receive and process a system interrupt. The system interrupts for the PMC and PS are described in System Interrupts. Processors in the PL need to have SMID assigned to them that are recognized by the protection mechanisms and must have access to the PS-PL system interrupt outputs.

A processor is also a source and a destination agent to itself.

Power Domain

The IPI is in the PS LPD power domain with system interrupt connections to the FPD, PMC, and PL.

System Errors

System errors include the following:

- APB programming interface address decode error
- Interrupt register and message buffer access violations
  - Detected in LPD_XPPU and IPI
  - Master system ID incorrect
  - Security level violation
  - Write violation

Agent Communications

The communications between the two agents depends on their prearranged protocol. The IPI provides a framework for this communications.

Source Agent Initiates Action

To generate an interrupt, the source agent writes a 1 to a bit in its trigger (TRIG) register that corresponds to the destination agent. The source agent can verify that the bit is set in the destination agent's status register by reading its own observation (OBS) register. However, it cannot determine if the interrupt is enabled to generate the IRQ interrupt signal to the destination processor without accessing its GIC interrupt controller.

Destination Agent Response

When an agent receives an IPI system interrupt, it reads its IPI interrupt status register (ISR) to determine the source agent. After servicing the interrupt, with or without a message response, the destination agent clears its ISR by writing a 1 to the bit. This clearing of the destination ISR bit can be detected by the source using its observation register. The destination agent can also issue an IPI system interrupt back to the source agent.

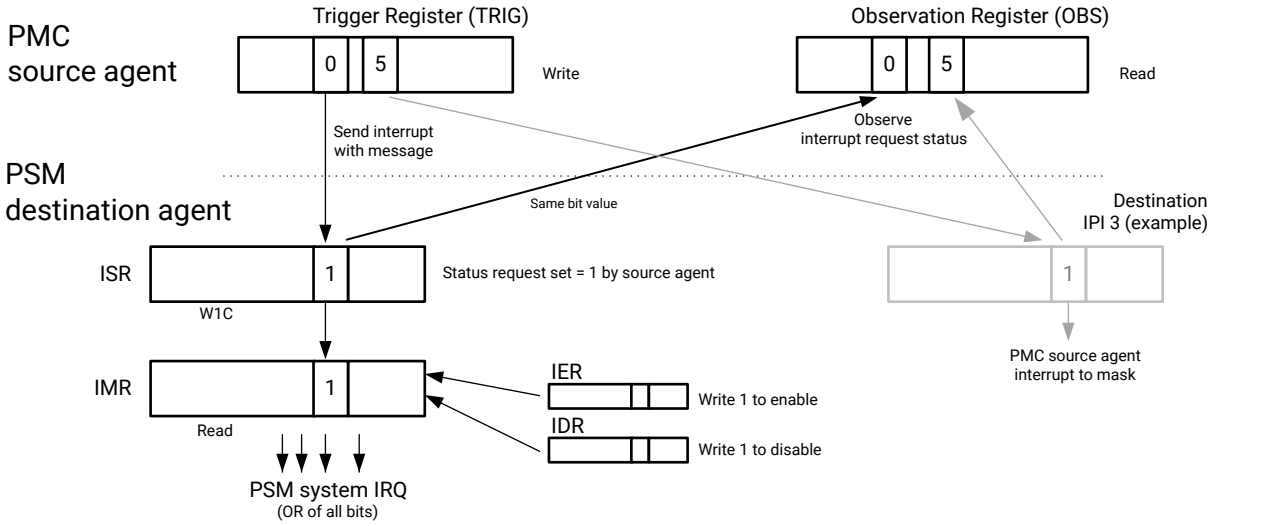
Interrupt Architecture

Interrupt Functionality

The interrupt architecture includes ten sets of registers with six registers per set. Each set is divided between sending an interrupt (TRIG and OBS) and receiving an interrupt (ISR, IMR, IER, and IDR); refer to Agent Interrupt Registers. Access to each set of interrupt registers is isolated to an agent by apertures in the LPD_XPPU protection unit followed by security screening by TrustZone apertures in the IPI (e.g., the IPI.TZ_APER_PSM register).

To send an interrupt, the source agent writes a 1 to the bit in its trigger register that corresponds to the desired destination agent processor. This causes the destination status register, ISR, bit to be set and generates a corresponding system interrupt. The source agent can observe the state of the interrupts that it has triggered to the destination agents using its observation register (OBS). The registers and signal routes are shown in the following figure.

Figure 59: **Source-Destination Interrupt Functions**



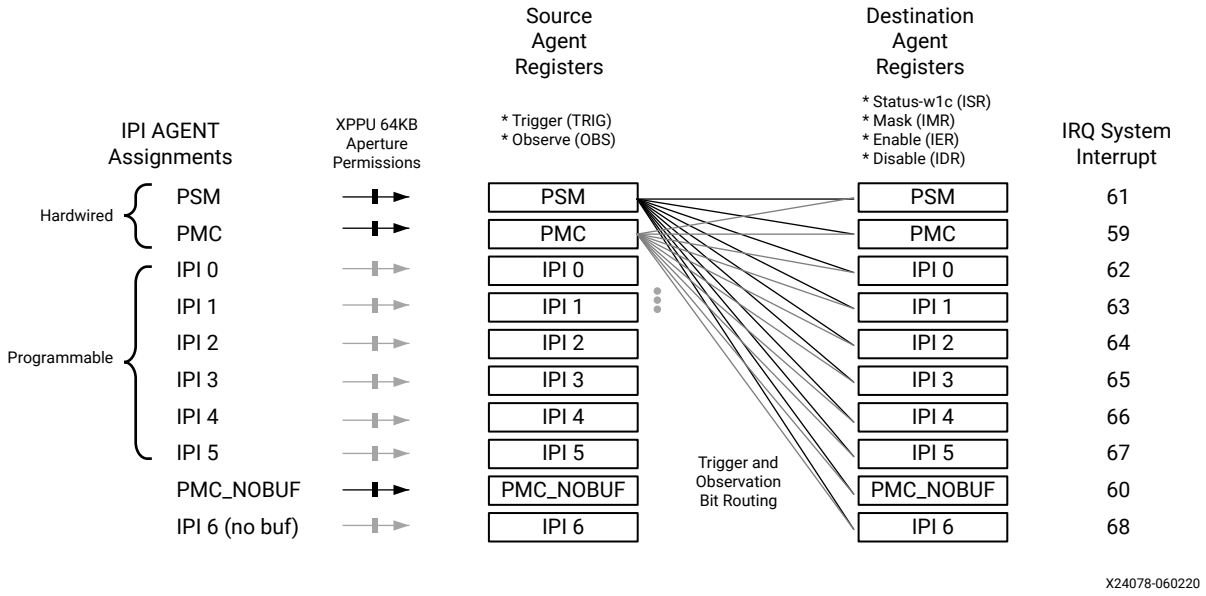
System Interrupt Registers

Software must program the system interrupt registers associated with the destination processor to enable the interrupt to propagate to the desired destination agent processor. This is one of the system interrupt controller registers (e.g., PMC_GLOBAL.GICP0_IRQ_MASK). All system interrupts are also routed directly to the PL. Refer to the System Interrupts chapter for the list of system interrupts. The destination agent processes interrupts in a normal manner; it can mask and clear its status register to control the system interrupt.

Interrupt Signal Mapping

Each interrupt channel has six registers. Two registers are for sending an interrupt and four registers are for receiving an interrupt. The trigger and observation registers are used to send and monitor interrupts. The status/clear, mask, disable, and enable registers are used to receive an interrupt. There are ten sets of interrupt registers. The hardwired and programmable channel assignments are shown in the following figure.

Figure 60: **IPI Interrupt Channel Architecture**



Note: It is the responsibility of the individual processors to mask unwanted IPI system interrupts in their GIC interrupt controller. These controllers are listed in System Interrupts.

Message Passing Architecture

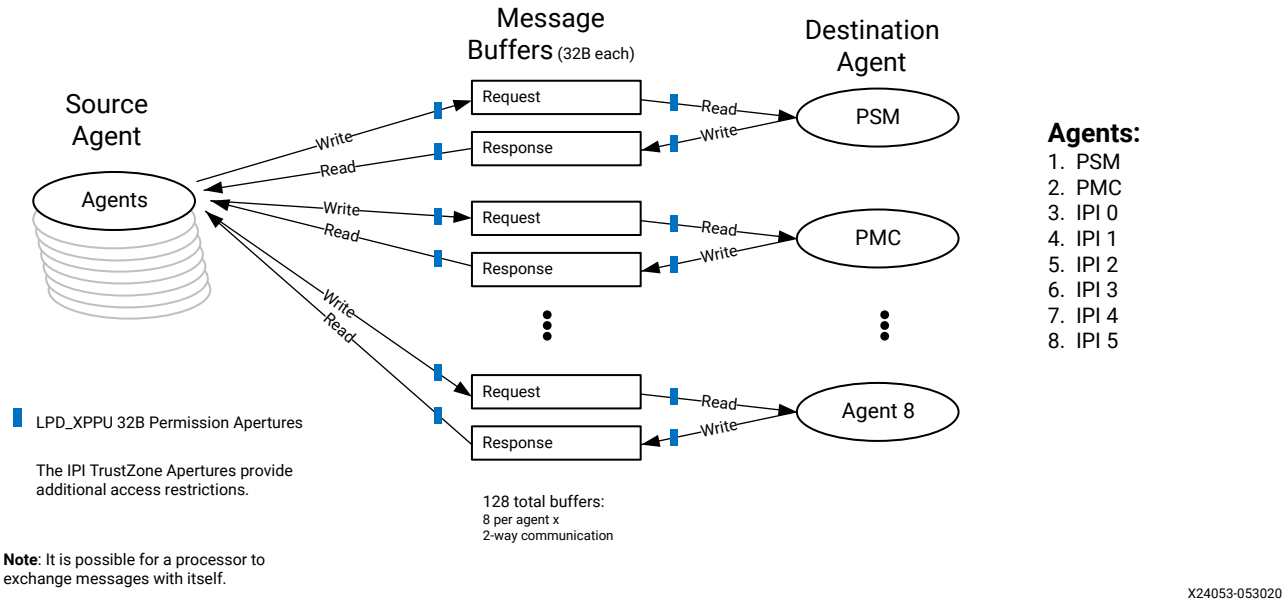
The messaging system connects eight agents together in a mesh configuration. The message passing between agents can be done exclusively between the sources and destinations by programming the 128 permission apertures in the LPD_XPPU that correspond to the `0xFF30_0000` to `0xFF30_01FF` memory range.

The IPI does not control the content of the message buffers. It is up to the source and destination processor agent software to define the back-and-forth interrupt signaling and the content put into the request and response message buffers. The content of the message buffers does not affect the hardware; it is only written and interpreted by the processors. The use of the message buffers is optional.

Messaging Diagram

The following figure illustrates the IPI message passing architecture for an agent.

Figure 61: **IPI Message Passing Diagram**



Agent Example

The example in this section shows the message buffer address offsets and access types for the APU assigned as the IPI_2 agent. The other IPI_x agents can be assigned as needed; this includes additional PSM or PMC agents. All buffers are 32 bytes. The base address for the message buffers is `0xFF3F_0000`.

Table 94: **IPI 2 Message Buffer Example Assigned to APU**

Offset Address	Buffer Type	Source Agent		Destination Agent	
		Name	Access Type	Name	Access Type
0x0800	Request	APU Example (assigned to IPI_2)	RW	PSM	R
0x0820	Response		R		RW
0x0840	Request		RW	PMC	R
0x0860	Response		R		RW
0x0880	Request		RW	IPI_0	R
0x08A0	Response		R		RW
0x08C0	Request		RW	IPI_1	R
0x08E0	Response		R		RW
0x0900	Request		RW	APU this example	RW
0x0920	Response		RW		RW
0x0940	Request		RW	IPI_3	R
0x0960	Response		R		RW
0x0980	Request		RW	IPI_4	R
0x09A0	Response		R		RW
0x09C0	Request		RW	IPI_5	R
0x09E0	Response		R		RW

Register Reference and Address Map

The IPI address space is protected by the LPD_XPPU. Apertures 048 through 063 are used to validate software accesses to the IPI address space.

Table 95: **IPI Address Map**

Address Range	Register Table	Protection	Notes
Control and Configuration Registers			
0xFF30_0000	Control Registers	LPD_XPPU and IPI.LOCK register	
Interrupt Registers: Trigger, status, observation, and mask			
0xFF31_0000 0xFF32_0000 ... 0xFF3A_0000	Agent Interrupt Registers		Only the first 32 bytes of address space are used
0xFF3C_0000	Reserved		
Message Buffers: Request and response			

Table 95: **IPI Address Map** *(cont'd)*

Address Range	Register Table	Protection	Notes
0xFF3F_0000	Message Buffer	IPI	4 KB of address space: 128 message buffers (32 bytes each)

Control Registers

The IPI control registers are summarized in the following table. Access to the registers are controlled by the LPD_XPPU protection unit and the IPI Register Write Lock Bit.

Table 96: **IPI Control Registers**

Register Name	Offset Address	Access Type	Lockable	Description
APB_ERR_CTRL	0x0000	RW	Yes	APB address decode SLVERR error signal enable
IPI_ISR IPI_IMR IPI_IER IPI_IDR	0x0010 0x001C 0x0018 0x001C	R, W1C R W W	All except ISR	Access violation and ECC error interrupt status, mask, enable and disable
LOCK	0x0090	RWSO	NA	Locks write access to all IPI registers except the ISR
SAFETY_CHK	0x0030	RW	No	Safety check registers
PROT_ERR_STATUS_L PROT_ERR_STATUS_H PROT_ERR_STATUS_ID	0x0028 0x0038 0x003C	R	NA	Address and ID of errored transaction
MASTER_ID00 MASTER_ID01 MASTER_ID02 MASTER_ID03	0x0040+	R	NA	Master identificaiton for: PSM read/write PSM read-only PMC read/write PMC read-only
MASTER_ID04 MASTER_ID05 Etc. MASTER_ID19	0x0050+	RW	Yes	Master identification for software defined masters
ECC_CTRL	0x0094	RW	Yes	ECC control
ECC_CE_FFA ECC_CE_FFD ECC_CE_FFE	0x0098+	R	Yes	First failing address, data and ECC register access with correctable error
ECC_UE_FFA ECC_UE_FFD ECC_UE_FFE	0x00A4+	R	Yes	First failing address, data and ECC register access with un-correctable error
FI_CNTR FI_D FI_S	0x00B0+		Yes	Fault injection count, data, and syndrome

Table 96: **IPI Control Registers** *(cont'd)*

Register Name	Offset Address	Access Type	Lockable	Description
TZ_APER_PSM TZ_APER_PMC TZ_APER_IPI0 TZ_APER_IPI1 TZ_APER_IPI2 TZ_APER_IPI3 TZ_APER_IPI4 TZ_APER_IPI5	0x00BC+	RW	Yes	Source agent message buffer TrustZone security access settings: 0: secure access required 1: non-secure
TZ_APER_INTR	0x00DC	RW	Yes	Interrupt register security access settings for all agents

Register Write Lock Bit

The IPI registers can only be configured by a TrustZone secure transaction. The secure transaction is routed through the LPD_XPPU protection unit to make sure the master has access privileges before it is allowed to reach the IPI programming interface with its additional restrictions.

Writes to the IPI registers can be blocked by setting the IPI.LOCK [ReqWrDis] lock bit = 1. Once this bit is set, it can only be cleared by a POR.

After the lock bit is set, many of the registers can no longer be written to until a POR occurs. The lockability of the registers are shown in the Control Registers table.

Agent Interrupt Registers

The IPI interrupt registers are listed in the following table. The base address is 0x0FF30_0000. These registers have access restriction based on the processor's SMID and settings in the IPI.TZ_APER_INTR register.

The IPI processor interrupt management registers are not affected by the IPI register LOCK control register.

Table 97: **IPI Processor Interrupt Management Registers**

Register Name	Offset Address	Access Type	Description
PSM_TRIG PSM_OBS PSM_ISR PSM_IMR PSM_IER PSM_IDR	0x10000+	W R W1C R W W	PSM agent interrupt registers

Table 97: **IPI Processor Interrupt Management Registers** *(cont'd)*

Register Name	Offset Address	Access Type	Description
PMC_TRIG PMC_OBS PMC_ISR PMC_IMR PMC_IER PMC_IDR	0x20000+	W R W1C R W W	PMC agent interrupt registers
IPIx_TRIG IPIx_OBS IPIx_ISR IPIx_IMR IPIx_IER IPIx_IDR	IPI0: 0x30000+ IPI1: 0x40000+ IPI2: 0x50000+ IPI3: 0x60000+ IPI4: 0x70000+ IPI5: 0x80000+ IPI6: 0xA0000+	W R W1C R W W	Programmable agents for IPI interrupts and messaging Except, IPI 6 does not include message or response buffers
PMC_NOBUF_TRIG PMC_NOBUF_OBS PMC_NOBUF_ISR PMC_NOBUF_IMR PMC_NOBUF_IER PMC_NOBUF_IDR	0x90000+	W R W1C R W W	PMC agent interrupt registers without message and response buffers

Message Buffer

The base address for message buffers is 0xFF3F_0000.

The IPI message buffer address map is shown in the following table.

Table 98: **IPI Message Buffer Address Map**

Master Offset Address	Buffer Type	Size	Source Agent		Destination Agent	
			Name	Access Type	Name	Access Type
0x000 to 0x01FF	Request	32B	PSM	RW	PSM	RW
	Response	32B		RW		RW
	Request	32B		RW	PMC	R
	Response	32B		R		RW
	Request	32B		RW	IPI 0	R
	Response	32B		R		RW
	Request	32B		RW	IPI 1	R
	Response	32B		R		RW
	Request	32B		RW	IPI 2	R
	Response	32B		R		RW
	Request	32B		RW	IPI 3	R
	Response	32B		R		RW
	Request	32B		RW	IPI 4	R
	Response	32B		R		RW
	Request	32B		RW	IPI 5	R
	Response	32B		R		RW
0x0200 to 0x03FF	Requests and Responses	512B	PMC	RW, R	PSM, PMC, IPI0, IPI1, IPI2, IPI3, IPI4, IPI5	RW, R
0x0400 to 0x05FF	"	512B	IPI 0	RW, R	"	RW, R
0x0600 to 0x07FF	"	512B	IPI 1	RW, R	"	RW, R
0x0800 to 0x09FF	"	512B	IPI 2	RW, R	"	RW, R
0x0A00 to 0x0BFF	"	512B	IPI 3	RW, R	"	RW, R
0x0C00 to 0x0DFF	"	512B	IPI 4	RW, R	"	RW, R
0x0E00 to 0x0FFF	"	512B	IPI 5	RW, R	"	RW, R

Programming Examples

Two programming examples are provided in this section.

Send an IPI Communication

This section describes how a source agent sends an IPI communication message. The source agent initiates the communications.

1. Write a 32 byte request message into the appropriate destination message buffer.

2. Write a 1 in the destination trigger bit.
3. Optionally, verify that the interrupt is posted by reading the observation register.
4. Determine that the interrupt has been processed with one of the following steps. The protocol must be established between the two agents:
    a. Source agent polls its observation register until the destination status bit is cleared indicating that the destination agent has processed the interrupt.
    b. Receive another IPI interrupt from the destination agent.

Receive an IPI Communication

This section describes how a destination agent receives an IPI communication message. The destination agent accesses its IPI registers.

1. Prepare to receive a message request with one of the following steps.
    a. Enable the interrupt from the sender using the IPI mask register, IMR, and in the processor's system interrupt controller by accessing its GIC registers.
    b. Destination agent polls its IPI status register for bits being set.
2. When an interrupt is received, optionally write a 32-byte response into the appropriate message buffer.
3. Signal to the source agent that the interrupt has been processed with one of the following steps.
    a. Clear the destination IPI status register.
    b. Issue an IPI interrupt back to the source agent.

System Errors

Each system error is routed to the PMC or PSM error accumulators. The accumulators latch the system error signals from over 100 sources. A system error signal indicates that one or more serious problems has been detected in a controller, processor, memory, or other functional unit. The system errors are accumulated in PMC and PSM status registers.

There are many types of system errors:

- Correctable and uncorrectable ECC
- APB programming interface address decode
- Single event upset (SEU) detected
- RPU lock-step and common cause failures
- Power failures
- Security violations

Each system error is associated with either the PMC or LPD error accumulator, which means that each system error is routed to only one status register bit. The PMC has two status registers that are independent of the two status registers located in the LPD.

The system error status registers are sticky. They are cleared by software writing a 1 or by a POR.

There are seven sets of mask registers, four in the LPD and three in the PSM. The output from the PMC status registers is also routed to the JTAG controller error status register. Software programs the state of the mask registers by writing to interrupt enable and disable registers.

The error accumulators can generate several major system events, each with its own set of programmable mask registers as follows:

- PMC system error accumulator events and interrupts:
  - Internal POR
  - System reset, SRST
  - ERROR_OUT pin
  - PMC IRQ

- LPD system error accumulator PSM interrupts:
  - PSM IRQ handler (general)
  - PSM IRQ handler (correctable)
  - PSM IRQ handler (uncorrectable)

PL System Error Status Port Signals

The outputs of both system error accumulators are also routed to the PL.

JTAG System Error Status Register

For JTAG users, the accumulated system error status bits are readable via the ERROR_STATUS Register.

The system errors are included in the Error ID table in the *Versal ACAP System Software Developers Guide* (UG1304).

Error Sources

Each system hardware error can be from a single event or an OR of several events. Each block or subsystem that creates a system error stores the details of its reported error, which includes the source and characteristics of the system error. Software can also generate system errors.

Error Status Registers

All system error status registers can only be cleared by software or by a POR. A system reset does not clear the system error status or mask registers.

Other Types of Errors

To clarify error types in the Versal™ device, there are other types of errors, including:

- Boot errors (see BootROM Error Codes)
- PLM software errors (see *Versal ACAP System Software Developers Guide* (UG1304))

System Error Accumulators

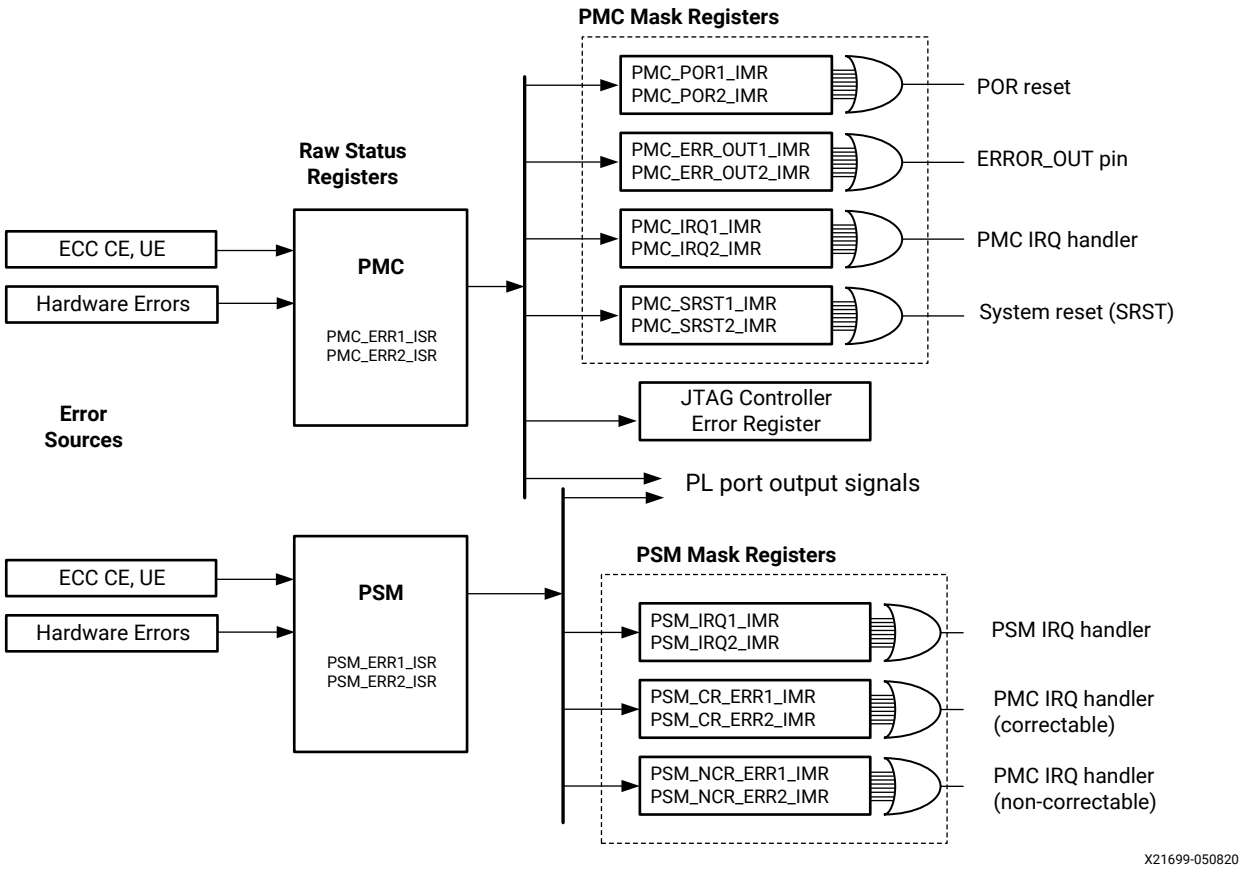
The system error accumulators are programmed to generate one or more system signal activities:

- PMC Error Status Accumulator Registers
  - Generate PMC IRQ interrupt to PLM

    - Assert ERROR_OUT output pin (PIN)
    - Assert system reset (SRST)
    - Assert internal POR
    - Most error in the JTAG controller error register, listed in ERROR_STATUS Register
- PSM Error Status Accumulator Registers
    - General IRQ handler in PSM firmware
    - Correctable IRQ handler in PSM firmware
    - Uncorrectable IRQ handler in PSM firmware
- System error signals are also routed to the PL

After the system error is latched, they are routed so they can generate one or more actions.

Figure 62: **System Error Signal Accumulation**



Functional Safety Errors

A safety error occurs when logic or a memory cell changes state due to a physical anomaly. The system can detect these anomalies. When a safety error occurs, it is important to ensure that the system remains in a safe state. This can include any of a number of actions. Broadly, responses fall into two categories.

- **Correctable Error:**

  A bit error is detected and corrected, usually by the hardware. The event is recorded and an interrupt is signaled.

  Note: The typical response is for the platform loader and manager (PLM) to report the event to the system safety software so it can be monitored and analyzed.

- **Uncorrectable Error:**

  An error that is detected but cannot be corrected. The event is recorded and an interrupt is signaled.

  Note: The typical response is for the PLM to indicate that a system-level intervention is required, which might include a partial or complete system reset.

Security Errors

A security error occurs when a secure asset is exposed. When a security error is detected, the system usually responds with a secure lockdown and zeroization of key system elements before a reset restart is issued.

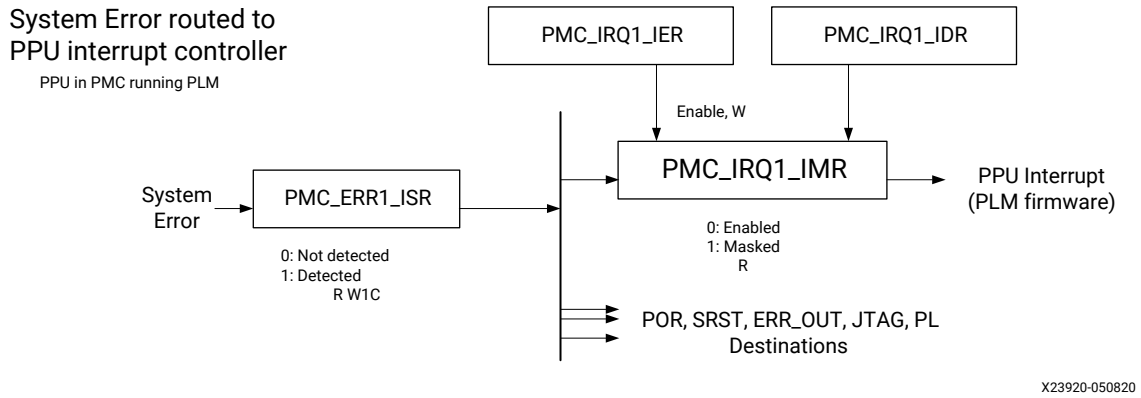
Programming Model

The PLM software and PSM firmware can store errors and other information in their respective storage registers. There are general software error registers in the PMC that are potentially accessible by all system processors including the PMC itself. For more information, see Error Accumulator Registers.

System Error Masking Example

The routing of each system error is controlled by mask registers. Each system error is normally routed to one of several destinations controlled by the mask registers as shown in the following figure.

Figure 63: **System Error Masking Example**



Programming Interface

The PMC error accumulator is programmed using a 32-bit APB programming interface.

Error Accumulator Registers

The system error accumulator registers are listed in the following table.

Table 99: **System Error Accumulator Registers**

Type	Register Names		Access Type	Description
	PMC Accumulator	PSM Accumulator		
Status	PMC_ERR1_ISR PMC_ERR2_ISR	PSM_ERR1_ISR PSM_ERR2_ISR	R	Raw status: 0: Deasserted 1: Asserted
Masks	PSM_OUT1_IMR PSM_OUT2_IMR PMC_POR1_IMR PMC_POR2_IMR PMC_POR1_IMR PMC_POR2_IMR PMC_POR1_IMR PMC_POR2_IMR PSM_IRQ1_IMR PSM_IRQ2_IMR PSM_SRST1_IMR PSM_SRST2_IMR	PSM_CR_ERR1_IMR PSM_CR_ERR2_IMR PSM_NCR_ERR1_IMR PSM_NCR_ERR2_IMR PSM_IRQ1_IMR PSM_IRQ2_IMR	R	Mask: 0: Enabled 1: Masked

Table 99: **System Error Accumulator Registers** *(cont'd)*

Type	Register Names		Access Type	Description
	PMC Accumulator	PSM Accumulator		
Enables	PSM_OUT1_IER PSM_OUT2_IER PMC_POR1_IER PMC_POR2_IER PSM_IRQ1_IER PSM_IRQ2_IER PSM_SRST1_IER PSM_SRST2_IER	PSM_CR_ERR1_IER PSM_CR_ERR2_IER PSM_NCR_ERR1_IER PSM_NCR_ERR2_IER PSM_IRQ1_IER PSM_IRQ2_IER	R	Enable: 0: Ignored 1: Enable error IMR set to 0
Disables	PSM_OUT1_IDR PSM_OUT2_IDR PMC_POR1_IDR PMC_POR2_IDR PSM_IRQ1_IDR PSM_IRQ2_IDR PSM_SRST1_IDR PSM_SRST2_IDR	PSM_CR_ERR1_IDR PSM_CR_ERR2_IDR PSM_NCR_ERR1_IDR PSM_NCR_ERR2_IDR PSM_IRQ1_IDR PSM_IRQ2_IDR	R	Disable: 0: Ignored 1: Disable error (IMR is set to 1)

PMC Error Status Accumulator Registers

The PMC error accumulation module includes two status registers:

- PMC_SYSTEM_ERR.PMC_ERR1_ISR; PMC Error Status 1
- PMC_SYSTEM_ERR.PMC_ERR2_ISR; PMC Error Status 2

PMC Error Status 1

The raw error status 1 bits in the PMC_SYSTEM_ERR.PMC_ERR1_ISR register are listed in the following table.

Table 100: **PMC System Error Accumulation Module Register 1**

Error Name	System Error Status Reg Bit	PLM Mask	JTAG Error Status Reg Bit	Description
reserved	0	0x000	63	reserved
BootROM NCR	1	0x001	62	BootROM non-correctable error; set during boot
PLM CR	2	0x002	61	PLM boot correctable error; set during boot
PLM NCR	3	0x003	60	PLM boot non-correctable error; set during boot
GSW CR	4	0x004	59	General software correctable error; set by any processor after boot

Table 100: **PMC System Error Accumulation Module Register 1** *(cont'd)*

Error Name	System Error Status Reg Bit	PLM Mask	JTAG Error Status Reg Bit	Description
GSW NCR	5	0x005	58	General software non-correctable error; set by any processor after boot
CFU	6	0x006	57	CFU error
CFRAME	7	0x007	56	CFRAME error
PSM CR	8	0x008	55	PSM correctable error
PSM NCR	9	0x009	54	PSM non-correctable error
DDRMC MB CR	10	0x00A	53	DDRMC MicroBlaze™ correctable ECC
DDRMC MB NCR	11	0x00B	52	DDRMC MicroBlaze non-correctable ECC
NOC CR	12	0x00C	51	NoC correctable error
NOC NCR	13	0x00D	50	NoC non-correctable error
NOC user	14	0x00E	49	NoC user error
MMCM lock	15	0x00F	48	MMCM lock error
AIE CR	16	0x10	47	AI Engine correctable error
AIE NCR	17	0x11	46	AI Engine non-correctable error
DDRMC MC ECC CR	18	0x12	45	DDRMC memory correctable ECC
DDRMC MC ECC NCR	19	0x13	44	DDRMC memory non-correctable ECC
GT CR	20	0x14	43	GT correctable error
GT NCR	21	0x15	42	GT non-correctable error
SYSMON CR	22	0x16	41	System monitor correctable error
SYSMON NCR	23	0x17	40	System monitor non-correctable error
User PL0	24	0x18	39	User-defined PL error
User PL1	25	0x19	38	User-defined PL error
User PL2	26	0x1A	37	User-defined PL error
User PL3	27	0x1B	36	User-defined PL error
NPI root	28	0x1C	35	NPI root error
reserved	29 to 31	0x1D to 0x1F	32 to 34	reserved

PMC Error Status 2

The raw error status 1 bits in the PMC_SYSTEM_ERR.PMC_ERR2_ISR register are listed in the following table.

Table 101: **PMC System Error Accumulation Module Register 2**

Error Name	System Error Status Reg Bit	PLM Mask	JTAG Error Status Reg Bit	Description
PMC APB	0	0x20	31	PMC APB programming interface address decode errors
PMC BootROM	1	0x21	30	BootROM validation error
RCU hardware	2	0x22	29	RCU hardware error
PPU hardware	3	0x23	28	PPU hardware error
PMC parity	4	0x24	27	PMC switch and IOP interconnect parity errors
PMC CR	5	0x25	26	PMC correctable errors
PMC NCR	6	0x26	25	PMC non-correctable errors
reserved	7 to 16	0x27 to 0x30	24 to 15	reserved
CFI NCR	17	0x31	14	CFI non-correctable error
SEU CRC	18	0x32	13	CFRAME SEU CRC error
SEU ECC	19	0x33	12	CFRAME SEU ECC error
reserved	20	0x34	11	reserved, returns 0
reserved	21	0x35	10	reserved, returns 1
RTC alarm	22	0x36	9	RTC alarm error
NPLL	23	0x37	8	PMC NPLL lock error; asserted while locking or when lock is lost
PPLL	24	0x38	7	PMC PPLL lock error; asserted while locking or when lock is lost
Clock monitor	25	0x39	6	Clock monitor errors
PMC timeout	26	0x3A	5	PMC interconnect timeout errors; from mission and timeout interrupt status registers
PMC XMPU	27	0x3B	4	PMC_XMPU error detection; includes read permission, write permission, and security violations
PMC XPPU	28	0x3C	3	PMC XPPU error detection; includes master ID not found, master ID parity error, read permission, master ID access, and TrustZone violations
reserved	29 to 31	0x3D to 03F		reserved

PSM Error Status Accumulator Registers

The PSM error accumulation module includes two status registers:

- PSM_SYSTEM_ERR.PSC_ERR1_ISR; PSM Error Status 1
- PSM_SYSTEM_ERR.PSC_ERR2_ISR; PSM Error Status 2

PSM Error Status 1

The raw error status 1 bits in the PSM_SYSTEM_ERR.PSM_ERR1_ISR register are listed in the following table.

Table 102: **PSM System Error Accumulation Register 1**

Error Name	System Error Reg Bit	PLM Mask	Description
PS_SW_CR	0	0x40	PS software write can set this bit
PS_SW_NCR	1	0x41	PS software write can set this bit
PSM_B_CR	2	0x42	PSM firmware write can set this bit
PSM_B_NCR	3	0x43	PSM firmware write can set this bit
MB_FATAL	4	0x44	OR of MicroBlaze fatal errors
PSM_CR	5	0x45	PSM correctable error
PSM_NCR	6	0x46	PSM non-correctable error
OCM_ECC	7	0x47	OCM ECC non-correctable error
L2_ECC	8	0x48	APU L2-cache ECC non-correctable error
RPU_ECC	9	0x49	OR of many errors
RPU_LS	10	0x4A	
RPU_CCF	11	0x4B	
GIC_AXI	12	0x4C	APU GIC access port
GIC_ECC	13	0x4D	APU GIC ECC non-correctable error
APLL_LOCK	14	0x4E	APU PLL lock error; asserted while locking or when looses lock
RPLL_LOCK	15	0x4F	RPU RPLL lock error; asserted while locking or when looses lock
CPM_CR	16	0x50	CPM correctable error
CPM_NCR	17	0x51	CPM non-correctable error
LPD_APB	18	0x52	LPD APB address decode errors: IPI, USB_2, CRL, S_AXI_LPD. LPD_IOP_SLCR, LPD_IOP_SECURE_SLCR
FPD_APB	19	0x53	FPD APB address decode errors: CRF, S_AXI_HP, S_AXI_HPC. FPD_SLCR, FPD_SECURE_SLCR
LPD_PAR	20	0x54	LPD AXI main interconnect parity error
FPD_PAR	21	0x55	FPD AXI main interconnect parity error
IOP_PAR	22	0x56	LPD IOP interconnect parity error
PSM_PAR	23	0x57	PSM interconnect parity error
LPD_TO	24	0x58	LPD interconnect timeout error
FPD_TO	25	0x59	FPD interconnect timeout error
PSM_TO	26	0x5A	PSM interconnect timeout error
XRAM_CR	27	0x5B	Accelerator RAM correctable error
XRAM_NCR	28	0x5C	Accelerator RAM non-correctable error
reserved	29 to 31	0x5D to 0x5F	reserved

PSM Error Status 2

The error status bits in the PSM_SYSTEM_ERR.PSM_ERR2_ISR register are listed in the following table.

Table 103: **PSM System Error Accumulation Register 2**

Error Name	System Error Reg Bit	PLM Mask	Description
LPD_SWDT	0	0x60	LPD system watchdog timer
FPD_SWDT	1	0x61	FPD system watchdog timer
reserved	2 to 17	0x62 to 0x71	reserved
LPD_MPU_ERR	18	0x72	LPD MPPU violations and errors
LPD_XPPU_ERR	19	0x73	LPD XPPU violations and errors
FPD_XMPU_ERR	20	0x74	FPD XPPU violations and errors
reserved	21 to 31	0x75 to 0x7F	reserved

Section X

Timers, Counters, and RTC

This section includes these chapters:

Summary of Counters and Timers

The following table is a summary of the system timers.

***Note*:** The xxx_LSBUS_CLK is the APB programming interface clock.

Table 104: **Summary of System Timers**

Name	Location	Time Base	Bits	Register Control	Usages
System counter	1x LPD	TS_REF_CLK	64	Memory-mapped registers in LPD and APU local-processor registers	System-wide physical count and virtual machine count using count offset System Counter
TTC	4x LPD	Selectable: - LPD_LSBUS_CLK - PS REF_CLK pin - RPU_REF_CLK	32	LPD_IOP_SLCR (clocking) TTC register set (config)	Triple timer counter for general purpose usage Triple-Timer Counter
SWDT	1x LPD	LPD_LSBUS_CLK		LPD and FPD SLCR (clocking) LPD and FPD SWDT (config)	System watchdog timer with windowing to define upper and lower expected response time
	1x FPD	FPD_LSBUS_CLK			
CoreSight™ debug counter					

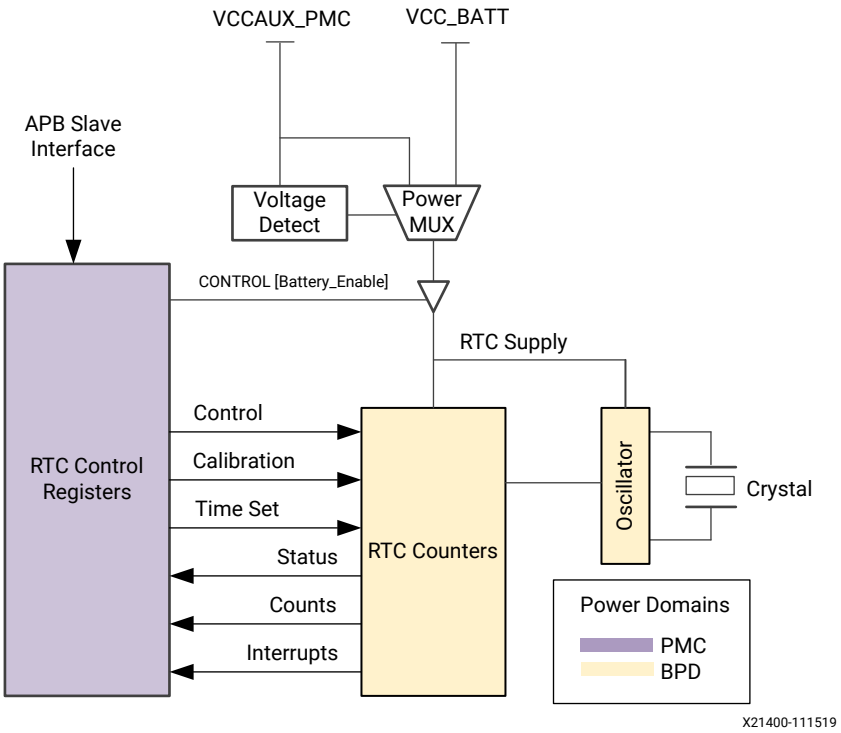
Real-Time Clock

The real-time clock (RTC) resides in the PMC and maintains an accurate time base for system and application software. It includes calibration circuitry to offset temperature and voltage fluctuations in applications requiring greater accuracy. The RTC also provides alarm setting and periodic interrupt features. The real-time clock provides continuous operation powered by the PMC auxiliary supply (VCCAUX_PMC) or the battery supply (VCC_BATT). When the auxiliary supply is available, the RTC uses it to keep the counters active. The RTC automatically switches to the battery power supply when the auxiliary supply is not available.

The RTC generates two system interrupt signals to the generic interrupt controller (GIC), the GIC proxy, and the programmable logic (PL) once every second and when an alarm event occurs. The periodic second tick interrupt can be used by all system processors. The alarm control must be managed at a system level with the processors.

As shown in the following figure, the RTC subsystem has three main modules: counter module, control register module, and oscillator module. The RTC counters module is powered by the battery power domain and includes three counters, calibration circuitry, and logic used to retain the programmed time. The RTC control register module is implemented in the PMC power domain and incorporates all of the registers associated with the RTC controller. The oscillator module is supplied by the battery power domain and provides the RTC clock.

Figure 64: **RTC Controller Block Diagram**



Features

The RTC has the following features:

- Continuous operation using auxiliary or battery power supplies
- Alarm setting and periodic interrupts
- Complex calibration circuits for highly accurate time keeping
- 32-bit seconds counter represents 136 years of time
- Three counters:
  - x 32-bit seconds counter
  - x 16-bit tick counter to measure a second based on 32 kHz crystal
  - x 4-bit fractional counter for calibration

Counter Module

The RTC counter module contains the 32-bit seconds counter, 16-bit tick counter, and 4-bit calibration counter. The counter module maintains a previously programmed time for read back and calibration by software and maintains the current time in seconds. The counter module calibration circuitry is used to calculate one second with a maximum PPM inaccuracy.

The seconds counter is a 32-bit synchronous counter that holds the number of seconds from a specific reference point known by the operating system. The seconds counter can represent a time of up to 136 years. Initially, the current time is calculated through the clock device driver in the operating system, which is based on the number of seconds that elapse from a reference point. This current time value is programmed into the RTC counters through the time-set register used to initialize the seconds counter. The seconds counter is then clocked every second to increment and hold the updated current time. The current time is read through the interface to the RTC controller.

For every oscillator clock cycle, the value in the tick counter is compared against the value stored in the calibration register. If these values match, the tick counter is reset to zero and an interrupt is generated. The interrupt signal from the RTC counters is asserted for one RTC clock cycle and is captured on the positive-edge transition of the interrupt status register RTC controller. The follow-on interrupt from the RTC counters can be used by a clock device driver to calculate the time and date.

When enabled, the fractional calibration feature takes effect every 16 seconds and delays the release of the clear signal to the tick counter by the number of oscillator cycles programmed in the fractional calibration field of the calibration register.

Calibration

The clear signal that is used to reset the tick counter can be extended/delayed by logic that operates with the fractional calibration value to provide fractional tick adjustment. Every time the fraction counter asserts an extend clear signal to the tick counter, the clear function to the tick counter remains asserted. Any inaccuracy in the oscillator is compensated for by adjusting the calibration value and making the remaining inaccuracy a fraction of a tick in every second. The impact of the remaining inaccuracy can be compensated for by using a fraction counter.

Every 16 seconds the accumulated inaccuracy can be approximated by the total number of ticks between zero and 16. This value is programmed in the fractional calibration segment of the calibration register. After 16 seconds, the fraction counter starts incrementing from zero to this value. During the time the fraction counter is incrementing, the clear signal to the tick counter stays asserted. As a result, the tick counter increments are delayed by the value of ticks every 16 seconds.

When the fraction comparator determines that the fraction counter value is equal to the maximum fractional calibration value, the fraction comparator releases the clear signal of the tick counter. This clear signal allows the fractional counter to start incrementing again. The fractional calibration register also includes an enable bit. When this bit is a 1, the fraction comparator performs the operations associated with fractional calibration, including the tick counter extend clear signal.

RTC Accuracy

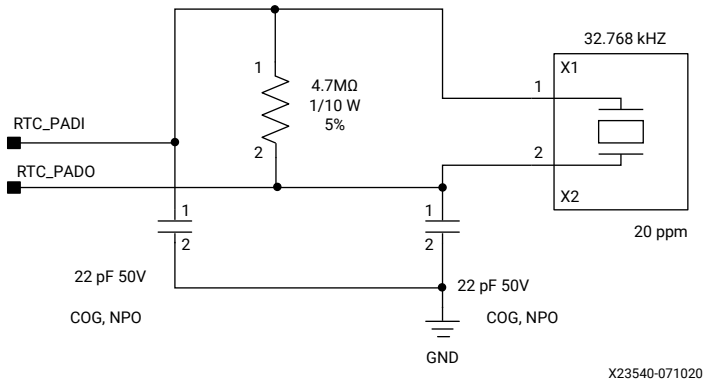
For the 32.768 kHz crystal oscillator, the static inaccuracy of the RTC is bounded to ±30.5 ppm if the selected crystal has a larger static inaccuracy. For example, a crystal inaccuracy of +50 ppm in one-million ticks generates 50 extra ticks (or off by 1–9/16 of a tick every second). Increasing the calibration value by one leaves 9/16 of the tick. Therefore, the static +50 ppm crystal in accuracy impacts the RTC similar to a +17.17 ppm crystal, because some of the inaccuracy is accounted for through the seconds calibration.

By enabling the fractional calibration feature, the second calculation logic can perform further calibration by delaying the clearing of the tick counter by one to 15 oscillator ticks every 16 seconds. In the previous example, after every 16 seconds, the clock is nine ticks ahead. Therefore, by programming the value of nine into the fractional calibration field of the calibration register, the time is adjusted by nine ticks every 16 cycles, which corrects the static inaccuracy of the oscillator.

External Clock Crystal and Circuitry

The typical crystal used for the RTC is a 20 ppm, 32.768 kHz crystal (see the following figure). Using the RTC calibration mechanism, the effective inaccuracy is reduced to less than two. Using a 65.536 kHz crystal further reduces the effective calibration inaccuracy to less than 1 ppm.

Figure 65: **Crystal Circuit Example**



Interfaces and Signals

The RTC interfaces to logic in the PMC and includes these features:

- An APB interface to access the registers within the controller and the RTC counters
- Interrupt status, interrupt mask, interrupt enable, and interrupt disable registers manage the seconds and alarm interrupts
- The RTC control register enables the crystal oscillator, controls power to the RTC, and enables address errors when accesses are made to the regions within the RTC address space that are not mapped to registers

IMPORTANT! *The RTC control register must be programmed every time the PMC power domain is powered. Otherwise, the value returned by reading the control register can be different from the actual control settings stored in the battery power domain.*

Registers

The RTC control and status registers are listed in the following table.

Table 105: **RTC Register Summary**

Register Name	Width	R/W Type	Reset Value	Description
SET_TIME_WRITE	32	WO	0h	Program the RTC with the current time
SET_TIME_READ	32	RO	0h	Read the last write done by software to SET_TIME_WRITE
CALIB_WRITE	21	WO	0h	Store the value that is used to generate one second based on the oscillator period
CALIB_READ	21	RO	0h	Read back the calibration value that was programmed in the RTC
CURRENT_TIME	32	RO	0h	32-bit timer value in seconds
ALARM	32	RW	0h	Program the alarm value for the RTC
RTC_INT_STATUS	2	WTC	0h	Raw interrupt status
RTC_INT_MASK	2	RO	3h	Interrupt mask applied to the status
RTC_INT_EN	2	WO	0h	Write a 1 to enable an interrupt
RTC_INT_DIS	2	WO	0h	Write a 1 to disable an interrupt
ADDR_ERROR	1	WTC	0h	Register address decode error interrupt status
ADDR_ERROR_INT_MASK	1	RO	1h	Register address decode error interrupt mask
ADDR_ERROR_INT_EN	1	WO	0h	Write a 1 to enable address decode error interrupt
ADDR_ERROR_INT_DIS	1	WO	0h	Write a 1 to disable address decode interrupt

Table 105: **RTC Register Summary** *(cont'd)*

Register Name	Width	R/W Type	Reset Value	Description
CONTROL	32	RW	0200_0000h	Controls the battery enable, clock crystal enable, and APD address decode error
SAFETY_CHK	32	RW	0h	Endpoint connectivity safety check

System Counter

The system counter is used by software to acquire a time stamp that is accessible to all processors. The APU can access the time count using a CPU local register. The other processors access the LPD_SCNTR register module. The system counter is physically located in the LPD.

The counter is clocked by the TS_REF_CLK from the LPD clock controller. This reference clock is controlled by the CRL.TIMESTAMP_REF_CTRL register.

System Memory Mapped Register Access

The count value is accessible using memory-mapped registers in the LPD memory space and by local registers in the APU cores. All registers access the same value from system counter in the LPD.

The system memory-mapped register modules include:

- LPD_SCNTR (read-only)
- LPD_SCNTRS (read/write by secure master)

A72 Local Register Access

Software can access the local processor counter registers in v8 architecture.

- Enabling and disabling the counter using CNTCR [EN] bit:
  - 0: disabled
  - 1: enabled
- Match the tick count in CNTFRQ register to the to TS_REF_CLK frequency
- Set the counter value:
  - Two contiguous RW registers CNTCV [31:0] and CNTCV [63:32] hold the current count
  - Writing to CNTCV [63:32] starts the counting
- Enable halt-on-debug for a debugger to use to suspend counting. Use CNTCR [HDBG] bit [1]:
  - 0: system counter ignores halt-on-debug signal
  - 1: halt-on-debug signal halts system counter update

Changing the operating mode to change the update frequency and increment value. CNTCR, counter control register FCREQ, bits [17:8]: frequency change request.

Processor Virtual Counters

The processor's virtual counters are derived from the system counter. The virtual counter is a count subtracted from the system count. Each virtual world can have their own counter value. The single physical system counter drives and multiple virtual counters.

Triple-Timer Counter

The triple-timer counter (TTC) can generate periodic interrupts or can be used to count the widths of signal pulses from an MIO pin or from the PL.

There are four TTCs in the LPD. Each TTC has its own set of control and status registers that are accessed via its 32-bit APB programming interface attached to the LPD IOP switch. Each TTC can be individually protected by the LPD_XPPU protection unit. All three timer/counters within a TTC must have the same security status because a single APB programming interface serves the entire TTC.

Features

- Selectable clock input:
  - Internal PS bus clock based on the APB interface (IOP_REF_CLK)
  - Internal clock (from PL)
  - External clock (from MIO)
- Three independent 32-bit timer/counters
- 16-bit prescaler for the clock
- Three system interrupts, one for each timer counter
- Interrupt on overflow and counter match programmable values is generated as a system interrupt
- Increment and decrement counting
- Generate a waveform output (for example, PWM) through the MIO and to the PL fabric

Operating Modes

Each of the timer counters can operate in one of these modes:

- Interval timing mode (increment and decrement count)
- Overflow detection mode (increment and decrement count)
- Event timer mode

The register matching interrupt can be enabled in each of these modes.

Reset State

After reset, the TTC counters are set to this configuration:

- Overflow mode
- Internal clock selected
- Counter disabled
- All interrupts disabled
- Event timer disabled
- Output waveforms disabled

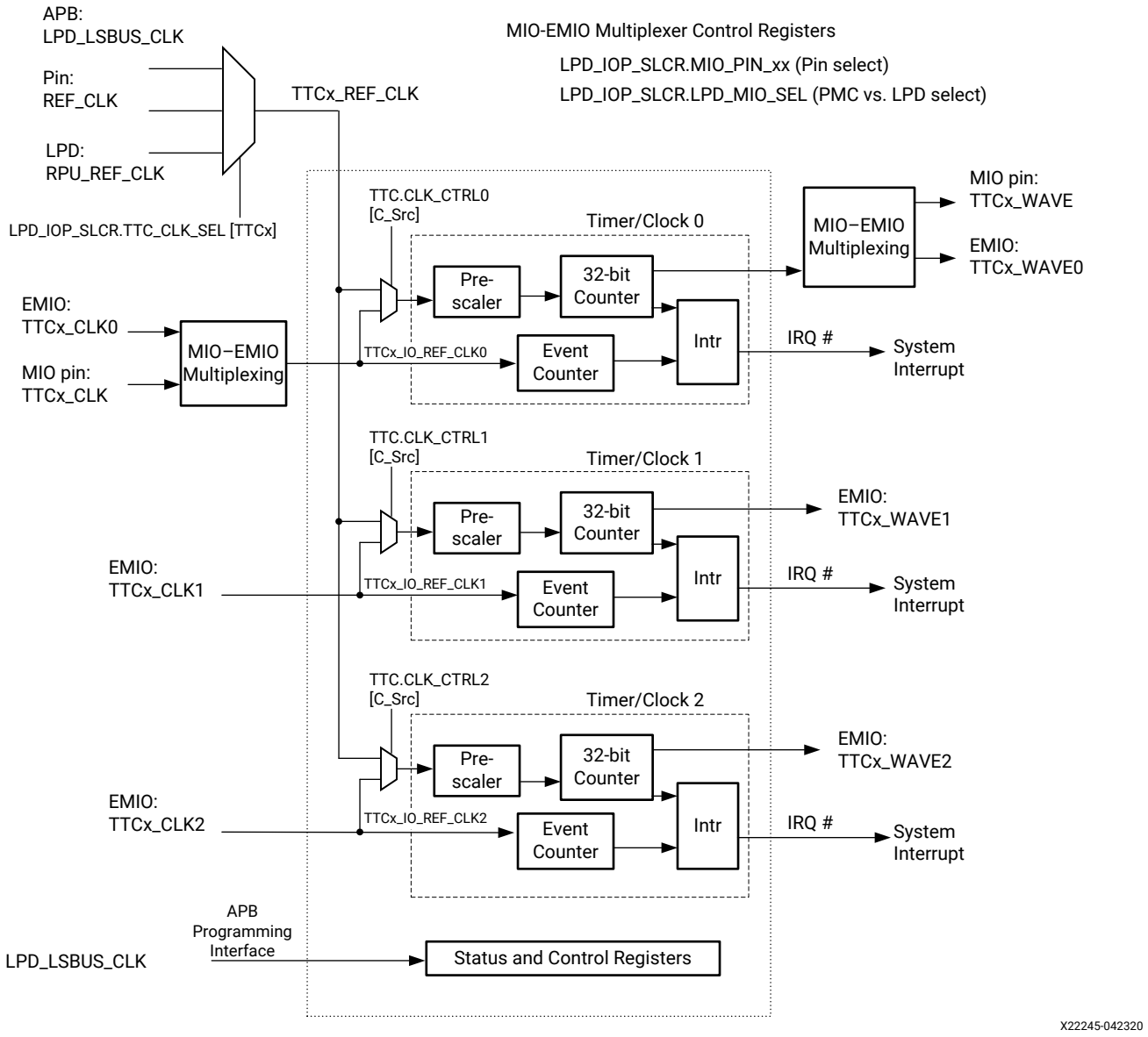
System Perspective

All four of the TTC controllers are located in the LPD subsystem.

Block Diagram

The following figure shows the input clocks, wave out signals, and system interrupts. TTC 0 has more clocking options than TTC 1 or 2 as shown in the figure.

Figure 66: **TTC Block Diagram**



Interrupts

Three interrupt signals are available for use at the system level. A system interrupt occurs when a bit in the interrupt enable register and the corresponding bit in the interrupt detect register are both set. The interrupt register takes the interrupt signals from the timer-counter module and stores them until the register is read. When the interrupt register is read by the processor, it is reset. To enable an interrupt, it is necessary to write a `1` to the corresponding bit position in the interrupt enable register.

The interrupt of various types are combined. Each type can be individually enabled within each timer/clock.

- Counter interval
- Counter matches
- Counter overflow
- Event timer overflow

Prescalar

The interface includes a prescaler module to provide a selectable clock frequency for driving the timer counter. The prescaler can be programmed to operate on the clock options shown in the figure. The selected clock is then divided down to provide the count clock. Division can be from ÷2 to ÷65536 using TTC.CLK_CTRL0 [PS_V].

Each prescaler can be independently programmed. The counter module can count up or count down, and can be configured to count for a given interval. It also compares three match registers to the counter value and generates an interrupt if one matches.

Counter Module

The counter module can increment or decrement and can be configured to count for a given interval. It also compares three match registers to the value of the counter and generates an interrupt if one matches.

Overflow Detection Functional Model

Overflow Detection Mode

If the interval bit in the counter control register is not set, the counter can count up to or down from its full 32-bit value. An interrupt is generated when the count passes through zero. To increment, when the counter value register reaches `FFFF_FFFFh`, it overflows to zero, and then the overflow interrupt is set and counting up is restarted. To decrement, when the counter value register reaches zero, the overflow interrupt is set. The counter then overflows to `FFFF_FFFFh` and counting down is restarted.

Interval Timing Functional Model

Interval Timing Mode

If the interval bit is set in the counter control register, the counter counts up to or down from a programmable interval value. An interrupt is generated when the count passes through zero. When interval mode operation is not enabled, the counter is free-running. To increment, when the counter value register is equal to the interval register value, the counter is reset to zero, the interval interrupt is set, and counting up is restarted. To decrement, when the counter value register is equal to zero, the interval interrupt is set. The counter is then reset to the interval register value and counting down is restarted.

Event Timer Functional Model

Event Timer Mode

The event control timer operates by having an internal 16-bit counter clocked by the local bus clock that resets to `0` during the non-counting phase of the external pulse and increments during the counting phase of the external pulse.

The event control timer registers (e.g., TTCn_EVENT_CONTROL_TIMER_1) control the behavior of the internal counter.

- [E_En] bit: when `0`, immediately resets the internal counter to `0`, and stops incrementing
- [E_Lo] bit: specifies the counting phase of the external pulse
- [E_Ov] bit: specifies how to handle an overflow at the internal counter (during the counting phase of the external pulse)
  - `0`: overflow causes [E_En] to be 0 (see the [E_En] bit description)
  - `1`: overflow causes the internal counter to wrap around and continues incrementing
  - When an overflow occurs, an interrupt is always generated (subject to further enabling through another register)

The event register is updated with the non-zero value of the internal counter at the end of the counting-phase of the external pulse. The event register shows the widths of the external pulse, measured in number of cycles of clock cycles. If overflow occurs, the event register is not updated and maintains the old value.

Programming Registers

See Processor Control and Status Registers for an overview of the AArch32 registers.

Accessibility

The TTC control and status registers are accessed by their APB bus interface via the PMC local AXI interconnect. These registers are protected from the non-PMC bus masters using the XPU protection unit at the incoming PMC AXI switch.

Register Overview

The following table lists the four sets of TTC registers. There are four TTC controllers (n = 0 to 3 in the table).

Table 106: **TTC Register Overview**

Name	Description
Clock_Control_x	Clock control register
Counter_Control_n	Operational mode and reset
Counter_Value_{1:3}	Current counter value
Interval_Counter_{1:3}	Interval value
Match_0_Counter_n	Match value 0
Match_1_Counter_n	Match value 1
Match_2_Counter_n	Match value 2
Match_3_Counter_n	Match value 3
Interrupt_Register_{1:3}	Interval, match, overflow, and event interrupts
Interrupt_Enable_{1:3}	ANDed with corresponding interrupt
Event_Control_Timer_{1:3}	Enable, pulse, and overflow
Event_Register_{1:3}	APB interface clock cycle count for event

TTC I/O Signals

The TTC controller includes two I/O signals. The signals are listed in the following table.

Table 107: **TTC Controller I/O Signals**

MIO				
Signal Name	**I/O**	**PMC MIO Pin**	**LPD MIO Pin**	**MIO-at-a-Glance Table**
TTC0_CLK TTC1_CLK TTC2_CLK TTC3_CLK	I	MIO-at-a-Glance Tables		0
TTC0_WAVE TTC1_WAVE TTC2_WAVE TTC3_WAVE	O			1

System Watchdog Timer

The system watchdog (SWDT) timer is used to detect and recover from various malfunctions. The watchdog timer can be used to prevent system lockup (e.g., when software becomes trapped in a deadlock). There are two watchdog functions:

- Generic timeout
- Windowed watchdog
  - Three time periods starting with the closed-window period
  - The size of the open and closed windows are programmable

The generic watchdog timer expects an interrupt handler running on a processor to restart the watchdog timer at regular intervals before the timer counts down to zero. When the timer does reach zero and the watchdog is enabled, one or a combination of a system reset, interrupt, or external signal is generated. The watchdog timeout period and the duration of any output signals are programmable.

The windowed watchdog function expects the interrupt handler to send a command on a periodic basis that is not too soon and not too late from the previous command. Watchdog timing is commonly used in embedded systems to activate fail-safe circuitry in the event of a fault. The Versal™ ACAP SWDT provides windowing, program sequence flow, and a Q&A token request/response protocols.

There are two system watchdog timers:

- LPD SWDT
  - Protecting the RPU MPCore software execution
  - Reset the RPU, other functional units, or the entire PS
- FPD SWDT
  - Protecting the APU MPCore software execution
  - Reset the APU and other FPD functional units

Applications

- Resetting watchdog too soon

When a fault occurs and causes the RPU processor to enter a software routine that writes the SWDT reset command, it is possible that the software erroneously stays in the loop and repeatedly issues the reset command in relatively quick succession. This can be detected by the SWDT and cause it to generate an interrupt. Windowing increases the complexity of resetting the timer to prevent the likelihood of this occurring.

- Performance bounding

  An additional use of the watchdog timer is to ensure the performance of the LPD stays within expected bounds. A narrow window of time is given to the open window that is carefully calculated to be achievable under all correct operation scenarios. If a fault occurs that causes some performance degradation, the timer is not reset within the narrow available time window and the SWDT issues an interrupt. If the watchdog timer detects a problem, the error manager is alerted via the SWDT interrupt.

Features

The SWDT has the following features:

- Selectable reference clock input from MIO/EMIO port signal or an LSBUS_CLK clock
- Up to 100 MHz reference clock frequency
- One or a combination of resets and interrupts
- Variable timeout periods from 1 ms to 30 seconds with a 100 MHz reference clock
- Programmable reset period
- Multiple device reset sources
- Windowed and generic watchdog timer functions

Comparison to Previous Generation Xilinx Devices

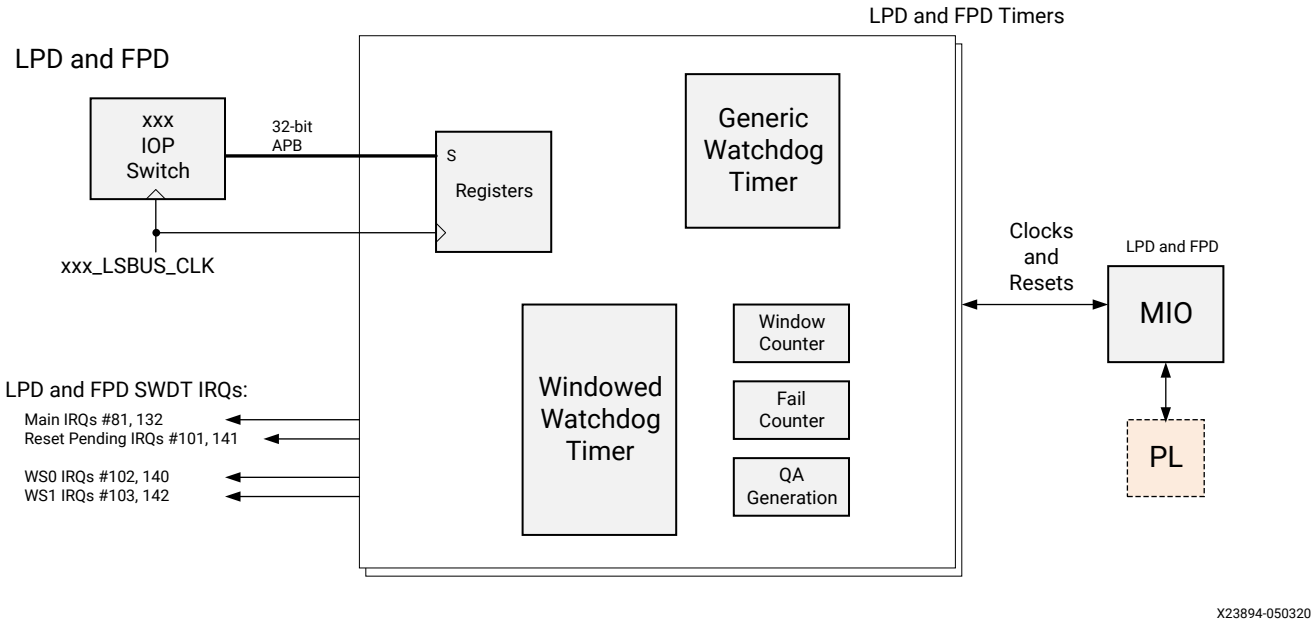
The IP for the Versal ACAP system watchdog timer is new. The new IP includes both generic functionality and windowed watchdog functions.

System Perspective

Block Diagram

The system watchdog timer block diagram is shown in the following figure.

Figure 67: **System Watchdog Timer Block Diagram**



System Interface

Both timers have a 32-bit APB slave programming interface.

- LPD SWDT is accessible from the FPD main switch
- FPD SWDT is accessible from the LPD IOP switch

System Signals

The system signals include clocks, resets, and interrupts.

Clocks

The watchdog timer receives two clocks. One clock is for the APB register module:

- LPD_LSBUS_CLK for the LPD SWDT
- FPD_TOPSW_CLK for the FPD SWDT

The other clock is used by the watchdog counters and has several sources:

- PMC MIO pins
- LPD MIO pins
- EMIO port signals

Resets

The SWDT includes a warm reset and a system reset from the LPD or FPD reset controller.

Interrupts

The SWDT generates several system interrupts that include controller, reset pending, and two window interrupts. The IRQ numbers for the SWDT and other units are listed in IRQ System Interrupts.

Programming Model

The watchdog timer includes memory-mapped registers that are accessed by software via the APB programming interface. The APB interface is used for accessing the SWDT registers via the IOP switch

Window Mode

In windowed mode, the timer starts with an adjustable period called "close (first) window time" followed by another period called "open (second) window time." The timer has to be restarted within the open window time. If software tries to restart the timer outside of the open window time period, it generates a reset. With this method, the timer includes a more stricter requirement on the software execution and provides more predictability. If software is running out of control, there is a greater probability that it fails to fulfill a strict timing requirement of the timer restart and, consequently, can reduce the probability of an errant restart of the watchdog.

As a result, the timer can increase the probability of recovering from anomalies whereas the timer might not be able to help.

The windowed mode helps to monitor the proper operation of the RPU processor. Its basic function is to expect the RPU to issue a periodic clear command to the timer by writing to one of its registers. The SWDT expects the command to occur on a periodic basis that is not too soon and not too late.

Modes and States

There are three modes of operation:

- Basic windowed watchdog mode
- Q & A watchdog mode
- Generic watchdog mode

Generic Mode

The basic function of the generic watchdog is to count for a fixed period, during which it expects to be refreshed by the system indicating normal operation. If a refresh occurs within the watch period, the period is refreshed to the start. If the refresh does not occur, the watch period expires, a signal is raised, and a second watch period starts.

The initial signal is typically wired to an interrupt and alerts the system. The system can attempt to take corrective action that includes refreshing the watchdog within the second watch period. If the refresh is successful, the system returns to the previous normal operation. If it fails, the second watch period expires, and a second signal is generated. The signal is fed to a higher agent as an interrupt or reset for it to take executive action.

The watchdog uses the generic timer system counter as the timebase for determining when to trigger an interrupt.

An explicit watchdog refresh happens when one of these events occurs:

- Generic Watchdog Refresh register is written
- Generic Watchdog Offset register is written
- Generic Watchdog Control and Status register is written

Basic Window Mode

When the timer is enabled in basic mode, the software must restart or disable it in the open (second) window duration only. If software is successful, it is considered a good event.

If the timer gets a restart attempt or disablement attempt in the close (first) window, it is considered a bad event. In this scenario, the timer is not disabled.

Also, if the timer does not get any restart or disablement before the second window expires, it is considered a bad event.

Open and Closed Windows

- **Closed Window:** A reset command received during this time is a fault of its own (interrupt).
- **Open Window:** The reset command is expected during this time period (normal).
- **Late:** The reset command was expected before this time (interrupt).

Q and A Window Mode

When the window timer mode is enabled in Q&A mode, the timer generates the question or token for the software to read from the TVAL of the Enable and Status Register (ESR). The software then generates a 32-bit response or answer. The response to each question is four bytes and is provided by four different APB writes into the TRR register.

A good event occurs if the three correct response bytes are written in the first window in correct order followed by writing the last response byte in the second window. When the response bytes are not written in correct order or in the correct window period, it is considered a bad event. The timer can be disabled only if the fail counter value is 0 in this mode.

The question/token start point can be controlled through a seed value that is taken into consideration from enable to enable. The question/token sequence (within a set of 16 questions) or repetition can be controlled through feedback configuration bits. The value is taken into consideration at the last correct answer. This requires shadow register implementation.

Challenge-Response Feature

The window mode provides a challenge-response feature. The timer has a set of 32-bit registers that contain the expected response. The responses are computed at boot time and programmed into the timer. Each time the timer is cleared a new question index is loaded into the "challenge" register. The CPU computes the response and places it in the response register and then clears the watchdog. The timer checks the response against the register value referenced by the question index. If the value does not match, the error aggregate module (EAM) is sent a signal. The challenge-response feature can be enabled or disabled. If enabled, the timer ensures that the watchdog is cleared with the time window and the challenge-response condition is satisfied.

Programming Sequences

Generic Mode Sequence

The programming sequence for the generic WDT mode is provided in this section.

1. Set the GWOR register to configure the window periods.
2. Set the GWEN bit to enable the generic WDT, which starts the first window.
3. Software might generate the explicit refresh by writing into GWRR, GWOR, or GWCSR.
4. If the explicit refresh is not received before the timeout of the first window, the GWS[0] bit is set and gwdt_ws0 is asserted, which starts the second window.
5. If the explicit refresh is not received before the timeout of the second window, the GWS[1] bit is set and gwdt_ws1 is asserted.

6. GWS[0] and GWS[1] remain asserted until an explicit refresh or watchdog reset occurs.

Basic Window Sequence

The programming steps for the basic window mode are provided in this section.

1. If required, set the [WDP] bit (to enable protection against accidental clearing).
2. Configure the first and second window count registers according to the close/open window requirements/constraints.
3. Set the interrupt position in the second window according to the requirements (SBC [7:0] & BSS [1:0]).
4. If required, the enable fail counter, program sequence monitor, and second sequence timer functions (FCE, PSME, and SSTE).
5. If PSME is enabled, write TSR0.
6. Enable the watchdog, [WEN] bit. This generates the first kick and starts the first window. This step (WEN 0->1) auto-clears the MWC bit to make the address space read-only.

   After completing the first window, the watchdog enters in the second window period and the core sets the WSW bit. Software might generate the next restart kick (or might disable the watchdog) any time after the WSW bit is set.

   The TSR1 can be written any time irrespective of whether the WSW is set or not (enable MWC, write TSR1, and disable MWC). The TSR0 and TSR1 comparison is done at the restart kick/disable event if the PSME is enabled.
7. Wait for the SWDT system interrupt.
8. Enable the MWC and restart/disable the watchdog according to the requirement (clear WINT, WSW or clear WINT, WSW, WDP, WEN).
   - If software attempts to restart or disable the watchdog in the first window, it is considered a bad event. The disable request is not honored.
   - If software does not restart or disable the watchdog before the second window rolls over, it is considered a bad event.
   - If a PSME is enabled and a TSR mismatch was found at the restart/disable time in the second window, it is considered a bad event. If the PSME is disabled, the TSR values are not compared.
   - If the fail counter is disabled, a single bad event leads to SWDT_RESET generation. The scratch bits store the last bad event. Scratch bits can be cleared post the AXI reset.
   - If the SSTE is disabled, the SWDT_RESET is generated immediately.
   - If the SSTE is enabled, the SWDT_RESET is generated after the SC count expires.
   - If software restarts/disables the watchdog in the second window, it is considered a good event.

- If the fail counter is enabled, a good event decrements the fail counter by 1 unless it is 0, and a bad event increments the fail counter by 1 unless it is 7.
- If the fail counter is 7 and a bad event occurs, it leads to SWDT_RESET generation based on [SSTE].
- Fail counter status can be tracked through the ESR register.

9. If the watchdog is restarted, it starts with a new cycle with the first window.

***Note*:** After generating the SWDT_RESET, the watchdog stops running and the [WEN] bit auto-clears.

Q and A Window Mode Programming Sequence

The programming sequence for the Q and A window mode are provided in this section.

1. If required, set the SWDT.FUNCT_CTRL [WDP] bit (to enable protection against accidental clearing).
2. Configure the first and second window count registers according to the open/close window requirements/constraints.
3. Set the interrupt position in the second window according to the requirements SWDT.FUNCT_CTRL [SBC] and [BSS] bits.
4. If required, enable the second sequence timer function using the [SSTE] bit. The fail counter is always enabled and the [PSME] bit has no meaning in this mode.
5. The seed value should be programmed in the SWDT.TOKEN_FB register before enabling the watchdog.
6. Set the first feedback configuration in the SWDT.TOKEN_FB [SEED] bit field.
7. Set the watchdog mode SWDT.FUNCT_CTRL [WM] bit.
8. Enable the watchdog SWDT.MISC_ISR [WEN] bit.
9. Enable the SWDT.WPROT [MWC] and write the TRR register to start the first question-answer sequence.

   After the window mode is enabled in Q and A mode, a write to the SWDT.TOKEN_RESP [ANS] register field triggers the first sequence run and starts the first window. Subsequently, each new token-response sequence starts after the last correct answer of the previous run.

   The default value of the [ACNT] field is “00” and after step 9, this field updates to “11.”

   Step 9 also takes into account the first feedback configuration for the first time. Subsequently, feedback configuration updates are considered at each new sequence run.

   The first question or token is presented in the ESR register after step 9. Subsequent tokens are presented at each new sequence run.

Software should provide four correct responses with correct timing for each question/token. The answer to each question is four byte data that needs to be written byte after byte (MSB to LSB) into the TRR register. The first three writes must occur in the first window time and the last write must occur in the second window.

10. Write three correct responses in the SWDT.TOKEN_RESP register in the first window interval. The first window always completes. The second window starts if the watchdog has received three correct responses in the first window.

    It is also possible to receive a fourth response (either correct or incorrect) in the first window (after receiving three correct responses). In this case, appropriate status error bots are set (TOKEN_EARLY, SEQ_ERR, and TOKEN_ERR) and the fail counter increments. The watchdog waits for the first window to expire and then moves to the second window, and expects the fourth correct response there.

    If the watchdog received no response (timeout), or received only one correct response or received only two correct responses until the first window expired, then it restarts with the first window and expects the same question-answer sequence (i.e., the question cannot be changed by token feedback configuration).

    With each correct byte response, the [ACNT] status updates. [ACNT] does not change with an incorrect response. Each incorrect response is considered a bad event and increments the fail counter. There can be more than three responses possible in the first window due to incorrect responses.

11. Wait for the SWDT system interrupt.
12. Change the token feedback configuration if required.
13. Write the last correct response in the second window.

    ***Note*:** The second window might time out in the absence of any response, expire waiting for the last or fourth correct response, or finish earlier than the programmed value after receiving the last or fourth correct response.

    If it times out or expires, the watchdog restarts with the first window and expects the same question-answer sequence (i.e., the question cannot be changed by token feedback configuration).

    SWDT.MISC_ISR [ACNT] does not change with an incorrect response. Each incorrect response is considered a bad event and increments the fail counter. With a correct byte response, the [ACNT] status updates to “11.”

    There can be more than one response possible in the second window due to incorrect responses.

14. The next question is presented in the SWDT.MISC_ISR register.
15. Repeat steps 10-13.

16. After enabled in Q and A mode, the window mode can be disabled only when the fail counter is zero. An attempt to disable the watchdog does not change the fail counter in Q and A mode.

***Note*:** After generating the SWDT_RESET, the watchdog stops running and the [WEN] bit auto-clears.

Register Reference

SWDT Register Set

The registers for the SWDT are summarized in the following table.

Table 108: **SWDT Register Set**

Register Name	Offset Address	Access Type	Description
Common Registers			
WPROT	0x0000	RW	Master write control
FUNCT_CTRL	0x0008	RW	Function control
ENABLE_AND_STATUS	0x0004	R, RW, W1C	Miscellaneous enable and status
IER IDR IMR	0x0030 0x0034 0x0038	W W R	Interrupt enable, disable, and mask
Generic Timer Registers			
G_REFRESH	0x1000	RW	Generic watchdog refresh
G_CTRL_STAT	0x2000	RW, R	Generic watchdog control and status
G_OFFSET	0x2008	RW	Generic watchdog offset
G_WARM_RST	0x2FD0	RW	Generic watchdog warm reset
Window Timer Registers			
WIND1_CFG	0x000C	RW	First window configuration
WIND2_CFG	0x0010	RW	Second window configuration
WIND_SST_CFG	0x0014	RW	Second sequence timer configuration
TASK_SIG0	0x0018	RW	Task signature reg 0
TASK_SIG1	0x001C	RW	Task signature reg 1
SST_CNT	0x0020	R	Second sequence timer
TOKEN_FB	0x0024	RW	Token feedback
TOKEN_RESP	0x0028	RW	Token response

SWDT I/O Signals

There are six SWDT signals available via the MIO. There are six groups of signals as shown in the MIO-at-a-Glance Tables and detailed in the following table.

Table 109: **SWDT Controller I/O Signals**

MIO									EMIO	
Signal Name	I/O	PMC MIO Pin				LPD MIO		MIO-at-a-Glance Table	Signal Name	I/O
		A	B	C	D	E	F			
SWDT0_CLK SWDT1_CLK	I	0 6	12 18	26 32	38 44	0 6	12 18	0		I
SWDT0_RST SWDT1_RST	O	1 7	13 19	27 33	39 45	1 7	13 19	1		O
SWDT0_RST_PEND SWDT1_RST_PEND[1]	O	2 8	14 20	28 34	40 46	2 8	14 20	2		O
SWDT0_INT SWDT1_INT[2]	O	3 9	15 21	29 35	41 47	3 9	15 21	3		O
SWDT0_WS0 SWDT1_WS0[3]	O	4 10	16 22	30 36	42 48	4 10	16 22	4		O
SWDT0_WS1 SWDT1_WS1[4]	O	5 11	17 23	31 37	43 49	5 11	17 23	5		O

Notes:

1. Window timer reset is asserted after the SST count expires (active-High).
2. Window mode, deasserts when the SWDT.MISC_ISR [WINT] bit is cleared (active-High).
3. Generic timer, asserts High after first window timeout.
4. Generic timer, asserts High after second window timeout.

XILINX

Section XI

Memory

This section includes these chapters:

Overview

The various memories are summarized in this section to provide a system perspective and to provide an accounting for safety-critical applications.

Memory Controllers

There are several types of memory controllers:

- DDR memory on NoC
  - DDR4 and LPDDR4
  - One or more 64-bit interfaces with 8-bit ECC
- Flash memory in PMC (OSPI, QSPI, SD/eMMC)
- Additional memory controllers instantiated in the PL

On-Chip Memories

There are several on-chip memories:

- OCM Memory in LPD
- XRAM Memory in LPD (device option)
- Embedded Memory
  - PPU RAM for PLM code and data
  - PMC RAM for boot image files and other data structures
  - RPU tightly coupled memories (TCMs)
  - APU L2-cache RAM

PL Building-Block Memories

On-chip memories can be instantiated in the PL using building block memories:

- Block RAM
- UltraRAM
- Distributed RAM

Small Storage Elements

For safety, many small storage memory elements include parity or ECC:

- Small Storage Elements

Data Retention

To reduce power consumption, the following memories can be placed into a data retention mode:

- 256 KB OCM
- 4 MB Accelerator RAM
- RPU TCMs
- APU L2-cache
- DDR memory

OCM Memory

The on-chip memory (OCM) contains 256 KB of RAM. It provides a 128-bit AXI slave interface port.

The 256-bit memory array provides high bandwidth for AXI read and write transactions. Optimal bandwidth is achieved when the read and write accesses are a multiple of 256 bits with 256-bit address alignment. The OCM controller implements a read-modify-write function to accommodate writes that are not 256 bits in size or are not aligned to a 64-bit boundary.

The OCM controller arbitrates between the read and write channels. The OCM has eight exclusive access monitors that can simultaneously keep track of up to eight exclusive access transactions.

Accesses to the OCM are protected by the OCM_XMPU protection unit. It divides the OCM memory space into 64 memory blocks of 4 KB each. Each block is assigned security attributes independently.

System masters access the OCM via the LPD OCM switch; this includes the two Cortex-R5F processors, and masters with access to the LPD or FPD main switches. Memory accesses from the RPU are treated with a higher priority than memory transaction requests from other masters.

Coherency

The OCM is normally accessed by the RPU, however, it can also be accessed by the APU. In cases where both the APU and the RPU use the OCM, and the APU caches the OCM memory range, the RPU can snoop the APU cache to maintain I/O coherency by routing the transaction through the FPD CCI. The APU cannot snoop the RPU caches if the RPU caches the OCM memory range.

Features

The OCM RAM and controller provide:

- 256 KB of high-speed, low-latency memory
- Optimized for RPU accesses
- 64-bit ECC with single-bit error correction and two-bit error detection
- Exclusive access requests (up to eight outstanding transactions)

- Memory protection via the OCM_XMPU with master ID and TrustZone
- Error reporting and injection
- Four memory banks with separate power islands

Comparison to Previous Generation Xilinx Devices

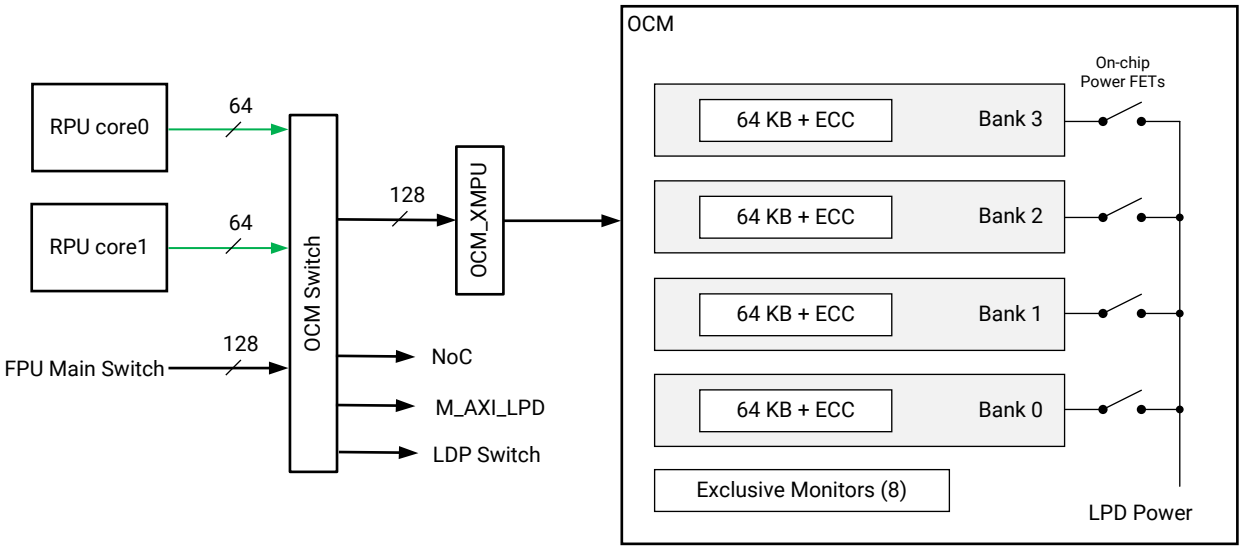
The OCM in the Versal™ device is similar to the Zynq® UltraScale+™ MPSoC.

System Perspective

Block Diagram

The OCM high-level block diagram is shown in the following figure.

Figure 68: **OCM High-level Block Diagram**



X23883-050820

States

Software must not access a bank that is powered down or in data retention.

Power

Each memory bank has on-chip FETs to control the bank's power using control register bits. See PSM Service Requests for a list of power requests.

Data Retention

The OCM memory retention state is accessible by the PMC_GLOBAL.AUX_PWR_STATE register.

Address Map

The address range assigned to the OCM memory exists in the higher 256 KB in the 4 GB address map (32-bit addresssing). This cannot be modified.

The 256 KB RAM array is mapped to a high address range (`0xFFFC_0000` to `0xFFFF_FFFF`) in a granularity of four independent 64 KB banks. Each bank is on a separate power island controlled by the PMC. The mapping summary is listed in the following table.

Table 110: **OCM Memory Bank Mapping**

Memory Bank	Address Range	Size
0	`0xFFFC_0000` to `0xFFFC_FFFF`	64 KB
1	`0xFFFD_0000` to `0xFFFD_FFFF`	64 KB
2	`0xFFFE_0000` to `0xFFFE_FFFF`	64 KB
3	`0xFFFF_0000` to `0xFFFF_FFFF`	64 KB

Memory Address Protection

The OCM memory is protected by the OCM_XMPU protection unit using Master ID, TrustZone status, and address ranges that are configured as read and/or write.

The XMPU defines 16 address regions where each region is configured for read or write protection for a set of masters, and it can be configured as a secure or non-secure TrustZone region. The base address and upper address for each region is defined with a 4 KB granularity.

If a read/write, master, or TrustZone violation occurs, the XMPU returns an SLVERR error signal back to the master. Valid data is not returned to the master on a read operation and no data is written to the OCM on a write operation.

ECC Protection

The 64-bit ECC protection applies to both AXI interfaces. If all eight bytes are being written, a new ECC value is generated (on per 64-bit aligned basis) and written to the ECC part of memory. If a sub-64 bit write transaction is requested (less than 8 bytes), the controller reads the associated 64-bit data from the RAM, modifies it, and writes it back with a new ECC value. If the read part of the read-modify-write sequence detects an uncorrectable error, the write is not performed and the controller responds by asserting the SLVERR error signal back to the master.

Error Reporting

If a correctable/uncorrectable error is detected during read, the read address is captured in the OCM controller. For a correctable error, an optional system interrupt can be generated. For uncorrectable errors, a SLVERR response is generated.

Subsequent errors generate an error signal and a SLVERR response, but if the previous read error address is not cleared by software, then any follow-on read error address is lost.

Error Injection

Software can inject 1-bit or 2-bit errors per 64-bit (an ECC word) based on register values (64+8 bits). The 72 bits are XOR-ed with data and syndrome bits being written.

ECC Operations

If errors occur due to a fault injection or other reasons, an interrupt is generated. The ocm.ISR register provides the interrupt status and the cause of the error. This is a sticky register that holds the value of the interrupt until cleared by a value of 1. Read bits 6 and 7 of the ocm.ISR register for information on whether the error is correctable or uncorrectable.

Read Correctable Error Registers

1. Retrieve the address of the first occurrence of an access with a corrected error. Read the 18-bit [ADDR] bit field in the ocm.CE_FFA register.
2. Retrieve ECC syndrome bits of corrected error. Read ocm.CE_FFE [SYNDROME] bit field.
3. Retrieve corrected data. Read the four data words using the ocm.CE_FFD0, 1, 2, and:3registers.

Read Uncorrectable Error Registers

1. Retrieve the address of the first occurrence of an access with an uncorrected error. Read the 18-bit [ADDR] bit field in the ocm.UE_FFA register.

2. Retrieve ECC syndrome bits of uncorrected error. Read ocm.UE_FFE [SYNDROME] bit field.
3. Retrieve uncorrected data. Read the four data words using the ocm.UE_FFD0, 1, 2 and 3 registers.

Inject Error

Errors can be injected into a RAM array.

1. Enable error response by setting the third bit of the ocm.ERR_CTRL register.
2. Enable ECC by setting the zeroth bit of the ocm.ECC_CTRL register.
3. To only detect single bit errors, set the first bit of the ocm.ECC_CTRL register. By default this bit is zero and it indicates that single-bit errors are corrected.
4. To inject an error on every write after fault injection count cycle, set the second bit of the ocm.ECC_CTRL register. If a zero is programmed for the same bit in the register, then only a single fault is injected.
5. The fault injection count must be programmed by setting the required value in the first 24 bits of the ocm.FI_CNTR register.
6. A fault can be injected into the syndrome bits using the ocm.FI_SY register. Faults in the data words can be injected using the ocm.FI_D0, 1, 2, and 3 registers.
7. Interrupts can be enabled for different errors by setting the required bits of the ocm.IEN register.
8. Unwanted interrupts can be disabled by setting the required bits of the ocm.IDS register.
9. Reading the ocm.IMR register gives information regarding the type of interrupts that are masked out. This is a read-only register and reflects the settings done on the ocm.IER and ocm.IDR registers.

XRAM Memory

The 4 MB accelerator RAM (XRAM) is available in some Versal™ AI Core series devices. The XRAM is divided into four separate memory banks with four system interfaces: an AXI port from the LPD PS and three PL AXI ports. The XRAM supports simultaneous access by each port to its associated bank. It also allows full cross-bank access from any port to any bank.

The LPD PS port is a 128-bit AXI4 interface. Each PL interface is AXI4 and can be independently configured for 32, 64, 128, or 256-bit data widths. Each port has their own XMPU protection unit to facilitate simultaneous read and write access with full security and access type restrictions on a per master basis.

The XRAM is initialized by the PLM during the boot process. The control of the XRAM defaults to the programming interface from the PMC. This can be handed off to the PL after boot, if desired. In this case, the LPD PS AXI port is disabled and all controls, including the memory clock are driven from the PL.

For power management, each bank is split into four 256 KB sub-banks that can individually power-down. The power to the sub-banks is controlled by register bits. The memory also provides a low-power, data retention mode.

Some Versal ACAPs include accelerator RAM, an additional 4 MB of memory with ECC located outside of the PS. This memory provides direct access from the RPU via a 128-bit AXI interface and can also be accessed from the PL through two 256-bit AXI interfaces. The memory is divided into four banks supporting concurrent read or write accesses from the PL and RPU to different banks.

Features

- Four 1 MB banks with parallel, concurrent access paths
- Sixteen power islands (256 KB each) with data retention control
- Each PL interface can be configured as 32, 64, 128, or 256-bit data width
- Controlled by LPD or PL
- 64-bit ECC protection on memory
  - ECC values are stored in physically separate memory from the data

    - ECC error injection support
- Read-modify-write for sub-width (less than 64-bit) transfers
- TrustZone support with 4 KB granularity
- Propagation of all errors to the system error accumulator

Comparison to Previous Generation Xilinx Devices

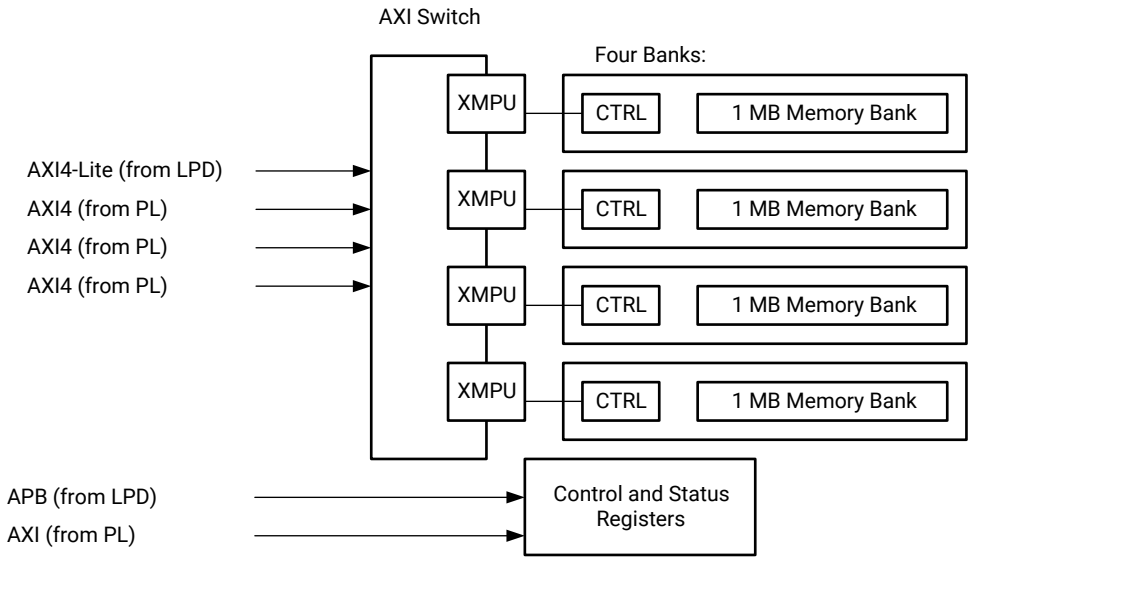
The accelerator RAM is new to Xilinx® Versal devices.

System Perspective

Block Diagram

The XRAM block diagram is shown in the following figure.

Figure 69: **XRAM Block Diagram**



System Interfaces

AXI4 Interfaces

The AXI4 ports include:

- One 128-bit to OCM switch in the LPD
- Three configurable 32 to 128-bit ports to PL

Programming Interface

There are two programming interface choices:

- 32-bit APB slave port attached to the LPD slave switch (reset default)
- AXI slave interface connected to the PL (optional interface)

Clock

The accelerator RAM clock is controlled by a clock divider in the LPD. It is clocked at the same frequency as the RPU and OCM memory using the CRL.CPU_R5_CTRL register.

Reset

The accelerator RAM is reset by the CRL.RST_XRAM [RESET] bit at `0xFF5E_0364`.

Power

The XRAM is located in the LPD power domain and includes 16 power islands.

Address Map

The XRAM has two main addressable features. The memory banks are addressable from `0xFE80_0000` to `0xFEBF_FFFF` (4 MB). The registers and control interface are addressable at `0xFF8E_0000`.

- Control and status: `0xFF8E_0000`
- Four XMPU protection units: `0xFF93_0000`, `0xFF93_4000`, `0xFF93_8000`, `0xFF93_C000`
- SLCR: `0xFF95_0000`

Memory Address Protection

The XMPU protection units described in Xilinx Memory Protection Unit are included; one on each of the four memory banks.

ECC Protection

ECC protection is applied to 64-bit data to generate eight syndrome bits.

Errors are reported as system interrupts IRQ #111 for correctable and #112 for uncorrectable. They are also reported as system errors to the PSM.

External Memory

External memory is controlled by on-chip memory controllers:

- DDR memory controller (DDR MC)
- Flash memory controllers (QSPI, OSPI, SD)
- PL instantiated DDR memory controller

DDR Memory Controller

The integrated DDR memory controller (DDRMC) supports both the DDR4 and LPDDR4 memory interfaces. It can be configured with a 32-bit or 64-bit DRAM interface with or without ECC. All devices have at least one DDR memory controller, and some devices include multiple DDR memory controllers.

The controller has four NoC interface ports to handle multiple streams of traffic and supports five quality of service (QoS) classes to ensure appropriate prioritization of the memory requests. The controller accepts burst transactions and implements command reordering to maximize the efficiency of the memory interface. Reliability features include error correction, address parity, and DQS gate tracking. Power saving features include DRAM self-refresh and automatic DRAM power down.

For more information on the integrated DDRMC, see the *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* (PG313).

Flash Memory Controllers (in PMC)

The flash memory controllers are introduced in I/O Signals with details in Section XIII: Flash Memory Controllers.

- Quad SPI controller
- Octal SPI controller
- Two SD/eMMC controllers

Embedded Memory

There are several types and sizes of on-chip, embedded memory.

- PPU RAM
- PMC RAM
- RPU tightly-coupled memory (TCM)
- APU L2-cache memory

PPU RAM

The PPU RAM is 384 KB with ECC protection. The memory is used for the platform loader and manager (PLM) software code and data structures. It is normally only accessed by the PLM.

PMC RAM

The PMC RAM is 128 KB with ECC protection. The memory is used by the PLM to store shared data structures and other purposes. It is accessible to any master with access privileges. It is protected by the PMC_XMPU protection unit.

RPU Tightly-Coupled Memory

The RPU MPCore includes small memories in 6 banks for high-bandwidth and predictable latencies. The tightly couple memory (TCM) banks are described in Tightly Coupled Memories.

APU L2-Cache Memory

The APU includes a 1 MB L2-cache that can be shared with processor in the PL with its own L2 cache using two-way coherency/snoop activities. The L2 cache is coupled with the CCI and SMMU to provide memory sharing and virtualization. The L2-cache is part of the APU.

Small Storage Elements

There are a few opportunities to store small amounts of data in the system.

Global Storage Registers

The PMC includes global storage registers. The persistent global storage registers maintain their data value through a system reset, but not a power-on reset.

Battery-Backed RAM

The battery-backed RAM (BBRAM) is in the PMC and can be used to store AES keys.

I/O Peripheral Controllers

The I/O peripherals (IOP) are integrated into the PMC and LPD. This section includes chapters for the following peripherals:

PMC I/O Peripherals

- GPIO Controller (one instance, two banks)
- I2C Controller (one instance)

LPD I/O Peripherals

- CAN FD Controller (two instances)
- Gigabit Ethernet MAC (two instances)
- GPIO Controller (one instance, one bank)
- I2C Controller (two instances)
- SPI Controller (two instances)
- UART SBSA Controller (two instances)
- USB 2.0 Controller (one instance)

XILINX

Chapter 67

CAN FD Controller

The control area network (CAN) is a computer network protocol and bus standard designed to allow controllers and devices to communicate with each other. CAN is designed specifically for automotive applications but is also used in other applications. The CAN flexible data-rate (CAN FD) controller provides additional features with higher I/O clock frequencies and more buffering.

The controller is designed to the ISO 11898-1/2015 specification that includes backward compatibility to the CAN 2.0 specification. There are two CAN FD controllers, and both are located within the LPD IOP.

This chapter contains these sections:

Features

The CAN FD controller features include the following:

Protocol

- Standard CAN 2.0 frames specified in ISO specification 11898-1:2015
- CAN FD protocol features
  - 64-byte frames
  - Flexible data rates up to 8 Mb/s

- Normal data rates up to 1 Mb/s

Controller Features

- TX delay compensation of up to three data bits
- TX mailbox buffers, 32
- TX event buffer
- RX buffers, dual 64-messages deep with 32 ID filter mask pairs
- Priority message IDs, lowest ID transmitted first
- TX message cancellation
- Timestamp for TX and RX
- Separate error logging for fast data rates

Messaging Features

- Disable auto-retransmission (DAR) mode
- Disable protocol exception event mode
- Bus monitoring, snoop mode
- Sleep mode with wake-up interrupt
- Bus-off recovery
  - Auto-recovery
  - User-intervention auto-recovery
- Up to three data bit transmitter delay compensation
- Internal loopback mode

Comparison to Previous Generation Xilinx Devices

The CAN FD controller is different from the CAN controller in Zynq® UltraScale+™ MPSoCs. The architecture, features, and programming model are new including the flexible data rate functionality.

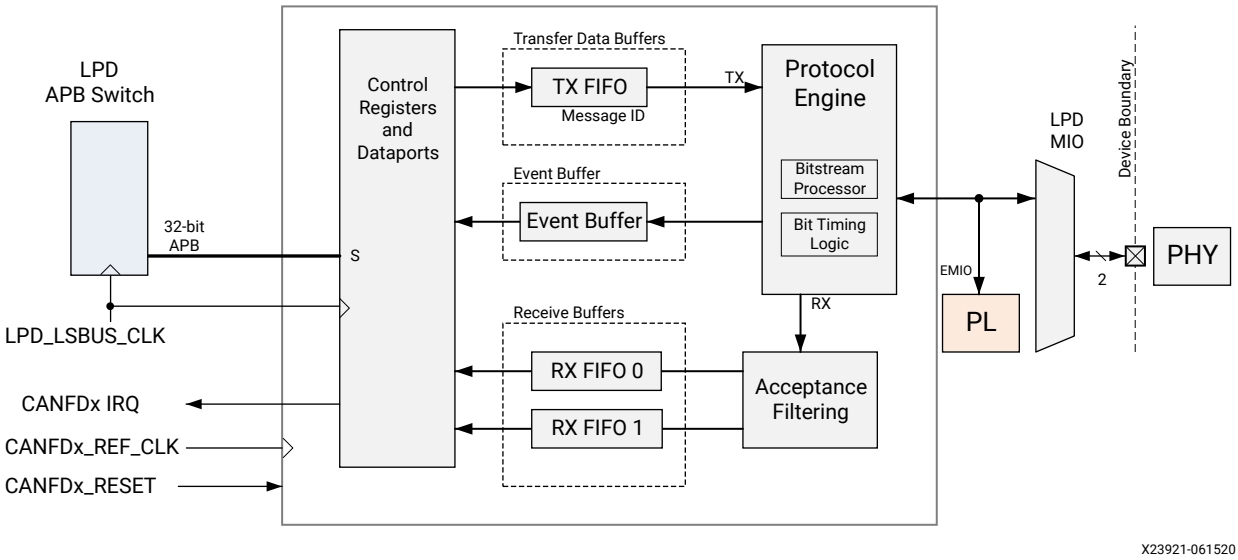
System Perspective

The main functions are divided into two independent layers as shown in the block diagram. The object layer interfaces with the LPD APB switch. The transfer layer interfaces with the external PHY. The CAN FD object layer provides a transmission and reception method to manage message buffers.

Block Diagram

The controller is divided into the object layer and transfer layer. The object layer includes the APB interface, programming registers with data ports, and the RX and TX buffers. The transfer layer connects to the I/O signals routed to the LPD MIO device pins or the EMIO interface to the PL.

Figure 70: **CAN FD High-level Block Diagram**



Object Layer

The messaging functionality is divided into the following sections.

- **Register module:** This module allows for read and write access to the registers using the APB programming interface.
- **TX buffer management module:** The TX buffer management module (TBMM) interfaces with the protocol engine to provide the next buffer to transmit on the CAN bus. It manages the host access to the TX RAM buffer.

- **RX buffer management module:** The RX buffer management module (RBMM) interfaces with the protocol engine to provide storage for message reception from the CAN bus. It manages the host access to the RX RAM buffer.

Logical Link Layer

The messaging functionality is divided into these sections:

- Configuration Sequence
- Loopback Modes

Related Information
PS Interconnect Diagram

MAC Transfer Layer

The transfer layer provides these main functions:

- Initiation of the transmission process after recognizing bus idle (compliance with interframe space)
  - Serialization of the frame
  - Bit stuffing
  - Arbitration and passing into receive mode in case of loss of arbitration
  - ACK check
  - Presentation of a serial bitstream to PHY for transmission
  - CRC sequence calculation including stuff bit count for FD frames
  - Bit rate switching
- Reception of a serial bitstream from the PHY
  - Deserialization and recompiling of the frame structure
  - Bit de-stuffing
  - Transmission of ACK
  - Bit rate switching
- Bit timing functions
- Error detection and signaling
- Recognition of an overload condition and reaction

System Interface

The CAN FD controller has a single 32-bit APB slave programming interface.

APB Programming Interface

The programming interface provides access to the configuration, control, and status registers, as well as data ports for the RX and TX message buffers. An overview of the controller registers is shown in Register Reference.

Interface Clock

The CAN controller is clocked by the APB programming interface using the LPD_LSBUS_CLK.

System Signals

The system signals include clock, reset, interrupt, and error signals.

- Reference Clock input
- Controller Reset input
- System Interrupt output
- System Error output

Reference Clock

The LPD clock controller provides a reference clock to each controller:

- CAN0_REF_CLK reference clock (CRL.CAN0_REF_CTRL register)
- CAN1_REF_CLK reference clock (CRL.CAN1_REF_CTRL register)

Controller Reset

The controller has one reset state that is entered when the device, PS, or LPD are reset, or by an individual reset to the controller from the LPD reset controller. The CANFD register set also includes a local reset control bit, CANFD.SRR [SRST].

For more information, see Reset State.

System Interrupt

Each controller generates a level-sensitive system interrupt that is routed to the RPU and APU GICs, and proxies as listed in the System Interrupts chapter.

- CANFD0: IRQ#52

- CANFD1: IRQ#53

System Error

The APB programming interface generates an address decode error if it detects an access violation.

I/O Interface

The I/O functionality includes the two-wire CAN signals. For more information, see I/O Signal Reference. An internal loopback connection can be enabled for test and debug.

The controller I/O can be routed to one of several places:

- LPD MIO pins
- PMC MIO pins
- PL EMIO port signals

Programming Model

The CAN FD controller includes a memory-mapped APB programming interface for software:

- Control and status
- Transmit data
- Receive data
- Establish autonomous monitoring and responses

The controller includes several Modes and States.

Modes and States

The CAN modes and states include:

- Reset State
- Mode Table
- Mode Transition
- Configuration Mode
- Normal Mode
- Sleep Mode

- Snoop (Bus Monitoring) Mode
- Loopback Modes
- Protocol Exception Event State
- Bus-Off Recovery State

Reset State

The controller is reset from several sources:

- Local controller reset: can.SRR [SRST] bit
- Resets from the LPD reset controller
  - CANFD0_RESET (CRL.RST_CAN0 [RESET] bit)
  - CANFD1_RESET (CRL.RST_CAN1 [RESET] bit)
- POR and system reset comes from PMC register controls for the LPD, PS, and whole device

Each reset source has the same effect on the controller as summarized in the following table.

Table 111: **CAN Reset Effects**

Reset Name	Reset Type	APB Interface and Registers	Protocol Engine	Buffers and Acceptance Filters
CAN.SRR [SRST]	Software	Yes	Yes	No
CRL.RST_CANx [RESET]				
POR or PS	Hard			

***Note*:** Because the buffers and acceptance filters are not reset, the software needs to ensure that they are programmed appropriately before operation.

Local CAN Reset Control

Write a 1 to the CAN.SRR [SRST] bit (this bit is self-clearing).

CRL Reset Control

The CRL reset is not self-clearing. Write a 1 and then write a 0 to the CRL.RST_CANx [RESET] bit. These can be individual back-to-back writes to the programming interface.

Mode Table

The control and status bits for the controller modes are listed in the following table. These relate to the Mode Transition figure.

Table 112: **CAN Controller Modes**

Controller Mode	System Reset[1]	Software Reset Register (SRR)		Mode Select Register, MSR			Status Register, SR						
		Reset [bits]		Control [bits][4]			Status [bits]						
		SRS	CEN	LBACK	SLEEP	SNOOP	CONFIG	NORM	SNOOP	SLEEP	LBACK	BSFR	PEE
Reset	1	x	x	x	x	x	1	0	0	0	0	0	0
	0	1											
Configuration		0	0										
Normal[5]		0	1	0	0	0	0	1					
Snoop[5]				0	0	1		1	1				
Sleep				0	1	0		0	0	1			
Loop back				1	0	0		0	0	0	1		
BSFR[2]				0	x	0		x	0	0	0	1	
PEE[3]				0	x	x		x	x	0	0	0	1

Notes:

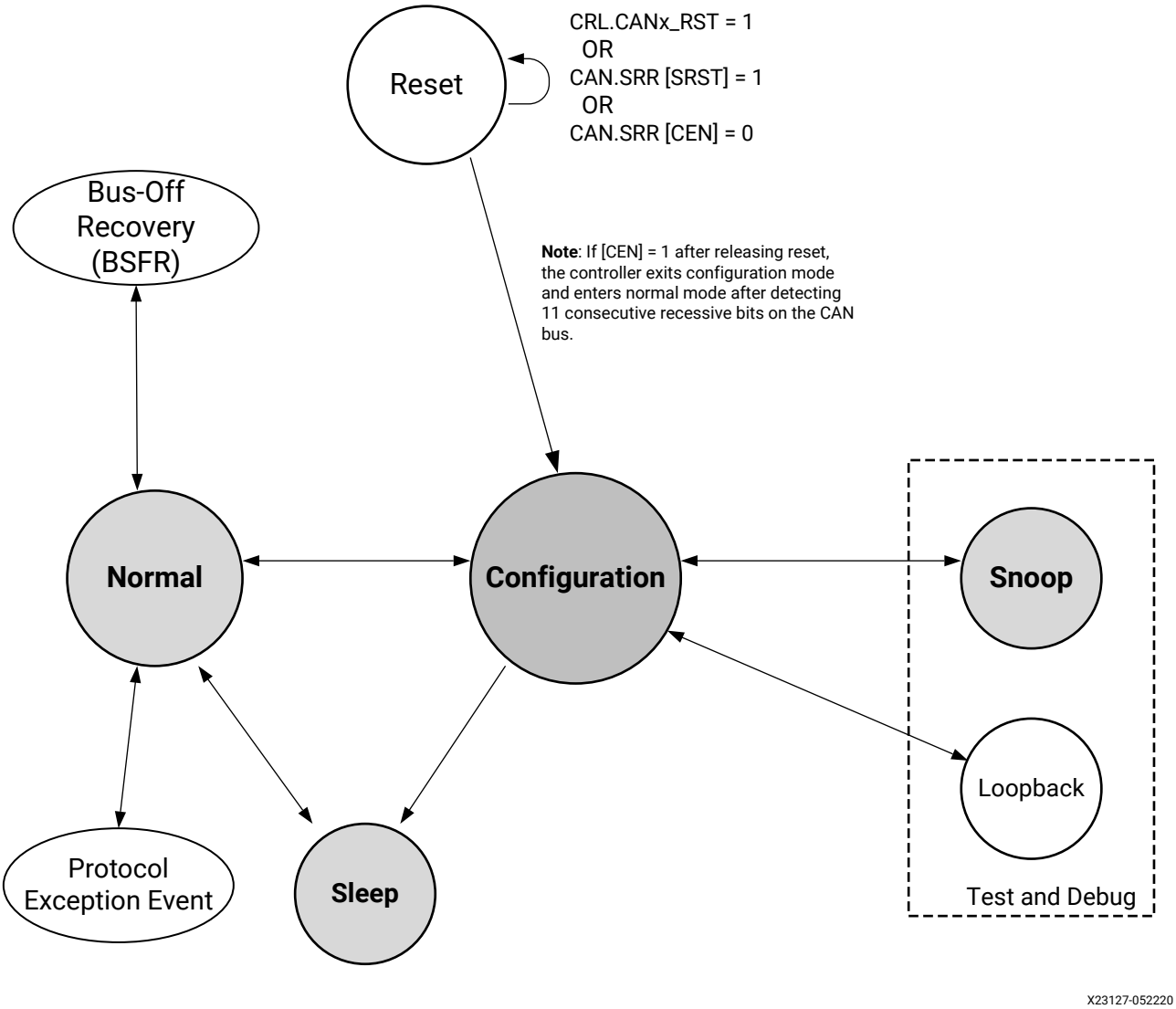
1. A hard system reset can be generated by the LPD reset controller using the CRL.RST_CANx [RESET] or one of multiple system-level resets.
2. The transition to bus-off state depends on the transmit error count value as per standard specification. The recovery from bus-off state depends on the MSR [SBR] and [ABR] settings as per respective bit behavior descriptions. Bus-off recovery can be tracked through status bit [BSFR_CONFIG] bit and the ECR [REC] field. Entry and exit from bus-off state can generate an interrupt.
3. The transition to protocol exception event state (PEE) depends on the MSR [DPEE] bit.The controller enters and exits the PEE state as per ISO standard specification and this is reflected by the status bit SR [PEE_CONFIG. The entry into the PEE state can generate an interrupt.
4. An "x" indicates "don't care" for control bits and has no meaning for status bits.
5. The sample point range should be 50%-80% of bit time for reliable operation.

Mode Transition

The mode transitions are shown in the following figure. The transitions are primarily controlled by the resets, [CEN] bit, MSR and SRR register settings, and a hardware wake-up mechanism.

The mode transition conditions are shown in the Mode Table.

Figure 71: **CAN Mode Transition Diagram**



Configuration Mode

The configuration mode is normally entered following reset, but can also be entered from several other modes including normal, snoop, sleep, and loopback. If the controller needs to move from the BSFR or PEE state, a system reset is required.

Configuration Mode Characteristics

The CAN FD has the following configuration mode characteristics:

- Controller loses synchronization with the CAN bus and drives a constant recessive bit on TX line.

- Error Counter register (ECR) and Error Status register (ESR) are reset.
- BTR and BRPR registers can be modified.
- SR [CONFIG] bit is set = 1.
- Controller does not receive or transmit any new messages.
- All configuration registers are accessible.
- If there are messages pending for transmission when [CEN] is written 0, the controller does not transmit any messages and:
  - TX messages are preserved unless canceled; message cancellation is allowed.
  - TX messages are transmitted when normal operation is resumed.
  - New TX messages can be added for transmission (provided the CANFD.MSR [SNOOP] is not set = 1).
- If there are RX messages in the RX buffer when [CEN] is set = 0, they are preserved until host reads them, but the controller does not receive new messages.

Interrupts

In configuration mode, the interrupt status might change.

- Interrupt Status register bits are cleared:
  - ARBLST
  - TXOK and RXOK
  - RXOFLW, RXOFLW_1
  - ERROR
  - BSOFF
  - SLP and WKUP
- Interrupt Status register bits are not cleared due to possible cancellation using:
  - TXTRS and TXCRS

***Note*:** A system interrupt is generated if an interrupt bit is set = 1 in the ISR register and the corresponding bit in the IER register = 1.

Exit Configuration Mode

The controller stays in configuration mode until the SRR [CEN] bit is set = 1.

- After the [CEN] bit is set to 1, the controller waits for a sequence of 11 nominal recessive bits before exiting configuration mode.

- Move the controller to normal, snoop, sleep, or loopback mode by setting one of the MSR [SNOOP], [SLEEP], and [LBACK] bits.

Normal Mode

In normal mode, the controller transmits and receives messages per industry specifications outlined in CAN FD Controller.

In normal mode, the controller does not store its own transmitted messages. The controller can enter normal mode only when the snoop, sleep, and loopback modes are deselected.

Sleep Mode

The controller enters sleep mode from configuration or normal mode when the MSR [SLEEP] bit is set = 1, CAN bus is idle, and there are no pending transmission requests.

The controller enters configuration mode when any configuration condition is satisfied.

When the controller wakes up, it clears the MSR [SLEEP] request bit and also clears the corresponding status bit. The controller enters normal mode under the following wake-up conditions:

- [SLEEP] is set = 0
- [SLEEP] bit is = 1, bus activity is detected
- New message are posted for transmission

Interrupt bits are set when the controller enters or wakes up from sleep mode.

Snoop (Bus Monitoring) Mode

Snoop mode is used for test and debug. When in snoop mode, the controller must only be programmed to enter configuration mode or be held in reset. In snoop mode, these actions occur:

- Controller transmits a recessive bitstream onto the CAN bus
- Controller does not participate in normal bus communication
- Controller receives messages that are transmitted by other CAN nodes but does not ACK
- RX messages are stored based on acceptance filtering
- Software can program acceptance filters to dynamically enable/disable and change criteria
- Error counters are disabled and cleared to 0; reads to error counter registers return to 0

RECOMMENDED: *Xilinx recommends that snoop mode be programmed only after system or software reset.*

Loopback Modes

There are two loopback types:

- Self loopback: TX output from a controller is connected to its own RX input
- Controller-to-controller loopback: connects CAN controller 0 to controller 1

Self Loopback

In self-loopback mode, the controller receives the messages that it transmits using an internal circuit from its TX signal to its RX signal. The received messages are stored in receive buffers based on an ID match result. The transmissions are acknowledged to itself by the receiver. The controller also stores the received messages (based on an ID match result). In self-loopback mode, the controller is disconnected from the MIO mux and pins.

This mode is normally used for diagnostics. The loopback mode is selected using the CANFD.MSR [LBACK] bit. The controller receives messages that it transmits. Received messages are stored in receive buffers based on an ID match result. The controller also stores its own transmitted messages (based on an ID match result).

The controller does not participate in normal bus communication and does not receive any messages transmitted by other CAN nodes (external TX line is ignored). It drives a recessive bitstream on the CAN bus (external TX line).

Controller-to-controller Loopback

The controller-to-controller loopback connection is selected using the LPD_IOP_SLCR.MIO_LOOPBACK [CAN0_LOOP_CAN1] control bit. When the [CAN0_LOOP_CAN1] is set = 1, these connections are made:

- CAN0 TX output is connected to the CAN1 RX input
- CAN1 TX output is connected to the CAN0 RX input

Protocol Exception Event State

The controller enters the CAN FD protocol exception event (PEE) state if the controller receives the "res" bit as recessive in the CAN FD frame (provided MSR [DPEE] bit is not set = 1). The controller exits this state after detecting a sequence of 11 nominal recessive bits on the CAN bus, and as per the protocol specification, the transmit and receive error count remains unchanged in this state.

Bus-Off Recovery State

The controller enters the bus-off state if the transmit error count reaches or exceeds its terminal point. Recovery from bus-off states is governed by the auto recovery [ABR] or manual recovery [SBR] bit setting in the MSR register and is done according to the protocol specification.

Note: In the case of a protocol exception or bus-off event, any pending messages/frames for transmissions must be canceled and re-queued for proper operation after recovering from these events.

Configuration Sequence

The following steps are for configuring the controller when it is powered on or after system or software reset.

1. Choose the operating mode:

   Note: The sample point position programming follows the industry standard.

   - Normal—write 0s to the [LBACK], [SNOOP], and [SLEEP] bits in the MSR. Write required value for [BRS] and [DAR] fields in the MSR register.
   - Sleep—write 1 to [SLEEP] bit and 0 to [LBACK] and [SNOOP]. Write required value for [BRS] and [DAR].
   - Loopback—write 1 to [LBACK] and 0 to [SLEEP] and [SNOOP] bits. Write required value for [BRS].
   - Snoop—write 1 to [SNOOP] bit and 0 to [LBACK] and [SLEEP].

2. Configure the Transfer Layer Configuration registers.

IMPORTANT! *For proper operation, ensure that all CAN FD nodes in the network are programmed to have the same arbitration phase bit rate, data phase bit rate, arbitration phase sample point position, and data phase sample point position.*

   - Program the Arbitration Phase (Nominal) Baud Rate Prescaler register and Arbitration Phase (Nominal) Bit Timing register with the value calculated for the particular arbitration phase bit rate.
   - Program the Data Phase Baud Rate Prescaler register and Data Phase Bit Timing register with the value to achieve desired data phase bit rate.
     - The Data Phase Bit Timing register also contains TDC control fields.

   Note: The bit rate configured for the data phase must be higher than or equal to the bit rate configured for the arbitration phase. The Transfer Layer Configuration registers can be changed only when the CANx.SRR [CEN] bit is 0.

Note: For CANFD operation with BRP=1 (register value 0), set both BRP for nominal and data phase as 1 (register value 0). Additionally, the user needs to program MSR.BRP_1_EN = 1 when BRP = 1. For BRP!=1, MSR.BRP_1_EN must be 0.

3. Configure the Acceptance Filter registers (AFR, AFMR, AFIR) to the following:
   - Write a 0 to the UAF bit in the register corresponding to the Acceptance Filter Mask and the ID register pair to be configured.
   - Write the required mask information to the Acceptance Filter Mask register.
   - Write the required ID information to the Acceptance Filter ID register.
   - Write 1 to the UAF bit corresponding to the Acceptance Filter Mask and ID register pair.
   - Repeat the steps for each Acceptance Filter Mask and ID register pair.
   - To enable RX buffer 1, arrange the Filter Mask and ID register as per the requirement. The [RXFP] field in the RX buffer Watermark register also needs to be set accordingly to a value less than `31d`.
4. Program the Interrupt Enable registers as per requirements.
5. Enable the protocol controller by writing a 1 to CAN.SRR [CEN]. After the occurrence of 11 consecutive recessive bits, the controller clears CAN.SRR [CONFIG] to 0 and sets the appropriate mode status bit in the Status register.

RECOMMENDED: *If the [CEN] bit is cleared during the controller operation, then reset the controller, too.*

RECOMMENDED: *The [LBACK], [SLEEP], and [SNOOP] bits should never be set to 1 at the same time.*

Message Transmission

All messages written in the TX buffer should follow the required message format for ID, DLC, and DW fields described earlier. Each [RRnn] bit of the TX Buffer Ready Request (TRR) register corresponds to a message element in the TX buffer.

Software Actions

1. Poll the TRR register to check current pending transmission requests.
2. If all bits of the TRR register are set, a new transmission request can be added only if:
   a. One or more buffer transmission requests are canceled, or
   b. One or more buffer transmission completes

3. If one or more bits of the TRR register are unset/clear, a new transmission request can be added as follows:
    a. Prepare one or more message elements in the TX buffers (by writing valid ID, DLC, and DW fields of each message element of the respective TX buffer). If event logging is required for this message element, set the CAN.DLC [EFC] bit.
    b. Enable interrupt generation as required.
    c. Set corresponding TRR register bit(s) to enable buffer ready requests. The host can enable many transmission requests in one write to the TRR register.
    d. Wait for interrupt (if enabled) or poll the TRR register to gather the request status.
4. The controller clears the TRR bit when a respective buffer request is completed (either due to transmission, cancellation, or DAR mode transmission).
5. The host can read the TX event buffer to determine the message timestamps and the order of transmissions.

    Note: The CAN.ISR [TXOK] bit is set after the core successfully transmits a message. The CAN.ISR [ARBLST] bit is set if the controller loses bus arbitration while transmitting a message. The [ERROR] bit in the ISR is set if the message transmission encountered any errors.

Controller Actions

1. The controller determines the next highest priority buffer to be transmitted. If two buffers have the same ID, the buffer with the lower index is selected.
2. If enabled, copies the ID and DLC fields to the TX event buffer and adds a message timestamp and event type.
3. Clears the respective [TRR] bit when the transmission request is served (either by successful transmission on the CAN bus, cancellation, or DAR-based transmission).
4. If enabled through the IETRS and IER, the CAN.ISR [TXRRS] bit is set = 1 and an interrupt is generated.

Note: The controller accesses the message element space of a buffer in the TX buffer only if the respective [TRR] bit is set.

Note: The host should respect the access rule to avoid memory collisions. That is, after the host sets a buffer ready request through the TRR register, it should not read or write the respective message element space until the respective [RRnn] bit is in a clear/unset state.

Cancellation

Each [CRnn] bit of the TX Buffer Cancel Request (CAN.TCR) register corresponds to a message element in the TX buffer (and, consequently, corresponds to a CAN.TRR [RRnn] bit).

Software Actions

1. Poll the TRR register to check current pending transmission requests.

2. Poll the TCR register to check current pending cancellation requests.
    a. Transmit cancellation for a buffer (TXB_i) can be requested only if there is a corresponding pending transmission request set in the TRR register.
    b. If there is already a pending cancellation request for TXB_i, no action is required and the host should wait (by poll/interrupt) until the core serves a cancellation request for TXB_i.
3. If the TXB_i buffer has a pending transmission request but no pending cancellation request, then transmit cancellation can be requested as follows:
    a. Enable interrupt generation as required.
    b. Set the required CAN.TCR [CRnn] bit(s). The host can request the cancellation of many buffers in one write to the TCR register.
    c. Wait for the interrupt or poll the TCR register to determine the cancellation status.
4. The controller clears the bit in the TCR register when the respective buffer transmit cancellation request is completed.
5. The controller also clears the corresponding bit in the TRR register when cancellation is performed.

Controller Actions

1. The controller performs the cancellation of a buffer immediately except:
    a. When the buffer is locked by the transfer layer for transmission on the CAN bus. In this case, cancellation is performed at the end of the transmission irrespective of whether the transmission is successful or not (arbitration loss or error).
    b. When the core is performing a scheduling round to find out the next buffer for transmission. In this case, cancellation is performed after the scheduling round is finished.
2. The controller clears the respective bits in the TCR and TRR registers when cancellation is done.
3. If enabled through IETCS and IER, the CAN.ISR [TXRCS] bit is set (when the controller clears the bit in the TCR register) and an interrupt is generated.

Message Reception

Whenever a new message (that passes the required filtering) is stored into the RX buffer, the controller updates the respective fill level field of the FSR register and sets CAN.ISR [RXOK].

Software Actions

1. As per the requirement, program the RX Buffer Watermark register to set full water marks and the [RXFP] field (the RX Buffer Watermark Register can be set/changed only when [CEN] = 0).

2. If required, enable [RXOK] and [RX] Overflow interrupt generation.
3. The new message availability can be determined by polling the FSR register or by a watermark full interrupts indication.
4. Read a new message (from RX Buffer 0 or RX Buffer 1) starting from its respective read index location (given in the FSR register field).
5. After reading the message, write the FSR register by setting the respective [IRI] bit to 1. This enables the core to increment the respective read index field by +1 and updates the corresponding fill level in the FSR register. If the fill level is 0, setting the [IRI] bit has no effect.
6. Repeat steps 3 through 5 until all messages are read from both RX Buffer 0 or RX Buffer 1.

Controller Actions

1. When a message is successfully received, the core writes the timestamp and matched filtered index field of the received message element.
2. The controller increments the fill level of its respective RX Buffer in the FSR register by 1 after every successful receive (without error and message passes filtering scheme).
3. The fill level is also updated by the core after the host writes the [IRI] bit of the respective RX buffer in the FSR register.

Acceptance Filters

Each acceptance filter has an acceptance filter mask register and an acceptance filter ID register. There are 32 acceptance filters.

RX Buffer Usages

Figure 72: **Normal Operation (RX Buffer 0)**

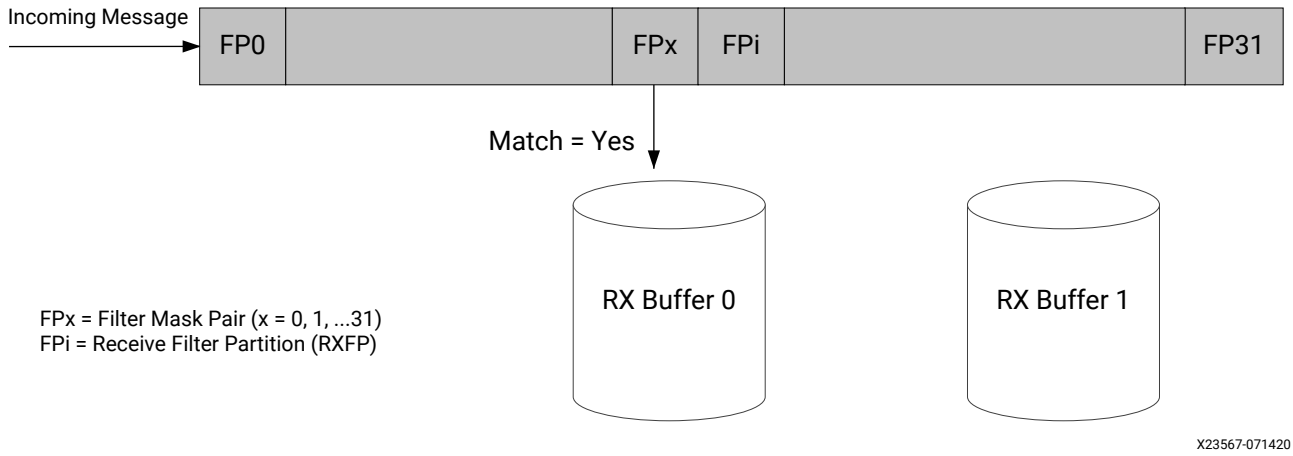


Figure 73: **Normal Operation (RX Buffer 1)**

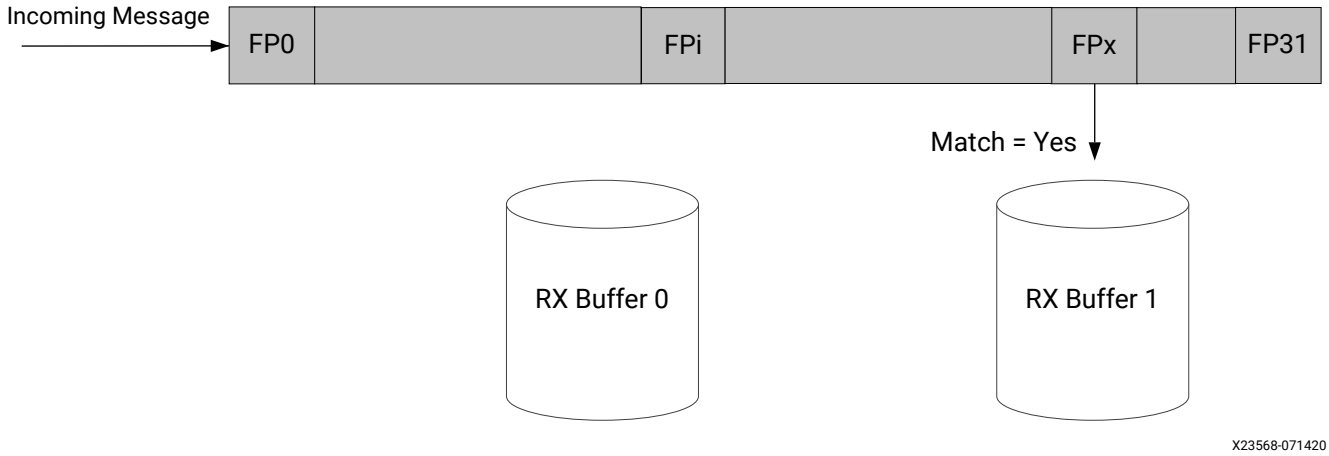


Figure 74: **Message Drop When RX Buffer 0 Full and Match = Yes**

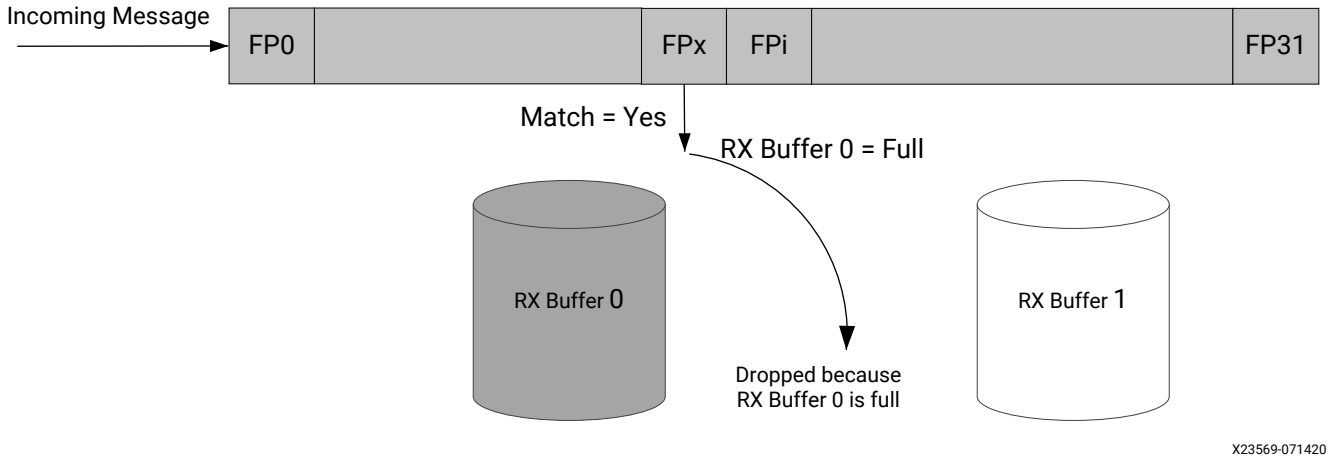
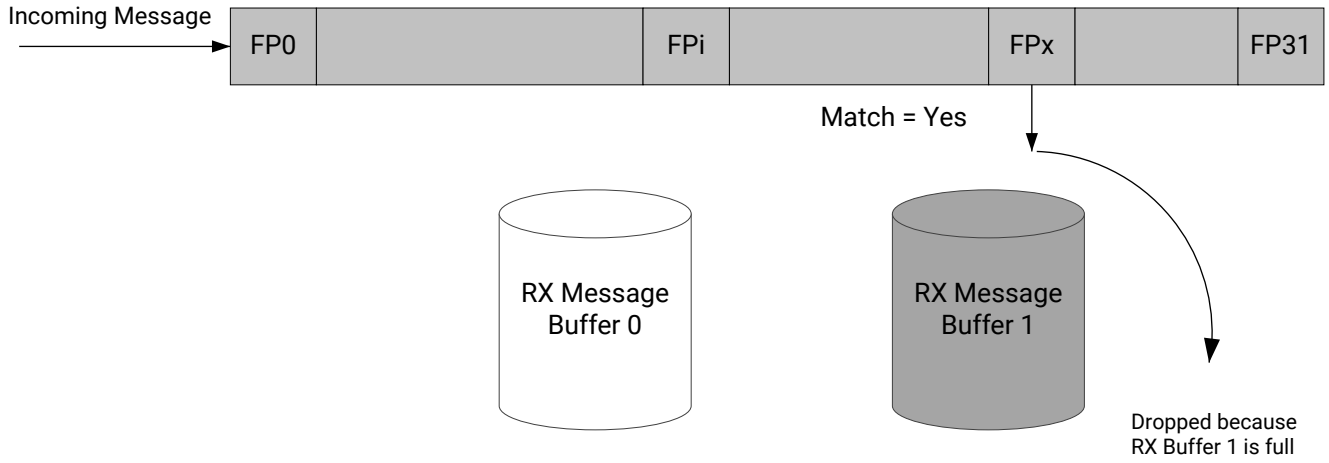


Figure 75: **Message Drop When RX Buffer 1 Full and Match = Yes**



***Note*:** If all UAF bits are set to 0, the received messages are not stored in any RX buffer.

IMPORTANT! *Ensure proper programming of the [IDE] bit for standard and extended frames in the Mask register and ID register. If the [IDE] bit is set to 0, it is considered to be a standard frame ID check. Consequently, if the standard ID bits of the incoming message match the respective bits of the filter ID (after applying Mask register bits), the message is stored.*

Disabled RX Buffer

Acceptance filtering is performed in this sequence:

1. The incoming identifier is masked with the bits in the acceptance filter mask register.
2. The Acceptance Filter ID register is also masked with the bits in the acceptance filter mask register.
3. Both resulting values are compared.
4. If both these values are equal, the message is stored in the RX buffer 0.
5. Acceptance filtering is processed by each of the defined filters. If the incoming identifier passes through any acceptance filter, the message is stored in the RX buffer 0.

***Note*:** RX buffer-1 can be disabled (i.e., stop routing messages to RX buffer 1) by programming [RXFP] as `31d` (in the RX buffer watermark register).

Enabled RX Buffer

In this case, the [RXFP] field (in the RX Buffer Watermark register) along with the Acceptance Filter (Control) register determines whether received messages are stored in RX buffer 0 or in RX buffer 1. In this case, note that the [RXFP] field should be less than `31d`.

1. The incoming identifier is masked with the bits in the Acceptance Filter Mask register.

2. The Acceptance Filter ID register is also masked with the bits in the Acceptance Filter Mask register.
3. Both resulting values are compared.
4. If both these values are equal and the matched filter index is less than equal to the [RXFP] field, the message is stored in RX buffer 0.
5. Otherwise, if both these values are equal and the matched filter index is greater than the [RXFP] field, the message is stored in RX buffer 1.

Note: The ID match process is a sequential process. It starts from the lowest enabled filter and stops at first match. Consequently, if an incoming message fulfills condition 4 but RX buffer 0 is full, the message is dropped (irrespective of RX buffer 1 status) and RX buffer 0 overflow is indicated.

Similarly, if an incoming message fulfills condition 5 and RX buffer 1 is full, the message is dropped (irrespective of the RX buffer 0 status) and RX buffer 1 overflow is indicated.

Note: If all [UAF] bits are set to 0, the received messages are not stored in any RX buffer.

Note: Filter pair registers are stored in the RAM. The host must ensure that each used filter pair is properly initialized. Asserting a software reset or system reset does not clear these register contents.

Note: The host must initialize/update/change the filter pair only when the corresponding UAF is 0.

IMPORTANT! *Ensure proper programming of the [IDE] bit for standard and extended frames in the Mask Register and ID Register. If the [IDE] bit in the Mask Register is set to 0, it is considered to be a standard frame ID check only. Consequently, if the Standard ID bits of the incoming message match with the respective bits of the Filter ID (after applying the Mask register bits), the message is stored.*

Register Reference

The controller is configured and data is accessed via its own register set (CANFD). The controller is also controlled by system-level registers.

- Control and Status
- Message Space Data
- Clock and reset control using Processing System Manager

Control and Status

- CANFD control and status registers:
  - Status, mode configuration
  - Errors, interrupts, watermarks

The following table lists the CANFD Control and Status registers.

Table 113: **CANFD Control and Status Registers**

Name	Offset	Type	Description
SRR	0x0000	Mix	Software reset and enable
MSR	0x0004	RW	Mode select
BRPR, BTR	0x0008, 0x000C	RW	Arbitration phase baud rate prescaler and bit timing
ECR, ESR	0x0010, 0x0014	R, WTC	Error counter and status
SR	0x0018	R	Mode and bus status
ISR IER ICR	0x001C 0x0020 0x0024	R RW W	Interrupt status, enable, and clear
TSR	0x0028	R, W	Timestamp
DP_BRPR DP_BTR	0x0088 0x008C	RW	Data phase baud rate prescalar and bit timing
TRR IETRS	0x0090 0x0094	RWSO RW	TX buffer ready request and interrupt enable
TCR IETCS	0x0098 0x009C	RWSO RW	TX buffer cancel request and interrupt enable
TxE_FSR TxE_WMR	0x00A0 0x00A4	R, W RW	TX event buffer status and watermark
AFR	0x00E0	RW	Acceptance filter control
FSR WMR	0x00E8 0x00EC	R, W RW	RX buffer status and watermark

Message Space Data

The message space includes:

- 32 TX buffers
- 32 RX ID filter-mask pairs
- 32 TX event buffers
- 64-deep message RX buffers

The following table provides CANFD message space register information.

***Note*:** This memory space is implemented in RAM. After a reset, the contents are not cleared, but should be considered invalid.

Table 114: **CANFD Message Space**

Name	Offset	Access Type	Description
32 TX Buffers - ID, DLC and 16 data words			
TB_ID_reg{0:31} TB_DLC_reg{0:31} TB_DW{00:15}_reg{0:31}	0x0100 + 0x0104 + 0x0108 +	RW RW RW	TX buffer identifier, DLC, and data words
32 RX Acceptance Filter - Mask and ID			
AF_MASK_reg{0:31} AF_ID_reg{0:31}	0x0A00 + 0x0A04 +	RW RW	RX acceptance filter mask and identifier
32 TX Event FIFO - ID and Data Length Codes			
TEF_ID_reg{0:31} TEF_DLC_reg{0:31}	0x2000 + 0x2004 +	R R	TX message type and identifier and TX event buffer DLC
32 Message RX Buffer 0			
RB0_ID_reg{0:31} RB0_DLC_reg{0:31} RB0_DW_reg{00:15}{0:63}	0x2100 + 0x2104 + 0x2108 +	R R R	RX buffer 0 identifier, DLC, and data words
32 Message RX Buffer 1			
RB1_ID_reg{0:31} RB1_DLC_reg{0:31} RB1_DW_reg{00:15}{0:63}	0x4100 + 0x4104 + 0x4108 +	R R R	RX buffer 1 identifier, DLC, and data words

System-level Control Registers

There are several registers to control I/O routing, and the APB programming interface.

Table 115: **CAN SLCR Registers**

Description	Register	Bit Fields	Offset Address	Access Type
Reference clock controls	CRL.CAN0_REF_CTRL CRL.CAN1_REF_CTRL	[SRCSEL], [DIVISOR], and [CLKACT]	0x0138 0x013C	RW
Software reset control	CRL.RST_CAN0 CRL.RST_CAN1	[RESET]	0x0328 0x032C	RW
PMC MIO pin multiplexing routing	PMC_IOP_SLCR.MIO_PIN_xx (see MIO Pin Configuration)	[L0_SEL], [L1_SEL], [L2_SEL], and [L3_SEL]	0x0000+	RW
LPD MIO pin multiplexing routing	LPD_IOP_SLCR.MIO_PIN_xx (see MIO Pin Configuration)	[L0_SEL], [L1_SEL], [L2_SEL], and [L3_SEL]	0x0000+	RW

Table 115: **CAN SLCR Registers** *(cont'd)*

Description	Register	Bit Fields	Offset Address	Access Type
MIO bank select: 0: PMC pin bank 1: LPD pin bank	LPD_IOP_SLCR.LPD_MIO_SEL	[CAN0_SEL] [CAN1_SEL]	0x0410	RW
MIO loopback enable: 0: No loopback 1: CAN0 ↔ CAN1	LPD_IOP_SLCR.MIO_LOOPBACK	[CAN0_LOOP_CAN1]	0x200	RW
APB programming interface: 0: no error 1: parity error	LPD_IOP_SLCR.PARITY_ISR	[perr_can0_apb] [perr_can1_apb]	0x0714	RW

I/O Signal Reference

CANFD I/O Signals

The CANFD controller I/O signals are routed to both the PMC and the LPD MIOs and the EMIO interface to the PL.

Table 116: **CANFD Controller MIO Signals**

MIO					EMIO
Signal Name	I/O	PMC Pin MUX	LPD Pin MUX	MIO-at-a-Glance Table	
CAN0_RX CAN1_RX	Input	MIO-at-a-Glance Tables		0	
CAN0_TX CAN1_TX	Output			1	

Gigabit Ethernet MAC

The gigabit Ethernet MAC (GEM) controller provides 10/100/1000 Mb/s interfacing via an RGMII, GMII, or MII interface. There are two controllers, which are located in the LPD with its DMA unit attached to the IOP master switch.

Each controller is operated independently and include a management data input/output (MDIO) interface for its external PHY for use with the RGMII interface. The I/O options include:

- RGMII (v2.0) is routed to the PMC or LPD MIO pins for connection to an external PHY
- GMII and MII are routed to the PL where they can be mapped to GTs or optionally be converted to other protocols using the PL logic
- Diagnostic internal loopback within each controller

Ethernet Specifications

The GEM controller implements several MAC layer specifications and time sensitive clauses:

- MAC layer
  - IEEE 802-2001
  - IEEE 802.3-2002
  - IEEE 802.3-2008
- Time sensitive network (TSN) clauses
  - IEEE 802.1AS Timing and Synchronization for Time-Sensitive Applications
  - IEEE 802.1Qav Credit-Based Shaper
  - IEEE 802.1Qaz Enhanced Transmission Selection
  - IEEE 802.1Qbv Enhancements for Scheduled Traffic
  - IEEE 802.1Qci Pre-Stream Filtering and Policing
- Additional implementation:
  - IEEE Std 1588 precision timestamp protocol
  - IEEE Std 802.1Q VLAN

This chapter contains these sections:

Features

MAC Features

- 10/100 Mb/s full and half duplex
- 1000 Mb/s full duplex
- Priority (Q1) on transmit and receive frames
- Jumbo frames up to 10,240 bytes
- Promiscuous mode, broadcast mode
- Collision detection and enforcement
- Wake on LAN

DMA Features

- 44-bit physical to memory-mapped destinations, or 48-bit virtual address to SMMU
- Descriptor driven with scatter-gather

Common Features

- Automatic pad and cyclic redundancy check (CRC) generation on TX frames
- Automatic discard of RX frames with errors
- Programmable inter-packet gap (IPG) stretch
- Full-duplex flow control with recognition of incoming pause frames and hardware generation of transmitted pause frames
- Address checking logic for four specific 48-bit addresses, four type ID values, promiscuous mode, hash matching of unicast and multicast destination addresses
- VLAN tagging with recognition of incoming VLAN and priority tagged frames
- IPv4 and IPv6 transmit and receive IP, TCP, and UDP checksum offload
  - Checksum offload can be done in the IP instead of the software stack
- Partial store and forward option
- Precision timestamp protocol
- Time sensitive networking (TSN)
- Interrupts for TX/RX, error handling, and wake on LAN

PHY Features

- MDIO programming interface for clause 22 protocol

I/O Features

- Local I/O loopback from TXD to RXD within the controller

Comparison to Previous Generation Xilinx Devices

The Versal™ ACAP GEM controller is similar to the controller in the Zynq® UltraScale+™ MPSoC.

New Features

- Time sensitive network (TSN)
- New RXFIFO high and low-level watermarks use pause frames for RX flow control
- Large segment offload (LSO) WANs added

Removed Features

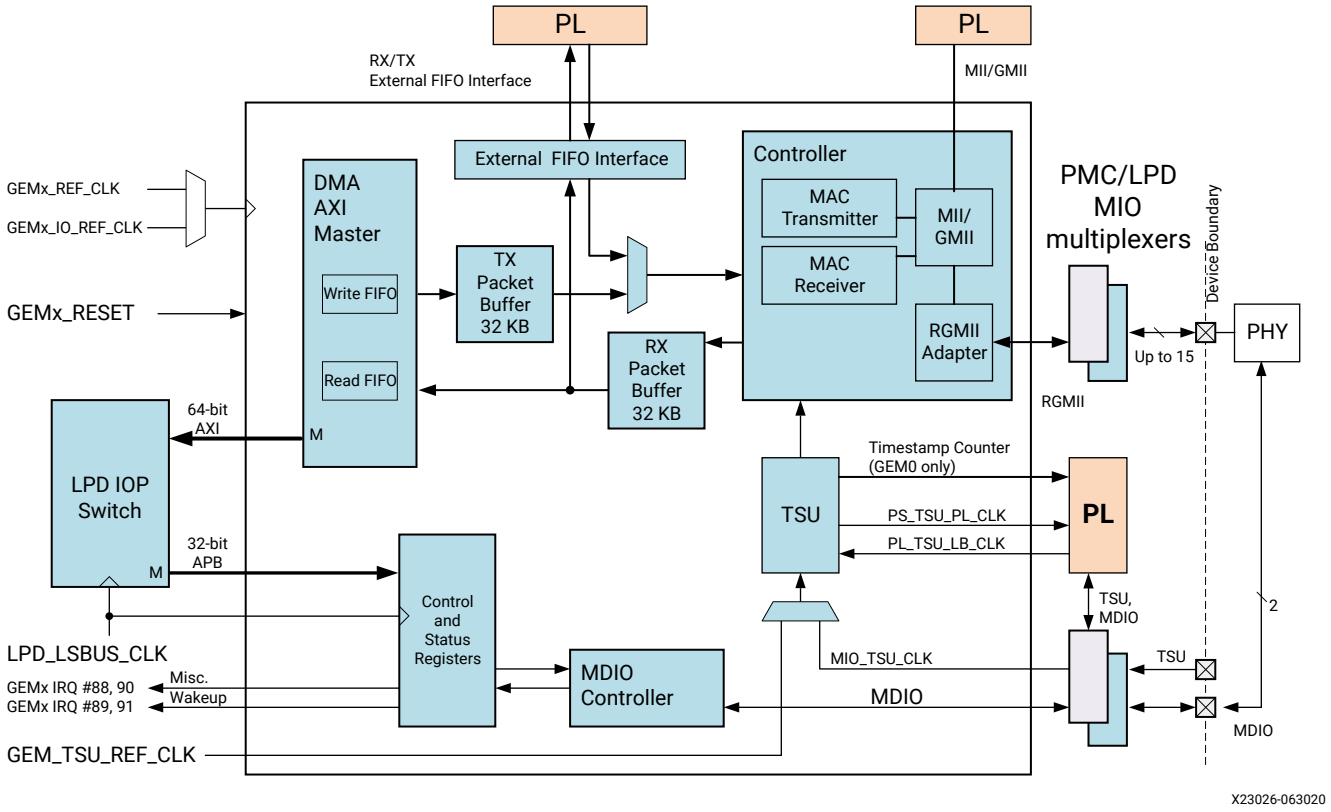
- SGMII interface to PL via EMIO
- 1000 BASE-x physical coding sublayer (PCS)
- Ten-bit interface (TBI) to PL via EMIO

System Perspective

Block Diagram

The high-level block diagram is shown in the following figure.

Figure 76: **GEM High-level Block Diagram**



Functional Units

The main components of the GEM controller are described in this section.

MAC Transmitter

The MAC transmitter takes data from the TX packed buffer, adds a preamble and, if necessary, a pad and frame check sequence (FCS).

Both half-duplex and full-duplex Ethernet modes of operation are supported. When operating in half-duplex mode, the MAC transmitter generates data according to the carrier sense multiple access with collision detect (CSMA/CD) protocol. The start of transmission is deferred if carrier sense (crs) is active. If a collision (col) occurs during transmission, a jam sequence is asserted and the transmission is re-tried after a random back off. The crs and col have no effect in full-duplex mode.

For more information, see MAC Transmitter.

MAC Receiver

The MAC receiver checks for valid preamble, FCS, alignment and length, and presents received frames to the MAC address checking block. Packets are forwarded to the RX packet buffer for the AXI DMA to access. Software can configure the GEM to receive jumbo frames up to 10,240 bytes. It can optionally strip FCS from the received frame prior to transfer to the RX packet buffer. The address checker recognizes a configurable number of maskable source or destination specific 48-bit addresses, can recognize four different type ID values, and contains a 64-bit hash register for matching multicast and unicast addresses as required. It can recognize the broadcast address of all ones, copy all frames and act on external address matching signals.

The MAC receiver can also reject all frames that are not VLAN tagged. The MAC receiver can recognize wake-on-LAN (WOL) events. Address comparison against individual bits of specific address register 1 can be masked by means of the specific address mask register. All other specific address filters are byte maskable.

The MAC receiver supports offloading of IP, TCP, and UDP checksum calculations (both IPv4 and IPv6 packet types supported), and can automatically discard frames with a bad checksum. The MAC receiver can be set up to identify 802.1CB streams and automatically eliminate duplicate frames. Statistics are provided to report counts of rogue and out-of-order frames, latent errors, and timer reset events.

For more information, see MAC Receiver.

Statistics

There are many statistic and status registers that can be read by software using the programming slave interface.

- TX frames
- TX Ethernet traffic
- RX frames
- RX Ethernet traffic

System Interfaces

The GEM controller includes three system interfaces.

AXI Master DMA Interface

The DMA is a master on the LPD IOP AXI interconnect. It generate a 44 or 48-bit address and 64-bit wide data words. When the transaction is routed to the SMMU, then a 48-bit address is used. Otherwise, only lower 44 bits are meaningful.

The DMA is controlled by a descriptor list. Packets are read from memory by the DMA and forwarded to the TX packet buffer for the MAC transmitter using the TX descriptor list. Packets are received from the RX packet buffer and written to system memory using the RX descriptor list.

Interface Clock

The AXI master interface is clocked by the LPD IOP master switch.

APB Programming Interface

The programming interface provides the software access to the memory-mapped control, status, and statistics registers. The registers are listed in Register Reference.

Interface Clock

The APB slave interface is clocked by the LPD_LSBUS_CLK associated with the LPD IOP switch.

PL External FIFO Interface

The MAC transmitter normally interfaces with the TX packet buffer and DMA. As an alternative, the PL can interface directly with the MAC transmitter. Also, the PL can receive packets directly from the MAC receiver, or receive them via the RX packet buffer. When the PL can be configured to read and write the packets, it uses 8-bit datapaths, and several control and status signals. When the external FIFO interface is selected, the PL manages the control, data, and status signals for the memory side of the packet buffers.

The descriptions of the data flows refer to the pathways between the MACs and the AXI DMA accessing system memory. However, if selected, these data flows are to the external FIFO interface instead of to the AXI DMA.

Interface Clock

The external FIFO interface is clocked by TX and RX clocks from the PL.

System Signals

System signals include:

- System Clocks
- Controller Reset
- System Interrupts
- System Error Signal

System Clocks

The controller's reference clock can be sourced from several places:

- LPD clock controller
- EMIO

LPD Clock Controller

The LPD clock controller provides separate reference clocks signals for each GEM controller:

- GEM0_REF_CLK
- GEM1_REF_CLK

These reference clocks are programmed by the LPD_IOP_SLCR.GEM0_CLK_CTRL and GEM1_CLK_CTRL registers.

Controller Reset

The GEM controllers are reset by a POR or system software reset, which are described in Resets. In addition, each controller can be individually reset:

- CRL.GEM0_RESET [RESET] bit
- CRL.GEM1_RESET [RESET] bit

System Interrupts

Each GEM controller generates two types of system interrupts:

- Controller system interrupt
  - Receive and transmit frames completed
  - RX frame errors
- RX wake-on-LAN system interrupt

All of the system interrupts are listed in IRQ System Interrupts.

System Error Signal

The APB interface includes an address decode error detector. If an error is detected, a system error is generated and the APB programming interface can optionally assert the SLVERR error signal back to the source and assert the address decode error interrupt.

I/O Interfaces

The controller provides I/O signals for two I/O interface paths:

- GMII/MII to the EMIO PL interface
- RGMII to the LPD MIO pins

GMII/MII Interface

The controller natively includes a GMII/MII interface that is routed to the EMIO PL interface where it can be converted to another format or consumed as is. The interface is also routed to the built-in RGMII adapter.

- EMIO - GMII/MII interface provides 10/100 Mb/s bandwidth

RGMII Interface

The controller includes a GMII to RGMII adapter. The RGMII I/O interface is multiplexed through the LPD MIO for connection to an external PHY. This interface supports the 10/100/1000 Mb/s protocol. See MIO - RGMII.

MDIO to External PHY for RGMII

To support the external PHY for the RGMII interface, the controller includes a master management data input/output (MDIO) interface. The MDIO interface includes clock, data, and output enable signals that are routed from the controller to LPD MIO pins. See MDIO PHY Interface.

Internal Loopback

The controller has an internal loopback from the TXD to RXD signals in the GMII/MII controller.

- Ethernet loopback connecting TXD to RXD within a controller using network_config [loopback_local] control bit

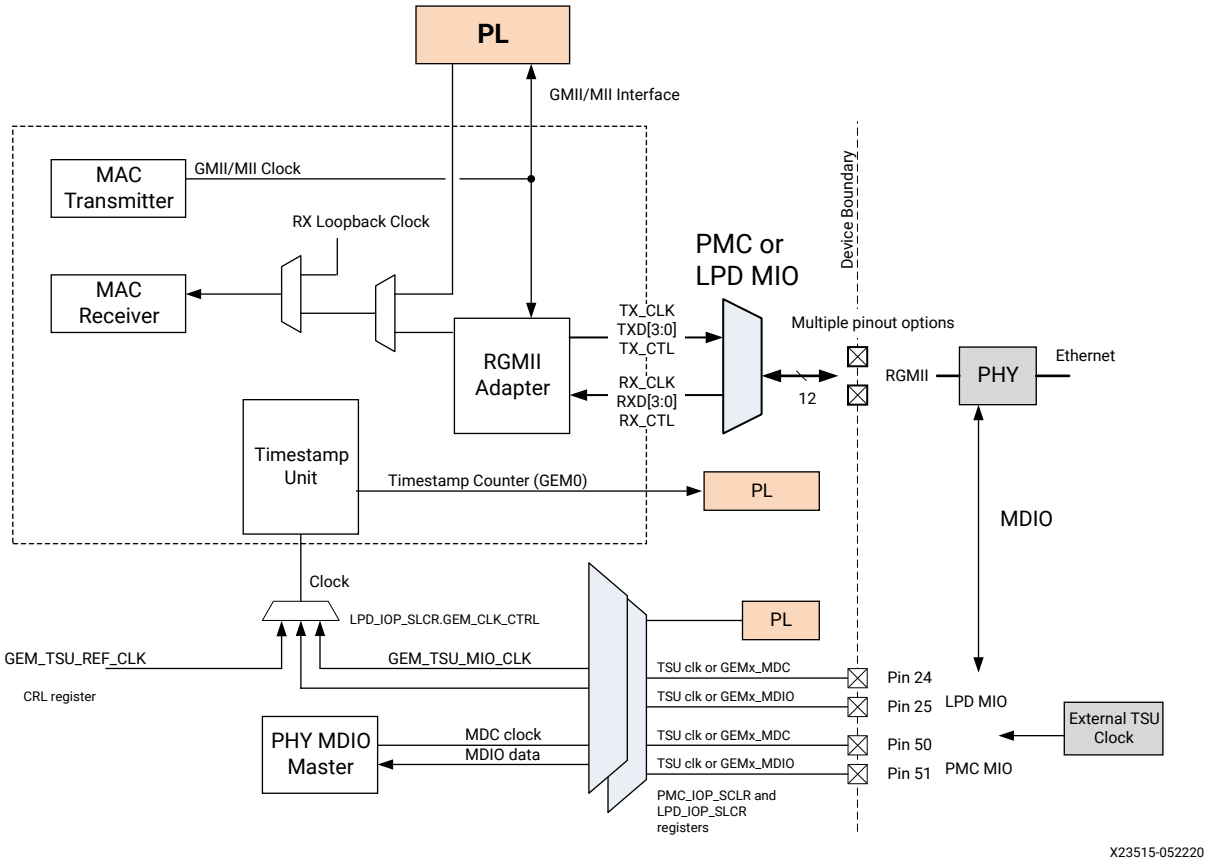
In MAC internal loopback mode, both transmit and receive clock are sourced from the GEM_REF_CLK from the LPD clock controller.

IMPORTANT! *Receive and transmit must be disabled when making the switch into and out of internal loopback because the clocks provided might glitch while switching to the loopback reference clock.*

I/O Block Diagram

The following figure includes the I/O interfaces and signals.

Figure 77: **GEM I/O Interface Block Diagram**



Clocks

There are several clocks associated with the controller:

- Controller reference clock
- Timestamp unit clock
- External FIFO interface clock
- AXI master and APB slave interface clocks

Timestamp Unit Clock

The TSU clock can come from one of several sources:

- LPD clock controller, GEM_TSU_REF_CLK
- External source connected to an LPD MIO pin
- EMIO

Programming Model

Software accesses the GEM 32-bit registers to program the controller, manage the DMA, monitor the FIFOs, collect statistics, manage the external PHY, provide address filtering and type ID matching, control the precision time protocol (PTP), and generate TX pause frames.

The descriptor-driven DMA controller moves data packets between system memory and the packet buffers. The descriptors provide several options including scatter-gather functionality. The DMA connects to system memory through its 64-bit AXI bus master on the LPD IOP master switch.

To transmit frames, software creates descriptors and data buffers in the system memory. The data buffers are fetched by the DMA and written into the TX packet buffer. The packet buffers are consumed by the MAC transmitter to generate Ethernet frames.

To receive frames, software programs the DMA RX descriptors to read packets put in the RX packet buffer by the MAC receiver. The DMA takes packets from the buffer and writes memory packets to system memory using the RX descriptors.

Summary of programming features:

- Memory mapped control registers
- Configuration
- Direct control
- Programmable DMA with descriptor words
- Autonomous monitoring and responses

Modes and States

The controller has several modes and options that can be enabled.

- 10/100 Mb/s via GMII or MII on the PL EMIO interface
- 10/100/1000 Mb/s via the RGMII interface on the LPD MIO pins

***Note*:** The timestamp unit clock must be active for GEM to operate.

Diagnostics

The controller also has a diagnostic mode. See Controller Reset.

- Loopback

Startup

10/100/1000 Operating Modes

The GEM operating mode is controlled by the network configuration register as shown in the following table.

Table 117: **GEM Operating Mode**

Operating Mode	Network_config		I/O Interface Options
	[Speed]	[Gigabit_mode_enable]	
10 Mb/s	0	0	RGMII/GMII/MII
100 Mb/s	1	0	RGMII/GMII/MII
1000 Mb/s	x	1	RGMII

Memory Packet Descriptors

The memory packets are transferred between the controller and system memory using descriptors. The descriptors are used by the packet buffer and the AXI DMA.

- Descriptor length
- Descriptor programming

Descriptor Length

The length of a descriptor entry depends on the interface (AXI or external FIFO interface) and if the timestamp feature is enabled. Every buffer descriptor entry has the same number of words for a given mode. This functionality applies to RX and TX descriptors.

Table 118: **GEM Descriptor Length**

Word Length	Bit Length	64-bit Addressing (AXI Master Interface)	32-bit Addressing (External FIFO Interface)	Timestamp Capture
2 words	64	~	Yes	~
4 words	128	Yes	~	~
4 words	128	~	Yes	Yes
6 words	192	Yes	~	Yes

DMA AXI Master

The DMA controller accesses system memory using a 44-bit address AXI master on the LPD IOP switch. If the transaction is routed through the FPD SMMU, then 48 address bits are used. The DMA controller processes descriptor tables in system memory to manage data between system memory buffers and the RX and TX packet buffers. The descriptor tables include information for the DMA to gather data from one or more memory locations in to one or more packet buffers for the MAC transmitter to create Ethernet frames.

Packet Buffer DMA

The DMA uses separate transmit and receive descriptor lists. Each descriptor entry has parameters that point to a memory location, specify the data buffer size, and indicate if the entry is a start for the frame (SOF) or end of frame (EOF). Multiple descriptor entries pointing to memory locations enable Ethernet packets to be broken up and scattered about the memory space.

The DMA and packet buffers include the following advantages:

- 64-bit AXI data bus width
- Maximum line rate by storing multiple frames in the packet buffer
- Efficient use of the AXI interface with FIFOs and bursting
- Full and partial store with forward
- Transmit TCP/IP checksum offload
- Priority queuing
- When a collision on the line occurs during transmission, the packet is automatically reaccessed directly from the packet buffer rather than having to re-fetch through the AXI interface
- Received error packets are automatically dropped before any of the packets are presented to the AXI, reducing AXI activity
- Manual RX packet flush capabilities

- Optional RX packet flush when the AXI becomes bandwidth limited

AXI Coherency and Bufferability

The AXI transaction requests can be routed directly into the FPD main switch and then to system memory via the NoC, or to system memory via the CCI for coherency and memory address translation.

Burst Transactions

The AXI master interface data width and word size is 64 bits. The burst size can be programmed to 1, 4, 8, 16, or 256 words using gem.dma_config [amba_burst_length].

The AXI DMA master always uses INCR type accesses. When performing data transfers, the burst length used can be programmed using bits [4:0] of the DMA configuration register. Either single or fixed length incrementing bursts up to a maximum of 256 are used as appropriate.

- 1xxxx: Attempt to use bursts of up to 16
- 01xxx: Attempt to use bursts of up to 8
- 001xx: Attempt to use bursts of up to 4
- 0001x: Use single word
- 00001: Use single word
- 00000: Attempt to use bursts of up to 256

Transaction Routing and Coherency

The AXI memory transactions have several attributes controlled by the LPD_IOP_SLCR register set. The transactions can be coherent to the APU L2-cache by routing the transaction to the CCI via the SMMU. The AXI coherency signals are programmed to request the caching policy. This programming is used for both reads and writes (example is for GEM 0 controller):

- LPD_IOP_SLCR.GEM0_IOP_INTERCONNECT_ROUTE [GEM0] routes the transactions through the CCI, which is required for coherency to the APU L2-cache
- LPD_IOP_SLCR.GEM0_IOP_COHERENT_CTRL [GEMx_AXI_COH] defines the AxUSER signals for the caching policy used by the CCI AXI-Lite connection

The encoding of [GEMx_AXI_COH] controls DMA AXI transaction coherency with respect to the APU L2-Cache and transaction buffers on the interconnect.

When the transaction bypasses the CCI, the [GEMx_AXI_COH] is only used to define the bufferability of the transaction. The coherency settings are ignored.

Transmit Dataflow

Packet Buffer TX Functionality

The software initiates the TX frames with some exceptions. In normal operation, the transmitter packet buffer continuously requests data from the TXFIFO to keep the packet buffer full. The transmitter packet buffer continues to attempt to fetch frame data from the DMA until the packet buffer itself is full, it then attempts to maintain a full level. To accommodate the status and statistics associated with each frame, three status words per packet are reserved at the end of the packet data. This can be used for the flow of data and to generate interrupts.

Operations

If any errors occur on the AXI transaction while reading the transmit frame, the fetching of packet data from system memory is halted. The MAC transmitter continues to fetch packet data, thereby emptying the packet buffer, and allowing any good non-errored frames to be transmitted successfully. When these frames are fully transmitted, the status/statistics for the errored frame is updated and software is informed through an interrupt that an AXI error occurred. The error is reported in the correct packet order.

The transmit packet buffer only attempts to read more frame data from the system memory when space is available in the packet buffer memory. If space is not available, the AXI interface must wait until the packet fetched by the MAC completes transmission and is subsequently removed from the packet buffer memory.

When full store and forward mode is active, and a single frame is fetched that is too large for the packet buffer memory, the frame is flushed and the DMA is halted with an error status. A complete frame must be written into the packet buffer before transmission can begin, and therefore the minimum packet buffer memory size should be chosen to satisfy the maximum frame to be transmitted in the application.

When the complete transmit frame is written into the packet buffer memory, a trigger is sent across to the MAC transmitter, which then begins reading the frame from the packet buffer memory. Because the whole frame is present and stable in the packet buffer memory, an underflow of the transmitter is not possible.

Half-duplex Mode

In half-duplex mode, the frame is kept in the packet buffer until notification is received from the MAC that the frame data has either been successfully transmitted or can no longer be retransmitted (too many retries in half-duplex mode). When this notification is received, the frame is flushed from memory to make room for a new frame to be fetched from AXI system memory.

Full-duplex Mode

In full-duplex mode, the frame is removed from the packet buffer in real time. Other than underflow, the only MAC related errors that can occur are due to collisions during half-duplex transmissions. When a collision occurs, the frame still exists in the packet buffer memory, and can be retried directly from there. Only when the MAC transmitter has failed to transmit after sixteen attempts is the frame finally flushed from the packet buffer.

TX Packets

The TX buffers become packets that are sent through the TX packet buffer. The MAC transmitter uses the descriptors, which provide the necessary information about and a pointer to the TX buffers.

The maximum frame size is 1522 bytes by default and 10,240 bytes (with jumbo frame support) bytes and the minimum frame size is 64 bytes.

In the transmit direction, the DMA (or FIFO interface) continues to provide TX packet data up to a limit of 2048 packets. The interface monitors the TX buffer full condition to avoid overwrites. The maximum useful size of the TX packet buffer is 32 KB.

The DMA uses the packet buffers to hold packets for both transmit and receive paths. While the packet is in the buffer, the controller decides what to do with it. If it is corrupt or does not pass the filters, it is discarded, and left unused in the buffer. This has several performance advantages:

- Retry collided TX frames: the MAC transmitter can re-read the packet directly from the buffer, which saves system interconnect bandwidth (or FIFO interface activity)
- Process the transmit IP/TCP/UDP checksum generation offload
- Generate the checksum before determining the fate of the frame

TX Descriptor Entry Words

The following table includes details of the transmit buffer descriptor list.

Table 119: **GEM TX Descriptor, Words 0 and 1**

Bit	Function
Word 0	
31:0	Byte address of buffer.
Word 1	
31	Used: must be zero for the controller to read data to the transmit buffer. After it is successfully transmitted, the controller sets this bit to one for the first buffer of a frame. Software must clear this bit before the buffer can be used again.
30	Wrap: marks the last descriptor in the transmit buffer descriptor list. This can be set for any buffer within the frame.

Table 119: **GEM TX Descriptor, Words 0 and 1** *(cont'd)*

Bit	Function
29	Retry limit exceeded, transmit error detected.
28	Always set to 0.
27	Transmit frame corruption due to AXI error: set if an error occurs midway while reading through the transmit frame from the AXI, including RESP errors, and buffers exhausted mid-frame. If the buffers run out during transmission of a frame, then transmission stops, the FCS is incorrect, and tx_er is asserted.
26	Late collision, transmit error detected. Late collisions force this status bit to be set in gigabit mode.
25:24	Reserved.
23	For extended buffer descriptor mode. This bit indicates a timestamp is captured in the buffer descriptor. Otherwise the bit is reserved.
22:20	Transmit IP/TCP/UDP checksum generation offload errors: `000b`: No error. `001b`: The packet is identified as a VLAN type, but the header is not fully complete, or has an error in it. `010b`: The packet is identified as a SNAP type, but the header is not fully complete, or has an error in it. `011b`: The packet is not of an IP type, or the IP packet was invalidly short, or the IP is not of type IPv4/IPv6. `100b`: The packet is not identified as VLAN, SNAP, or IP. `101b`: Non-supported packet fragmentation occurred. For IPv4 packets, the IP checksum is generated and inserted. `110b`: Packet type detected is not TCP or UDP. TCP/UDP checksum is therefore not generated. For IPv4 packets, the IP checksum is generated and inserted. `111b`: A premature end of packet is detected and the TCP/UDP checksum cannot be generated.
19:17	Reserved.
16	No CRC to be appended by the MAC. When set this bit implies that the data in the buffers already contains a valid CRC and no CRC or padding is appended to the current frame by the MAC. This control bit must be set for the first buffer in a frame and is ignored for the subsequent buffers of a frame. This bit must be clear when using the transmit IP/TCP/UDP checksum generation offload, otherwise checksum generation and substitution does not occur.
15	Last buffer, this bit (when set) indicates that the last buffer in the current frame is reached.
14	Reserved.
13:0	Length of buffer.

Table 120: **GEM TX Descriptor Word Summary**

64-bit Addressing for AXI DMA Interface	32-bit Addressing for External FIFO Interface	Field	Description
Word 0	Word 0	31:0	Byte address of buffer
Word 1	Word 1	31:0	Miscellaneous fields
Word 2		31:0	Upper 32-bit address of the data buffer
Word 3	Word 2	31:0	Not used
Word 4	Word 3	31:30 29:0	Timestamp seconds [1:0] Timestamp, nanoseconds

Table 120: **GEM TX Descriptor Word Summary** *(cont'd)*

64-bit Addressing for AXI DMA Interface	32-bit Addressing for External FIFO Interface	Field	Description
Word 5		31:4 3:0	Unused Timestamp seconds [5:2]

TX Descriptor Processing

Each transmit frame is stored in one or more memory buffers. Zero length memory buffers are allowed. The maximum number of buffers permitted for each TX frame is 128. The size of the descriptor entry is described in Descriptor Length. To transmit frames, the buffer descriptors must be initialized by writing an appropriate byte address to bits [31:0] in the first word of each descriptor list entry.

The second word of the TX descriptor is initialized with control information that indicates the length of the frame, whether the MAC is to append CRC, and whether the buffer is the last buffer in the frame.

After transmission, the status bits are written back to the second word of the first buffer along with the used bit. Bit [31] is the used bit that, if transmission is to take place, must be zero when the control word is read. It is written to one once the frame is transmitted. Bits [29:20] indicate various transmit error conditions. Bit [30] is the wrap bit, which can be set for any buffer within a frame. When no wrap bit is encountered, the queue pointer continues to increment.

The transmit-buffer queue base address register can only be updated while transmission is disabled or halted. Otherwise, any attempted write is ignored. When transmission is halted, the transmit-buffer queue pointer maintains its value. Consequently, when transmission is restarted, the next descriptor read from the queue is from immediately after the last successfully transmitted frame. While transmit is disabled, bit [3] of the network control is set Low, the transmit-buffer queue pointer resets to point to the address indicated by the transmit-buffer queue base address register. Disabling receive does not have the same effect on the receive-buffer queue pointer.

When the transmit queue is initialized, transmit is activated by writing a 1 to the transmit start bit [9] of the network control register. Transmit is halted when the used bit of the buffer descriptor is read, a transmit error occurs, or by writing to the transmit halt bit of the network control register.

Transmission is suspended if a pause frame is received while the pause enable bit is set in the network configuration register. Rewriting the start bit while transmission is active is allowed. This is implemented with a transmit_go variable, which is read from the transmit status register at bit [3].

The transmit_go variable is reset when the following occurs.

- Transmit is disabled
- A buffer descriptor's ownership bit set is read
- Bit [10], tx_halt_pclk, of the network control register is written
- There is a transmit error due to too many retries, late collision (gigabit mode only), or a transmit under-run

To set transmit_go, write to bit [9], tx_start_pclk of the network control register.

Transmit halt does not take effect until any ongoing transmit finishes. The entire contents of the frame are read into the transmit packet buffer memory, any retry attempt is replayed directly from the packet buffer memory rather than re-fetching it through the AXI. If a used bit is read mid-way through transmission of a multi-buffer frame, the bit is treated as a transmit error. Transmission stops, tx_er is asserted, and the FCS is bad. If transmission stops due to a transmit error or a used bit being read, transmission is restarted from the first buffer descriptor of the frame being transmitted when the transmit start bit is rewritten.

MAC Transmitter

The MAC transmitter can operate in either half-duplex or full-duplex mode, and transmits frames in accordance with the Ethernet IEEE Std 802.3. In half-duplex mode, the CSMA/CD protocol is followed.

TX frame assembly starts by adding the preamble and the start frame delimiter. The packets are taken from the TXFIFO.

For short packets, padding is added to take the frame length to 60 bytes. CRC is calculated using an order 32-bit polynomial. This is inverted and appended to the end of the frame taking the frame length to a minimum of 64 bytes. If the [No_CRC] descriptor bit is set =1 of the last buffer descriptor of a TX frame, neither pad nor CRC are appended. The [No_CRC] bit can also be set through the FIFO.

In full-duplex mode (at all data rates), frames are transmitted immediately. Back-to-back frames are transmitted at least 96-bit times apart to check the inter-packet gap.

In half-duplex mode, the transmitter checks carrier sense. If asserted, the transmitter waits for a signal to become inactive, and then starts transmission after the inter-packet gap of 96-bit times.

Collisions in Half-duplex Mode

If the collision signal is asserted during transmission, the transmitter transmits a jam sequence of 32 bits taken from the data register and then retries transmission after the backoff time has elapsed. If the collision occurs during either the preamble or SFD, then these fields are completed prior to generation of the jam sequence.

The backoff time is based on an XOR of the 10 least significant bits of the data coming from the packet buffer and a 10-bit value from the pseudo-random number generator. The number of bits that are actually used depends on the number of collisions seen. After the first collision, one bit is used to determine the backoff time. After the second collision, two bits are used. This continues up to a maximum of 10 bits for the 10th through 16th collision. When a frame transmits without a collision, the number of bits used for a collision starts back at 1.

After 16 collisions in a row, an error is indicated and no further TX attempts are made, which is in accordance with the truncated binary exponential backoff algorithm.

In 10/100 Mb/s mode, both collisions and late collisions are treated identically (backoff and retry are performed up to 16 times). When operating in 1000 Mb/s mode, late collisions are treated as an exception and the transmission is aborted without a retry. This condition is reported in the transmit buffer descriptor word [1] (late collision, bit [26]) and also in the transmit status register (late collision, bit [7]).

An interrupt can also be generated (if enabled) when this exception occurs, and bit [5] in the interrupt status register is set.

When bit [28] is set in the network configuration register, the IPG can be stretched beyond 96 bits depending on the length of the previously transmitted frame and the value written to the stretch_ratio register. The least significant 8 bits of the stretch_ratio register multiply the previous frame length (including preamble) and the next significant 8 bits (+1 so as not to get a divide by zero) divide the frame length to generate the IPG.

IPG stretch only works in full-duplex mode and when bit [28] is set in the network configuration register. The stretch_ratio register cannot be used to shrink the IPG below 96 bits.

TX Broadcast Frames

Broadcast frames are transmitted with a destination address of all 1s and used to communicate with all nodes on a network.

TX Pause Frame

Automatic transmission of pause frames is supported through the transmit pause frame bits of the network control register and from the external input signals tx_pause, tx_pause_zero, and tx_pfc_sel. If either bit [11] or bit [12] of the network control register is written with a logic 1, or if the input signal tx_pause is toggled when tx_pfc_sel is Low, an IEEE Std 802.3 pause frame is transmitted providing full duplex is selected in the network configuration register and the transmit unit is enabled in the network control register.

Pause frame transmission occurs immediately if transmit is inactive or if transmit is active between the current frame and the next frame due to be transmitted.

Transmitted pause frames comprise of the following:

- A destination address of 01-80-C2-00-00-01.
- A source address taken from specific address register 1.
- A type ID of 88-08 (MAC control frame).
- A pause opcode of 00-01.
- A pause quantum register.
- Fill of 00 to take the frame to the minimum frame length.
- A valid FCS.

The pause quantum used in the generated frame depends on the trigger source for the frame.

- If bit [11] is written with a one, the pause quantum is taken from the transmit pause quantum register. The transmit pause quantum register resets to a value of `0xFFFF` giving maximum pause quantum as the default.
- If bit [12] is written with a one, the pause quantum is zero.
- If the tx_pause input is toggled, tx_pfc_sel is Low and the tx_pause_zero input is held Low until the next toggle, the pause quantum is taken from the transmit pause quantum register.
- If the tx_pause input is toggled, tx_pfc_sel is Low and the tx_pause_zero input is held High until the next toggle, the pause quantum is zero.

After transmission, a pause frame transmitted interrupt is generated (bit [14] of the interrupt status register) and the only statistics register incremented is the pause frames transmitted register. Pause frames can also be transmitted by the MAC using normal frame transmission methods.

Quantum Time Base

The quantum value for transmitting a pause frame depends on the trigger source for the pause frame.

The quantum value is either zero or takes its value from the gem.tx_pause_quantum register.

- If bit [11] is written with a one, the pause quantum is taken from the transmit pause quantum register. The transmit pause quantum register resets to a value of `0xFFFF` giving maximum pause quantum as the default.
- If bit [12] is written with a one, the pause quantum is zero.
- If the tx_pause input is toggled, tx_pfc_sel is Low and the tx_pause_zero input is held Low until the next toggle, the pause quantum is taken from the transmit pause quantum register.
- If the tx_pause input is toggled, tx_pfc_sel is Low and the tx_pause_zero input is held High until the next toggle, the pause quantum is zero.

After transmission, a pause frame transmitted interrupt is generated (bit [14] of the interrupt status register) and the only statistics register incremented is the pause frames transmitted register. Pause frames can also be transmitted by the MAC using normal frame transmission methods.

Receive Dataflow

The MAC receiver divides the frames into one or more packets. Each packet includes status and statistics.

RX data packets are routed through the packet buffer and then managed by the RX descriptors.

RX Packets

Packets Held in Packet Buffer

The RX packets are held in the packet buffer until the MAC receiver determines what to do with it. If a packet is corrupt or does not pass the RX filter criteria, the packet is discarded. This has several performance advantages:

- RX packets with errors can be discarded before propagating further, which saves system interconnect bandwidth and the need for the device driver to discard them
- Process the IP/TCP/UDP checksum generation offload
  - Generate the checksum before determining the fate of the frame

Packet Buffer Overflow

The RX packet buffer can overflow with packets and generate an interrupt when data is received, but there is not enough room to store it. An overflow also occurs if the limit of 2048 packets is breached. The maximum usable size of the packet buffer is 32 KB.

RX Packet Flow Monitoring

Frames with errors are flushed from the packet buffer memory, good frames are pushed onto the DMA AXI interface.

The packet buffer monitors the data flow from the MAC receiver to create packet pushes into the packet buffer. At the end of the received frame, the status and statistics information are stored along side the packet for use when the frame is read out.

Good Frame

The DMA only begins to fetch packets from the packet buffer when the status and statistics for the Ethernet frame are available. If the frame has a good status, the three status and statistics words of information are used to read the frame from the packet memory and written to system memory by the DMA. After the last frame data is transferred to the packet buffer, the status and statistics are updated to the controller's registers.

Bad Frame

When programmed in full store and forward mode, if the frame has an error, the frame data is immediately flushed from the packet buffer memory allowing subsequent frames to use the newly opened space. The status and statistics for bad frames are written to the system buffer and used to update the controller's status and statistics registers.

To accommodate the status and statistics associated with each frame, three words per packet are reserved at the end of the packet data. When a packet is bad and is dropped, the status and statistics is the only information stored for that packet.

The packet buffer can detect a full condition and an overflow condition can also be detected. If this occurs, subsequent packets are dropped and an overflow interrupt is raised.

RX Descriptor Words

Received frames with optional FCS are written to receive buffers in system memory. The memory start location for each receive buffer is stored in the receive buffer descriptor table at an address location pointed to by the value in the receive-buffer queue pointer registers.

Each receive buffer start location is a word address. The start of the first memory buffer in a frame can be offset by up to three bytes depending on the value written to bits [14] and [15] of the network configuration register. If the start location of the AXI buffer is offset the available length of the first AXI buffer is reduced by the corresponding number of bytes.

There are six descriptor words per entry to provide a 44 or 48-bit address for the DMA AXI master interface, see Descriptor Length.

Table 121: **GEM RX Descriptor, Word 0**

Bit	Description
31:3	Starting RX memory buffer address, bits [31:3]. Bits [47:32] are held in descriptor entry word 3.
2	Timestamp enable: 0: None 1: Valid timestamp
1	Wrap enable: 0: No wrap 1: Wrap

Table 121: **GEM RX Descriptor, Word 0** *(cont'd)*

Bit	Description
0	Data ownership: 0: The controller can write data to the RX buffer 1: The controller sets this bit to 1 once the frame leaves the RXFIFO (written to system memory) Software must clear this bit to 0 before the buffer can be used again.

Table 122: **GEM RX Descriptor, Word 1**

Field	Description
31	Global all ones broadcast address detected.
30	Multicast hash match.
29	Unicast hash match.
28	I/O address match.
27	Specific address register match found, bit [25] and [26] indicate the specific address register that caused the match.
26:25	Address register match indicator: 00: Specific address register 1 match 01: Specific address register 2 match 10: Specific address register 3 match 11: Specific address register 4 match If more than one specific address is matched, only one is indicated with priority 4 down to 1.
24	Indicates different information when the RX checksum offloading is enabled or disabled. • When RX checksum offloading is disabled, bit [24] is cleared and the network configuration type ID register match is found. Bit [22] and bit [23] indicates which type ID register caused the match. • When RX checksum offloading is enabled, bit [24] is set in the network configuration. 0: Frame is not SNAP encoded and/or has a VLAN tag with the CFI bit set 1: Frame is SNAP encoded and has either no VLAN tag or a VLAN tag without the CFI bit set
23:22	Indicates different information when the RX checksum offloading is enabled or disabled. • When RX checksum offloading disabled, bit [24] is cleared in the network configuration type ID register match. The encoded matches are: 00: Type ID register 1 match 01: Type ID register 2 match 10: Type ID register 3 match 11: Type ID register 4 match If more than one type ID is matched, only one is indicated with priority 4 down to 1. • When RX checksum offloading enabled, bit [24] is set in the network configuration. 00: Both the IP header checksum and the TCP/UDP checksum were not checked 01: The IP header checksum is checked and is correct. Both the TCP or UDP checksum were not checked 10: Both the IP header and TCP checksum were checked and were correct 11: Both the IP header and UDP checksum were checked and were correct
21	VLAN tag detected: type ID of `0x8100`. For packets incorporating the stacked VLAN processing feature, this bit is set if the second VLAN tag has a type ID of `0x8100`.
20	Priority tag detected: type ID of `0x8100` and null VLAN identifier. For packets incorporating the stacked VLAN processing feature, this bit is set if the second VLAN tag has a type ID of `0x8100` and a null VLAN identifier.
19:17	VLAN priority: only valid if bit [21] is set.
16	Canonical format indicator (CFI) bit: only valid if bit [21] is set.

Table 122: **GEM RX Descriptor, Word 1** *(cont'd)*

Field	Description
15	End of frame: when set, the buffer contains the end of a frame. If end of frame is not set, then the only valid status bit is start of frame bit [14].
14	Start of frame: when set, the buffer contains the start of a frame. If both bits [15] and [14] are set, the buffer contains a whole frame.
13	Indicates different information when the ignore FCS mode is enabled or disabled. • This bit is zero if ignore FCS mode is disabled. • When ignore FCS mode is enabled, bit [26] is set in the network configuration register. The per-frame FCS status indicates the following: 0: Frame had good FCS 1: Frame had bad FCS and if the ignore FCS mode is enabled, the frame is copied to memory
12:0	These bits represent the length of the received frame that could include FCS if the FCS discard mode is enabled or disabled. • FCS discard mode disabled: Bit [17] is cleared in the network configuration register. The least significant 12 bits for length of frame include FCS. • FCS discard mode enabled: Bit [17] is set in the network configuration register. The least significant 12 bits for length of frame exclude FCS.

Table 123: **GEM RX Descriptor Word Summary**

Word	Field	Description
Word 0	31:0	Timestamp enable, wrap enable, ownership.
Word 1	31:0	Miscellaneous fields.
Word 2	15:0	Upper sixteen AXI address bits [47:32].
	31:16	Not used.
Word 3	31:0	Not used.
Word 4	29:0	Timestamp, nanoseconds.
	31:30	Timestamp, seconds, bits [1:0].
Word 5	3:0	Timestamp, seconds, bits [5:2].
	31:4	Not used.

RX Descriptor Processing

The start location of the RX buffer descriptors must be written with the receive-buffer queue base address before reception is enabled (receive enable in the network control register). After reception is enabled, any writes to the receive-buffer queue base address register are ignored.

When read, it returns the current pointer position in the descriptor list, though this is only valid and stable when receive is disabled.

If the filter block indicates that a frame should be copied to memory, the receive data DMA operation starts writing data into the receive buffer. If an error occurs, the buffer is recovered.

An internal counter represents the receive-buffer queue pointer and it is not visible through the CPU interface. The receive-buffer queue pointer increments by two words after using each buffer. It re-initializes to the receive-buffer queue base address when any descriptor has its wrap bit set.

As receive AXI buffers are used, the receive AXI buffer manager sets bit zero of the first word of the descriptor to logic one, to indicate that the AXI buffer was used.

Software should search through the used bits in the AXI buffer descriptors to determine how many frames are received by checking the start of frame and end of frame bits.

By default, partial store and forward is not enabled; that is, the controller waits for the full packet to be available before forwarding. If the DMA is configured in the packet buffer partial store and forward mode, received frames are written out to the AHB/AXI buffers as soon as enough frame data exists in the packet buffer, which means several full buffers are used before some error conditions can be detected. If a receive error is detected, the receive buffer currently being written is recovered. Previous buffers are not recovered. For example, when receiving frames with CRC errors or excessive length, it is possible that a frame fragment might be stored in a sequence of receive buffers. Software can detect these fragment by looking for start-of-frame bit set in a buffer following a buffer with no-end-of frame bit set.

A properly working 10/100/1000 Ethernet system does not have excessive length frames or frames greater than 128 bytes with CRC errors. When using a default value of 128 bytes for the receive buffer, it is rare to find a frame fragment in a receive AXI buffer because collision fragments are less than 128 bytes long.

Only good received frames are written out of the DMA and no fragments exist in the AXI buffers due to MAC receiver errors. However, there is still the possibility of fragments due to DMA errors. For example, when a used bit is read on the second buffer of a multi-buffer frame.

If bit zero of the receive buffer descriptor is already set when the receive buffer manager reads the location of the receive AXI buffer, the buffer is already used and cannot be used again until the software has processed the frame and cleared bit zero. In this case, the buffer not available bit in the receive status register is set and an interrupt is triggered. The receive resource error statistics register is also incremented.

There is an option to automatically discard received frames when no AXI buffer resource is available. Bit [24] of the DMA configuration register controls this option. By default, the received frames are not automatically discarded. When this feature is off, the received packets remain stored in the packet buffer until an AXI buffer resource becomes available. This can lead to an eventual packet buffer overflow occurs when packets continue to be received because the [0, used] bit of the receive-buffer descriptor is still set.

After a used bit is read, the receive-buffer manager re-reads the location of the receive buffer descriptor every time a new packet is received.

When the DMA is configured in the packet buffer full store and forward mode, a receive overrun condition occurs when the receive packet buffer is full, or if an AXI error occurred.

For a receive overrun condition, the receive overrun interrupt is asserted and the buffer currently being written is recovered. The next frame that is received whose address is recognized reuses the buffer.

A write to bit [18] of the network control register forces a flush of the packet from the receive packet buffer. This only occurs when the RX DMA is not currently writing packet data out to the AXI (that is, it is in an IDLE state). If the RX DMA is active, a write to this bit is ignored.

MAC Receiver

The MAC receiver checks incoming frames for a valid preamble, the frame check sequence (FCS), alignment, and length. The receiver then processes the RX frames and writes packets into the RX packet buffer with status that is to be read by the DMA controller. The MAC also stores the frames destination address for use by the address checking unit.

If the RX frame is too long, a bad frame indication is sent to the RXFIFO. The receiver logic ceases to send data to memory as soon as this condition occurs.

At end of frame reception, the MAC receiver indicates to the DMA controller whether the frame is good or bad. The DMA controller recovers the RX buffer if the frame is bad.

RX Ethernet frames are normally stored with the FCS. Setting network_config [fcs_remove] = 1 causes RX frames to be stored without their FCS. The reported frame length field is reduced by four bytes to reflect this operation.

The MAC receiver updates the status registers:

- Increment the alignment
- CRC (FCS)
- Short frame, long frame
- Jabber or receive symbol errors when any of these exception conditions occur

If network_config [ignore_rx_fcs] is set = 1, then errors are ignored and frames with CRC errors are not discarded, though the frame check sequence errors statistic register is still incremented.

Bit [13] of the receiver descriptor word [1] is updated to indicate the FCS validity for the particular frame. This is useful for applications where individual frames with FCS errors must be identified.

Received frames can be checked for length field error by setting the network_config [length_field_error_frame_discard]. When this bit is set = 1, the receiver compares a frame's measured length with the length field (bytes 13 and 14) extracted from the frame. The frame is discarded if the measured length is shorter. The RX frame length is checked for the range starting at 64 bytes. The upper limit depends on register bit settings. The upper range is:

- 1,518 bytes (normally)
- 1,536 bytes if bit network_config [8] is set =1
- 10,240 bytes if bit network_config [3] is set =1

Each discarded frame increments the 10-bit excessive_rx_length [count] statistics register field.

Filtering

When enabled, the MAC receiver filter determines which frames should be written to the RXFIFO.

Filtering includes:

- State of the I/O matching signals
- Register programming:
  - Specific address
  - Specific type
  - Hash
- Destination address and type field of the field

If the network_config [en_half_duplex_rx, 25] is set = 0, a frame is not placed in the RXFIFO if transmitting in half-duplex mode at the time a destination address is received.

Ethernet frames are transmitted a byte at a time, least significant bit first. The first six bytes (48 bits) of an Ethernet frame make up the destination address. The first bit of the destination address (least significant bit of the first byte) defines the casting:

- 0: Unicast address
- 1: Multicast address

An address of all 1's is a special case of the multicast, broadcast address.

Address Filtering using Four Specific Addresses

The MAC receiver recognizes up to four specific addresses. Each specific address requires two registers, spec_add1_top (two bytes) and spec_add1_bottom (four bytes). The address stored can be specific, group, local, or universal.

When address filtering is enabled, the RX frame destination address is compared against up to four specific addresses stored in registers. If a receive frame address matches an active specific address, the frame is written to the RX packet buffer.

Address filtering is activated when the spec_add1_top register is written; therefore write, the spec_add1_bottom register first. Filtering is deactivated by writing to the spec_add1_bottom register or by the GEM_RESET.

Type ID Filtering

Frame type IDs are used by software to identify a particular stream of traffic. They can be filtered using the spec_type{1:4} registers. Four type ID registers are available. A type ID register is enabled writing a 1 to the spec_type{1:4} [enable_copy, 31] bit. When a frame is received, the enabled type ID matching results (up to 4) are OR'd together.

The contents of each type ID registers are compared against the length/type ID of the frame being received (for example, bytes 13 and 14 in non-VLAN and non-SNAP encapsulated frames) and written into the RXFIFO if a match is found. The encoded type ID match bits (word 1, bit [22] and bit [23]) in the receive buffer descriptor status are set to indicate which type ID register generated the match, if the receive checksum offload is disabled. The reset state of the type ID registers is zero and is disabled.

Filtering Example

This example illustrates the use of the specific address and type ID match registers for a MAC address of 21:43:65:87:A9:CB. The sequence in the following table shows the beginning of an RX frame. The byte order of transmission starts with the preamble, shown at the top of the table.

For a successful match to specific address 1 register, write the destination address.

***Note*:** In this example, the address mask bits are all disabled (reset value).

- Write `8765_4321h` to gem.spec_add1_bottom
- Write `0000_CBA9h` to gem.spec_add1_top

For a successful match to the type ID 1 register, write the ID and enable the register:

- Write `8000_4321h` to gem.spec_type1

Table 124: **GEM Address and Type ID Filtering Example**

Byte Type	Example Value	Description
Preamble	55	
SFD	D5	Start frame delimiter

Table 124: **GEM Address and Type ID Filtering Example** *(cont'd)*

Byte Type	Example Value	Description
DA (octet 0, LSB)	21	Destination address
DA (octet 1)	43	
DA (octet 2)	65	
DA (octet 3)	87	
DA (octet 4)	A9	
DA (octet 5, MSB)	CB	
SA (octet 0, LSB)	xx	Address of transmitting device
SA (octet 1)	xx	
SA (octet 2)	xx	
SA (octet 3)	xx	
SA (octet 4)	xx	
SA (octet 5, MSB)	xx	
Type ID (MSB)	43	Type ID match 1
Type ID (LSB)	21	

Hash Addressing

The RX hash matching is enabled separately for unicast and multicast frames:

- Enable unicast hash matching. Write a 1 to gem.network_config [unicast_hash_enable].
- Enable multicast hash matching. Write a 1 to gem.network_config [multicast_hash_enable].

The destination address is reduced to a 6-bit index using the following hash function. The hash function is an XOR of every sixth bit of the destination address. If the hash_index points to a bit set in the 64-bit hash address (defined by the hash_top and hash_bottom registers), a match is detected.

Data bit 00 presents the least significant bit of the first byte (this is the multicast/unicast indicator). Data bit 47 represents the most significant bit of the last byte.

Table 125: **Ethernet Hash Indexes**

Hash Index	Data Bits Received, da[nn]							
0	00	06	12	18	24	30	36	42
1	01	07	13	19	25	31	37	43
2	02	08	14	20	26	32	38	44
3	03	09	15	21	27	33	39	45
4	04	10	16	22	28	34	40	46
5	05	11	17	23	29	35	41	47

If the hash index points to a bit that is set in the hash register, the frame is matched according to whether the frame is multicast or unicast.

A multicast match is signaled if:

- [multicast_hash_enable] = 1
- da[00] = 1
- Hash index points to a bit set in the hash register, hash_top and hash_bottom

A unicast match is signaled if:

- [unicast_hash_enable] = 1
- da[00] = 0
- Hash index points to a bit set in the hash register, hash_top and hash_bottom

To receive all multicast frames:

- Write 1's to the hash register
- Write 1 to [multicast_hash_enable]

Capture All Frames

The MAC receiver can capture all valid frames regardless of the address using the copy all frames feature The promiscuous mode is enabled when network_config [copy_all_frames] is set = 1. In this mode, all RX frames are copied into the RXFIFO except for the frames that are:

- Too long (over 1536 bytes)
- Too short (under 64 byes), or
- GMII's RX error (rx_er) signal assert during reception

If the RX frame includes an FCS error, the frame is only captured if the network_config [ignore_rx_fcs, 26] bit is set = 1.

RX Broadcast Frames

When the MAC receiver detects a broadcast frame (address = `0xFFFF_FFFF_FFFF`), the receiver normally writes the frame to the RX packet buffer.

If network_config [no_broadcast] = 1, the broadcast frame is ignored and not written into the packet buffer.

VLAN Support

The Ethernet encoded IEEE Std 802.1Q VLAN tag includes:

- 16-bit tag protocol identifier (TPID): `8100h`.
- 16-bit tag control information (TCI): first three priority bits, then CFI bit, then 12 VID bits.

The VLAN tag is inserted at the 13th byte of the frame adding an extra four bytes to the frame length. To support these extra four bytes, the GEM can accept frame lengths up to 1,536 bytes by setting the network_config [receive_1536_byte_frames, 8] bit = 1.

If the VLAN identifier (VID) is null (`0000h`), a priority-tagged frame is indicated.

The following bits in the RX buffer descriptor status Word [1] provide information about VLAN tagged frames:

- Set bit [21] if the receive frame is VLAN tagged (that is, type ID of `0x8100`).
- Set bit [20] if receive frame is priority tagged (that is, type ID of `0x8100` and null VID). If bits [20] is set, bit [21] is also set.
- Set bits [19], [18], and [17] to priority if the bit [21] is set.
- Set bit [16] to CFI if bit [21] is set.

The controller can be configured to reject all frames except VLAN tagged frames by setting the discard non-VLAN frames bit in the network configuration register.

Wake-on-LAN Support

The MAC receiver supports wake-on-LAN (WOL) by detecting these events on incoming RX frames:

- Magic packets
- Address resolution protocol (ARP) requests to the device IP address
- Specific address 1 filter match
- Multicast hash filter match

The receiver must be enabled by writing a 1 to gem.network_control [enable_receive]. These events can be individually enabled using the gem.wol_register [wol_mask_x] bits.

Also, when WOL is detected, the gem.int_status [wol_interrupt] bit is set by the controller.

IMPORTANT! *A receive buffer in the RXFIFO does not have to be available, but the descriptor must be fetchable from memory when the wake-up event occurs. Alternately, the receive DMA queues can be disabled by setting the GEM.receive_q_ptr [dma_rx_dis_q] bit = 1.*

The wake-up interrupt is asserted for several reasons:

- An RX multicast filter event occurred
- An ARP request is generated
- Specific address 1 match even in the presence of a frame error

Magic Packet Events

For magic-packet events, the frame must be correctly formed and error free. A magic-packet event is detected when all of the following are true.

- Magic-packet events are enabled through bit [16] of the wake-on-LAN register
- RX frame destination address matches the specific address 1 register
- RX frame is correctly formed with no errors
- RX frame contains at least 6 bytes of `0xFF` for synchronization
- There are 16 repetitions of the contents of the specific address 1 register immediately following the synchronization

Address Resolution Protocol

An ARP request event is detected when all of the following are true.

- ARP request events are enabled through bit [17] of the wake-on-LAN register
- Broadcasts are allowed by bit [5] in the network configuration register
- RX frame has a broadcast destination address (bytes 1 to 6)
- RX frame has a type ID field of `0x0806` (bytes 13 and 14)
- RX frame has an ARP operation field of `0x0001` (bytes 21 and 22)
- The least significant 16 bits of the frame's ARP target protocol address (bytes 41 and 42) match the value programmed in bits[15:0] of the wake-on-LAN register

The decoding of the ARP fields adjusts automatically if a VLAN tag is detected within the frame. The reserved value of `0x0000` for the wake-on-LAN target address value does not cause an ARP request event, even if matched by the frame.

Specific Address 1 Filter Match

A specific address 1 filter match event occurs when all of the following are true.

- Specific address 1 events are enabled through bit [18] of the wake-on-LAN register
- RX frame destination address matches the value programmed in the specific address 1 registers

Multicast Hash Filter Match

Multicast filter match event occurs when all of the following are true.

- Multicast hash events are enabled through bit [19] of the wake-on-LAN register
- Multicast hash filtering is enabled through bit [6] of the network configuration register
- RX frame destination address matches against the multicast hash filter
- RX frame destination address is not a broadcast

Precision Timestamp Unit

The timestamp unit (TSU) supports the IEEE Std 1588 for precision time synchronization in local area networks. The TSU works with the exchange of special precision time protocol (PTP) frames. The PTP messages can be transported over IEEE Std 802.3/Ethernet, over Internet Protocol Version 4, or over Internet Protocol Version 6 as described in the annex of IEEE Std P1588.D2.1.

***Note*:** The TSU clock must be active for the GEM controller to operate; this impacts the transmit scheduler. The TSU clock is controlled and activated with the CRL.GEM_TSU_REF_CTRL register.

Synchronization of Master and Slave Clocks

The controller detects when the PTP event messages sync, delay_req, pdelay_req, and pdelay_resp are transmitted and received. Synchronization between master and slave clocks is a two stage process.

The offset between the master and slave clocks is corrected by the master sending a sync frame to the slave with a follow-up frame containing the exact time the sync frame was sent. The GEM controller assist modules on the master and slave side detect exactly when the sync frame was sent by the master and received by the slave. The slave then corrects its clock to match the master clock.

The transmission delay between the master and slave is corrected. The slave sends a delay request frame to the master, which sends a delay response frame in reply. The controller assist modules on the master and slave side detect exactly when the delay request frame was sent by the slave and received by the master. The slave then has enough information to adjust its clock to account for delay.

See the IEEE 1588 v1/v2 or 802.1AS standards for more detailed information on how the slave software calculates the delay based on this information. When GEM is a PTP slave, its timer can be adjusted with this delay. See the Timer Adjustment section below.

Sync and Delay_Req Messages

For TSU assist, it is necessary to timestamp when sync and delay_req messages are sent and received. The timestamp is taken when the message timestamp point passes the clock timestamp point. The message timestamp point is the SFD and the clock timestamp point is the MII. The MAC samples the TSU timer value synchronous to MAC TX/TX clock domains at the MII/GMII boundary.

The MAC inserts the timestamp into the transmitted PTP sync frames (if the one step sync feature is enabled) for capture in the TSU_TIMER_MSB_SEC, TSU_TIMER_NSEC, TSU_TIME_SEC registers, or to pass to the DMA to insert into TX or RX descriptors. For each of these, the SOF event, which is captured in the tx_clk and rx_clk domains, respectively, is synchronized to the tsu_clk domain, and the resulting signal is used to sample the TSU count value. This value is kept stable for an entire frame, or specifically for at least 64 TX/RX clock cycles, because the minimum frame size in Ethernet is 64 bytes and worst case is a transfer rate of 1 byte per cycle. It is used as the source for all the various components within the GEM that require the timestamp value. The IEEE Std 1588 specification refers to sync and delay_req messages as event messages, as these require timestamping. Follow up, delay response, and management messages do not require timestamping and are referred to as general messages.

Peer Delay Request and Response Messages

The IEEE Std 1588 version 2 defines two new PTP event messages that replace the delay request/response messages. These are the peer delay request (pdelay_Req) and peer delay response (pdelay_Resp) messages. These messages are used to calculate the delay on a link. Nodes at both ends of a link send both types of frames (regardless of whether they contain a master or slave clock). The pdelay_resp message contains the time where a pdelay_req was received and is itself an event message. The time at which a pdelay_resp message is received is returned in a pdelay_resp_follow_up message.

PTP Event Message Encapsulation

The controller recognizes four different encapsulations for PTP event messages:

- IEEE Std 1588 version 1 (UDP/IPv4 multicast)
- IEEE Std 1588 version 2 (UDP/IPv4 multicast)
- IEEE Std 1588 version 2 (UDP/IPv6 multicast)
- IEEE Std 1588 version 2 (Ethernet multicast)

Note: Only multicast packets are supported.

Timer Adjustment

The TSU consists of a timer with seconds + nanoseconds + sub nanoseconds registers, increment and adjust registers, and these are accessible through the APB programming interface. The initial value of the timer is written through the tsu_timer_msb_sec, tsu_timer_sec, and tsu_timer_nsec registers. The amount the timer increments by each clock cycle is set by the tsu_timer_incr and tsu_timer_incr_sub_nsec registers. The timer can be adjusted by adding or subtracting an integral number of nanoseconds in a one-off write to the tsu_timer_adjust register. Alternatively, the tsu_timer_incr and tsu_timer_incr_sub_nsec can also be adjusted to tune a slave's timer minutely based on the delay. See the register descriptions for more information.

PTP Event Packet Timestamping

The TSU consists of a timer and registers to capture the time at which PTP event frames cross the message timestamp point. These are accessible memory-mapped through the APB programming interface. An interrupt is issued when a capture register is updated.

The MAC provides timestamp registers that capture the departure time (for transmit) or arrival time (for receive) of PTP event packets (sync and delay request), and peer event packets (peer delay request or peer delay response). Interrupts are optionally generated upon timestamp capture.

MAC Pause Frames

The start of an IEEE Std 802.3 pause frame includes:

- Destination address: `0x0180_C200_0001`
- Source Address: 6 bytes
- Type (MAC control frame): `0x8808`
- Pause opcode: `0x0001`
- Pause time: 2 bytes

The controller supports both a hardware controlled pause of the transmitter upon reception of a pause frame and a hardware generated pause frame transmission.

TIP: *See Clause 31, and Annex 31A and 31B of the IEEE Std 802.3 for a full description of pause operation.*

RX Pause Frames

Bit [13] of the network configuration register is the pause enable control for reception. If this bit is set and a non-zero pause quantum frame is received, transmission pauses.

If a valid pause frame is received, then the pause time register is updated with the new frame's pause time regardless of whether a previous pause frame is active. An interrupt (either bit [12] or bit [13] of the interrupt status register) is triggered when a pause frame is received, but only if the interrupt is enabled (bit [12] and bit [13] of the interrupt mask register). Pause frames received with non-zero quantum are indicated through the interrupt bit [12] of the interrupt status register. Pause frames received with zero quantum are indicated on bit [13] of the interrupt status register.

When the pause time register is loaded and the frame currently being transmitted is sent, no new frames are transmitted until the pause time reaches zero. The loading of a new pause time, and the pausing of transmission, only occurs when the controller is configured for full-duplex operation. If the controller is configured for half-duplex there is no frame is defined as having a destination address that matches either the address stored in specific address register 1 or if it matches the reserved address of `0x0180C2000001`. It must also have the MAC control frame type ID of `0x8808` and have the pause opcode of `0x0001`.

Pause frames that have FCS or other errors are treated as invalid and are discarded. IEEE Std 802.3 pause frames that are received after priority-based flow control (PFC) is negotiated are also discarded. Valid pause frames received increment the pause frames received statistic register. The pause time register decrements every 512-bit times once transmission has stopped. For test purposes, the retry test bit can be set (bit [12] in the network configuration register) which causes the pause time register to decrement every tx_clk cycle when transmission has stopped.

The interrupt (bit [13] in the interrupt status register) is asserted whenever the pause time register decrements to zero (assuming it was enabled by bit [13] in the interrupt mask register). This interrupt is also set when a zero quantum pause frame is received.

PFC Priority-based Pause Frame

TIP: *See the IEEE Std 802.1Qbb for a full description of priority-based pause operation.*

The controller supports PFC priority-based pause transmission and reception. Before PFC pause frames can be received, bit [16] of the network control register must be set. The start of a PFC pause frame includes:

- Destination address: `0x0180C2000001`
- Source address: 6 bytes
- Type (MAC control frame): `0x8808`
- Pause opcode: `0x0101`
- Priority enable vector: 2 bytes
- Pause times: 8 x 2 bytes

Pause Frame Reception

The ability to receive and decode priority-based pause frames is enabled by setting bit [16] of the network control register. When this bit is set, the controller matches either classic IEEE Std 802.3 pause frames or PFC priority-based pause frames. After a priority-based pause frame is received and matched, the controller only matches on priority-based pause frames (this is an IEEE Std 802.1Qbb requirement, known as PFC negotiation). After a priority-based pause is negotiated, any received IEEE Std 802.3x format pause frames are not acted upon. The state of PFC negotiation is identified using the output pfc_negotiate. If a valid priority-based pause frame is received, then the controller decodes the frame and determines which, if any, of the eight priorities are require to be paused. Up to eight pause time registers are then updated with the eight pause times extracted from the frame, regardless of whether a previous pause operation is active or not. An interrupt (either bit [12] or bit [13] of the interrupt status register) is triggered when a pause frame is received, but only if the interrupt is enabled (through bit [12] and bit [13] of the interrupt mask register).

Pause frames received with non-zero quantum are indicated through the interrupt bit [12] of the interrupt status register. Pause frames received with zero quanta are indicated on bit [13] of the interrupt status register. The state of the eight pause time counters are indicated through the outputs rx_pfc_paused. These outputs remain High for the duration of the pause time quanta. The loading of a new pause time only occurs when the controller is configured for full-duplex operation.

If the controller is configured for half-duplex operation, the pause time counters are not loaded, but the pause frame received interrupt is still triggered. A valid pause frame is defined as having a destination address that matches either the address stored in specific address register 1 or if it matches the reserved address of `0x0180C2000001`. It must also have the MAC control frame type ID of `0x8808` and have the pause opcode of `0x0101`.

Pause frames that have FCS or other errors are treated as invalid and are discarded. Valid pause frames received increment the pause frames received statistic register.

The pause time registers decrement every 512-bit times immediately following the PFC frame reception. For test purposes, the retry test bit can be set (bit [12] in the network configuration register).

After transmission, a pause frame transmitted interrupt is generated (bit [14] of the interrupt status register) and the only statistics register that is incremented is the pause frames transmitted register.

PFC pause frames can also be transmitted by the MAC using normal frame transmission methods.

Disable Copy of Pause Frames

Receive pause frames are not captured in the RX buffer if network_config [disable_copy_of_pause_frames, 23] is set = 1.

This setting overrides these conditions:

- [copy_all_frames] bit = 1
- Hash match is true, type ID match is true, destination address match is true

Checksum Hardware

The controller can be programmed to perform IP, TCP, and UDP checksum offloading in both receive and transmit directions, enabled by setting bit [24] in the network configuration register for receive, and bit [11] in the DMA configuration register for transmit.

IPv4 packets contain a 16-bit checksum field, which is the 16-bit 1's complement of the 1's complement sum of all 16-bit words in the header. TCP and UDP packets contain a 16-bit checksum field, which is the 16-bit 1's complement of the 1's complement sum of all 16-bit words in the header, the data, and a conceptual IP pseudo header.

Calculating these checksums in software requires each byte of the packet to be processed.

For TCP and UDP a large amount of processing power can deter the process. Offloading the checksum calculation to the GEM controller can result in significant performance improvements.

For IP, TCP, or UDP checksum offload to be useful, the operating system containing the protocol stack must be aware that this offload is available for the GEM controller to either generate or verify the checksum.

IMPORTANT! *Checksum offload is not possible when partial store and forward is enabled.*

***Note*:** To enable the controller, compute the proper checksum needed by the system software to ensure that the checksum fields are initialized to 0.

RX Checksum Offload

When receive checksum offloading is enabled, the IPv4 header checksum is checked per the IETF Std RFC 791, where the packet meets the following criteria.

- If present, the VLAN header must be four octets long and the CFI bit must not be set
- Encapsulation must be IETF Std RFC 894 Ethernet type encoding or IETF Std RFC 1042 SNAP encoding

- It is a IPv4 packet
- IP header is of a valid length

The controller also checks the TCP checksum per IETF Std RFC 793, or the UDP checksum

per IETF Std RFC 768, if the following criteria are met.

- A IPv4 or IPv6 packet
- Good IP header checksum (if IPv4)
- No IP fragmentation
- A TCP or UDP packet

When an IP, TCP, or UDP frame is received, the receive buffer descriptor provides an indication if the controller was able to verify the checksums. There is also an indication if the frame had LLC SNAP encapsulation. These indication bits replace the type ID match indication bits when receive checksum offload is enabled.

If any of the checksums are verified to be incorrect by the controller, the packet is discarded and the appropriate statistics counter is incremented.

TX Checksum Offload

The transmitter checksum offload is only available when the full store and forward mode is enabled. This is because the complete frame to be transmitted must be read into the packet buffer memory before the checksum can be calculated and written back into the headers at the beginning of the frame.

Transmitter checksum offload is enabled by setting bit [11] in the DMA configuration register. When enabled, it monitors the frame as it is written into the transmitter packet buffer memory to automatically detect the protocol of the frame. Protocol support is identical to the receiver checksum offload.

For transmit checksum generation and substitution to occur, the protocol of the frame must be recognized and the frame must be provided without the FCS field, by ensuring that bit [16] of the transmit descriptor word [1] is clear. If the frame data already had the FCS field, it would be corrupted by the substitution of the new checksum fields.

If these conditions are met, the transmit checksum offload engine calculates the IP, TCP, and UDP checksums as appropriate. When the full packet is completely written into packet buffer memory, the checksums are valid and the relevant status buffer locations are updated for the new checksum fields as per standard IP/TCP and UDP packet structures.

If the transmitter checksum engine is prevented from generating the relevant checksums, bits [22:20] of the transmitter DMA writeback status are updated to identify the reason for the error. The frame is still transmitted, but without the checksum substitution. Typically the reason that the substitution does not occur is that the protocol is not recognized.

Register Reference

The GEM registers are divided into these groups:

- Control and Status (from GEM register set)
- Statistics (from GEM register set)
- System-Level Registers (from CRL and LPD_IOP_SLCR register sets)
- AXI Transaction Control

Control and Status

The GEM control registers are summarized in the following table.

Table 126: **GEM Control and Status Registers**

Register Name	Offset Address	Access Type	Description
Controller and MAC Configuration			
network_control, network_config, network_status	0x000, 0x004 0x008	RW R	Network control, configuration, and status
pause_time	0x038	R	RX quantum pause time
tx_pause_quantum	0x03C	RW	TX quantum pause time
PHY Management			
phy_management	0x034	RW	MDIO interface control for external PHY
DMA and Buffer Descriptor Control			
dma_config transmit_status, receive_status receive_q_ptr, receive_q1_ptr transmit_q_ptr, transmit_q1_ptr dma_addr_or_mask	0x010 0x014, 0x020 0x018, 0x480 0x01C, 0x440 0x0D0	RW WTC R R RW	Receive and transmit status, queue base address, and controls
Interrupts			
int_status int_enable, int_disable int_mask	0x024 0x028, 0x02C 0x030	WTC W R	Interrupt status, enable/disable, and mask
Miscellaneous			

Table 126: **GEM Control and Status Registers** *(cont'd)*

Register Name	Offset Address	Access Type	Description
pbuf_txcutthru, pbuf_rxcutthru	0x040, 0x044	RW	Partial store
jumbo_max_length	0x048	RW	Maximum jumbo frame size
external_fifo_interface	0x04C	RW	External FIFO interface enable
axi_max_pipeline	0x054	RW	Maximum outstanding AXI transactions
rsc_control	0x058	RW	Receive side coalescing
int_moderation sys_wake_time fatal_or_non_fatal_int_sel	0x05C 0x060 0x064	RW	Interrupt generation modulator, system wake time, fatal interrupt select
lockup_config, lockup_config3 rx_mac_lockup_time	0x068, 0x070 0x06C	RW	Lock-up configuration
rx_water_mark	0x07C	RW	RX high and low watermark control
hash_bottom, hash_top	0x080, 0x084	RW	Hash address
Address Filtering and ID Match			
spec_add{1:4}_bottom spec_add{1:4}_top, mask_add1_bottom mask_add1_top spec_type{1:4}	0x088 + 0x08C + 0x0C8 0x0CC 0x0A8 +	RW	Address filtering Match type
wol_register	0x0B8	RW	Wake on LAN
stretch_ratio	0x0BC	RW	Inter-packet gap stretch value
stacked_vlan	0x0C0	RW	Stack VLAN match and enable
tx_pfc_pause	0x0C4	RW	Priority flow control
rx_ptp_unicast tx_ptp_unicast tsu_nsec_cmp tsu_sec_cmp tsu_msb_sec_cmp	0x0D4 0x0D8 0x0DC 0x0E0 0x0E4	RW	Timestamp control
tsu_ptp_tx_msb_sec tsu_ptp_rx_msb_sec tsu_peer_tx_msb_sec tsu_peer_rx_msb_sec	0x0E8 0x0EC 0x0F0 0x0F4	R	Timestamp status
Timestamp Unit, Precision Time Protocol			
tsu_timer_nsec tsu_timer_adjust tsu_timer_incr	0x1D4 0x1D8 0x1DC	RW	IEEE Std 1588: second, nanosecond counter and adjustment, increment

Table 126: **GEM Control and Status Registers** *(cont'd)*

Register Name	Offset Address	Access Type	Description
tsu_timer_incr_sub_nsec tsu_timer_msb_sec tsu_timer_sec tsu_strobe_msb_sec tsu_strobe_sec tsu_strobe_nsec[1]	0x1BC 0x1C0 0x1D0 0x1C4 0x1C8 0x1CC	RW RW R R R	Timestamp timer control and strobe value
tsu_ptp_tx_sec tsu_ptp_tx_nsec tsu_ptp_rx_sec tsu_ptp_rx_nsec tsu_peer_tx_sec tsu_peer_tx_nsec tsu_peer_rx_sec tsu_peer_rx_nsec	0x1E0 0x1E4 0x1E8 0x1EC 0x1F0 0x1F4 0x1F8 0x1FC	R	IEEE Std 1588: TX and RX normal/peer second, nanosecond counter
Low-Power Idle Control			
rx_lpi, rx_lpi_time tx_lpi, tx_lpi_time	0x270, 0x274 0x278, 0x27C	R R	Transaction count and time
Design Configuration			
designcfg_debug{1:12}	0x280 to 0x2AC	R	Design configuration
Miscellaneous			
cbs_control cbs_idleslope_q_a, cbs_idleslope_q_b	0x4BC 0x4C0, 0x4C4	RW	Credit-based shaping control
upper_tx_q_base_addr upper_rx_q_base_addr	0x4C8 0x4D4	RW	Descriptor queue base address
tx_bd_control, rx_bd_control	0x4CC, 0x4D0	RW	Timestamp insertion mode
scr1_reg0, scr1_reg1 scr1_reg2, scr1_reg3 scr2_reg0, scr2_reg1 scr2_reg2, scr2_reg3	0x500, 0x504 0x508, 0x50C 0x540, 0x544 0x548, 0x54C	RW	Screen 1 and 2 control
tx_sched_ctrl bw_rate_limit_q0to1 tx_q_seg_alloc_q0to1	0x580 0x590 0x5A0	RW	TX queue scheduling mode, bandwidth weighing, and space allocation
int_q1_status int_q1_enable, int_q1_disable int_q1_mask	0x400 0x600, 0x620 0x640	WTC W R	Queue 1 status and interrupt enable, disable, mask
scr2_ether_reg0, scr2_ether_reg1 scr2_ether_reg2, scr2_ether_reg3	0x6E0, 0x6E4 0x6E8, 0x6EC	RW RW	Screen type 2 Ethernet type compare registers

Table 126: **GEM Control and Status Registers** *(cont'd)*

Register Name	Offset Address	Access Type	Description
scr2_compare0_wd0, scr2_compare0_wd1 scr2_compare1_wd0, scr2_compare1_wd1 scr2_compare2_wd0, scr2_compare2_wd1 scr2_compare3_wd0, scr2_compare3_wd1	0x700, 0x704 0x708, 0x70C 0x710, 0x714 0x718, 0x71C	RW	Four screen type 2 compare functions (words 0 and 1)
enst_start_time_q0, enst_start_time_q1 enst_on_time_q0, enst_on_time_q1 enst_off_time_q0, enst_off_time_q1 enst_control	0x800, 0x804 0x820, 0x824 0x840, 0x844 0x880	RW	Queue start, open, and close times, and enable, disable
Extended Stream Identification Functions			
frer_timeout, frer_red_tag frer_control_1_a, frer_control_1_b frer_statistics_1_a, frer_statistics_1_b etc for control/status 1 to 16	0x8A0, 0x8A4 0x8C0, 0x8C4 0x8C8, 0x8CC 0x8D0 to 0x9BC	RW	Timeout, control, and statistics
rx_q0_flush, rx_q1_flush	0xB00, 0xB04	RW	Queue flush
scr2_reg0_rate_limit, scr2_reg1_rate_limit	0xB44	RW	Maximum rate limit for screen 2

Notes:

1. The timer sync strobe registers are loaded with the value of the timer when the input signal emio_enet{0:3}_tsu_inc_ctrl[1:0] = 00b. However, the timer sync strobe registers are updated only when emio_enet{0:3}_tsu_inc_ctrl signal toggles between 11b and 00b.

Statistics

The statistics registers hold counts for various types of events associated with transmit and receive operations. These registers, along with the status words stored in the receive buffer list, enable software to generate network management statistics compatible with IEEE Std 802.3. These registers are listed in the following table.

Table 127: **GEM Statistics Registers**

Register Name	Offset Address	Access Type	Description
OCTETS_TXED_BOTTOM, OCTETS_TXED_TOP FRAMES_TXED_OK BROADCAST_TXED, MULTICAST_TXED PAUSE_FRAMES_TXED FRAMES_TXED_64, FRAMES_TXED_65 FRAMES_TXED_128, FRAMES_TXED_256 FRAMES_TXED_512, FRAMES_TXED_1024 FRAMES_TXED_1519	0x100, 0x104 0x108 0x10C, 0x110 0x114 0x118, 0x11C 0x120, 0x124 0x128, 0x12C 0x130	R	TX frames statistics
TX_UNDERRUNS SINGLE_COLLISIONS EXCESSIVE_COLLISION SLATE_COLLISIONS	0x134 0x138 0x140 0x144	R	TX statistics

Table 127: **GEM Statistics Registers** *(cont'd)*

Register Name	Offset Address	Access Type	Description
OCTETS_RXED_BOTTOM, OCTETS_RXED_TOP FRAMES_RXED_OK BROADCAST_RXED, MULTICAST_RXED PAUSE_FRAMES_RXED FRAMES_RXED_64, FRAMES_RXED_65 FRAMES_RXED_128, FRAMES_RXED_256 FRAMES_RXED_512, FRAMES_RXED_1024 FRAMES_RXED_1519	0x150, 0x154 0x158 0x15C, 0x160 0x164 0x168, 0x16C 0x170, 0x174 0x178, 0x17C 0x180	R	RX frame statistics
UNDERSIZE_FRAME, EXCESSIVE_RX_LENGTH RX_JABBERS FCS_ERRORS RX_LENGTH_ERRORS, RX_SYMBOL_ERRORS ALIGNMENT_ERRORS, RX_RESOURCE_ERRORS	0x184, 0x188 0x18C 0x190 0x194, 0x198 0x19C, 0x1A0	R	RX statistics
RX_OVERRUNS, RX_IP_CK_ERRORS RX_TCP_CK_ERRORS, RX_UDP_CK_ERRORS AUTO_FLUSHED_PKTS	0x1A4, 0x1A8 0x1AC, 0x1B0 0x1B4	R	RX statistics

System-Level Registers

The controller includes registers from multiple system-level register sets:

- CRL to generate clocks and resets
- LPD_IOP_SLCR to control the routing for clocks
- PMC_IOP_SLCR to control the routing for clocks

***Note*:** The GEM_TSU_CLK clock must be active whenever the GEM is used. If the timestamp unit is not used, it still must be actively clocked.

Table 128: **GEM System-Level Registers**

Register Set	Register Name	Bit Field	Description
Controller Clocks			
CRL	GEM0_REF_CTRL GEM1_REF_CTRL	[SRC_SEL] [DIVISOR] [CLKACT] [CLKACT_TX] [CLKACT_RX]	Clock dividers to generate controller reference clock; independent dividers for each controller.
LPD_IOP_SLCR	GEM_CLK_CTRL	[GEMn_REF_SRC_SEL]	GEMx_REF_CLK source clock select: clock divider or EMIO.
LPD_IOP_SLCR	GEM_CLK_CTRL	[GEMn_RX_SRC_SEL]	RX I/O clock select: MIO or EMIO.
TSU Clock			
LPD_IOP_SLCR	GEM_CLK_CTRL	[TSU_CLK_SEL]	TSU clock select: reference clock or MIO pin (PMC or LPD).

Table 128: **GEM System-Level Registers** *(cont'd)*

Register Set	Register Name	Bit Field	Description
CRL	GEM_TSU_REF_CTRL	[SRC_SEL] [DIVISOR] [CLKACT]	TSU reference clock. This clock is common to both controllers.
LPD_IOP_SLCR	GEM_CLK_CTRL	[TSU_CLK_LB_SEL]	TSU clock loopback select: PS or PL.
TSU Clock Routing			
PMC_IOP_SLCR	MIO_PIN_50 MIO_PIN_51	[L0_SEL]	Select routing through the MIO or, as default EMIO.
LPD_IOP_SLCR	MIO_PIN_24 MIO_PIN_25		
			Select between the PMC or LPD MIO.
LPD_IOP_SLCR	GEM_CLK_CTRL	[GEMn_FIFO_CLK_SEL]	TX clock select for GMII/MII: normal clock or TX clock loopback from PL.
Controller Resets			
CRL	RST_GEM0 RST_GEM1	[RESET]	Controller reset: High to assert and hold the reset signal.

AXI Transaction Control

The AXI transactions generated by the DMA master can be configured for bufferability and coherency.

Table 129: **GEM AXI Transaction Control Registers**

Register Name	Offset Address	Access Type	Description
Coherency			
GEM0_IOP_COHERENT_CTRL GEM1_IOP_COHERENT_CTRL	0x0324 0x0344	RW	Select cache coherency policy and bufferability.
Routing			
GEM0_IOP_INTERCONNECT_ROUTING GEM1_IOP_INTERCONNECT_ROUTING	0x0328 0x0348	RW	Enable transaction to be routed to CCI.
QoS			
GEM0_IOP_INTERCONNECT_QOS GEM1_IOP_INTERCONNECT_QOS	0x032C 0x034C	RW	Select QoS bit values

I/O Signal Reference

The GEM controller has several I/O interfaces and routing options:

- MIO - RGMII on PMC or LPD MIO pins
- EMIO - GMII/MII port interface signals
- MDIO PHY Interface on LPD MIO or PL EMIO
- Timestamp Unit Interface on LPD MIO or PL EMIO or LPD clock controller

MIO - RGMII

The RGMII signals are listed in the following table.

Table 130: **GEM Controller RGMII Interface Signals**

MIO				
Signal Name	**I/O**	**PMC MIO**	**LPD MIO**	**MIO-at-a-Glance Table**
GEM0_TX_CLK	Output	26	0	0
GEM0_TXD[0:3]	Output	27:30	1:4	1:4
GEM0_TX_CTRL	Output	31	5	5
GEM0_RX_CLK	Input	32	6	6
GEM0_RXD[0:3]	Input	33:36	7:10	7:10
GEM0_RX_CTRL	Input	37	11	11
GEM1_TX_CLK	Output	38	12	0
GEM1_TXD[0:3]	Output	39:42	13:16	1:4
GEM1_TX_CTRL	Output	43	17	5
GEM1_RX_CLK	Input	44	18	6
GEM1_RXD[0:3]	Input	45:48	19:22	7:10
GEM1_RX_CTRL	Input	49	23	11

EMIO - GMII/MII

The I/O interface to the PL provides design flexibility for the interface protocol. The GMII/MII signals on the EMIO interface to the PL can be used to implement interface standards on the PL pins. This is shown in Block Diagram.

MDIO PHY Interface

The MDIO signals are available on the MIO pins or the EMIO port interface signals.

Table 131: **GEM Controller MDIO Signals**

MIO							EMIO	
Signal Name	I/O	PMC MIO Pin		LPD MIO Pin		MIO-at-a-Glance Table	Signal Name	I/O
		A	B	C	D			
GEM0_MDIO_CLK GEM1_MDIO_CLK	Output	50	~	24	~	0		Output
GEM0_MDIO_DATA GEM1_MDIO_DATA	I/O	51	~	25	~	1		I/O

Timestamp Unit Interface

The timestamp unit (TSU) clock signal can be sourced from the LPD clock controller, an MIO pin, or a EMIO port interface signal. The TSU clock is programmed using the registers listed in System-Level Registers. There is one clock shared by both GEM controllers.

Table 132: **GEM Timestamp Unit Interface**

MIO							EMIO	
Signal Name	I/O	PMC MIO Pin		LPD MIO Pin		MIO-at-a-Glance Table	Signal Name	I/O
		A	B	C	D			
GEM_TSU_CLK	Input	50	51	24	25	0		
COUNTER	Output	~						
CTRL0	Input							
CTRL1	Input							

GPIO Controller

Two Controllers, Multiple Banks

The general purpose I/O (GPIO) is a collection of input/output signals available to software applications. Each GPIO channel is independently and dynamically programmed as input, output, or interrupt sensing mode.

Software applications can read all GPIO values within a bank using a single load instruction. Software can write data to one or more GPIOs using a single store instruction on a half-bank boundary. The channel architecture is shown in the Channel Block Diagram.

The two GPIO controllers have the same functionality. There are a total of 174 channels in two controllers:

- PMC GPIO controller:
  - Two banks (26 channels each) to PMC MIO
  - Two banks (32 channels each) to PL EMIO
- LPD GPIO controller:
  - One bank (26 channels) to LPD MIO
  - One bank (32 channels) to PL EMIO

The GPIO channels are programmed via an APB slave programming interface; the Registers section summarizes them.

The Input Programming Model shows how to configure a GPIO as an input. The Interrupt Programming Model describes an input can generate a system interrupt.

The GPIO can be an output as described in the Output Programming Model chapter.

The I/O signal MIO muxing and EMIO port signals are listed in the GPIO I/O Signals section.

Features

Each GPIO channel can be dynamically programmed on an individual or bank basis.

- Enable, bit or bank data write, output enable and direction controls
  - Enable 3-state output
  - Write output logic level
  - Direction control
- Programmable interrupts on individual GPIO basis
  - Raw status read and masked interrupt
  - Selectable sensitivity: Level-sensitive (High or Low) or edge-sensitive (rising, falling, or both)
- Two methods to write output logic levels:
  - Full-bank write using the DATA_x registers
  - Maskable-bit write on half-bank basis using the MASK_DATA_x_{LWS, MWS} registers
- Simultaneous output switching is possible, see note in GPIO Register Descriptions
- Input logic levels are read one bank at a time using the DATA_RO_x registers

Comparison to Previous Generation Xilinx Devices

The functionality of the GPIO controller is similar to previous devices except there are two GPIO controllers:

- PMC GPIO controller with four banks (two to MIO, two to EMIO)
- LPD GPIO controller with two banks (one to MIO, one to EMIO)

Each MIO bank includes 26 channels. Each EMIO bank includes 32 channels. There are also differences in the MIO buffer control registers, for more information see the GPIO programming model in MIO Pin Configuration.

System Perspective

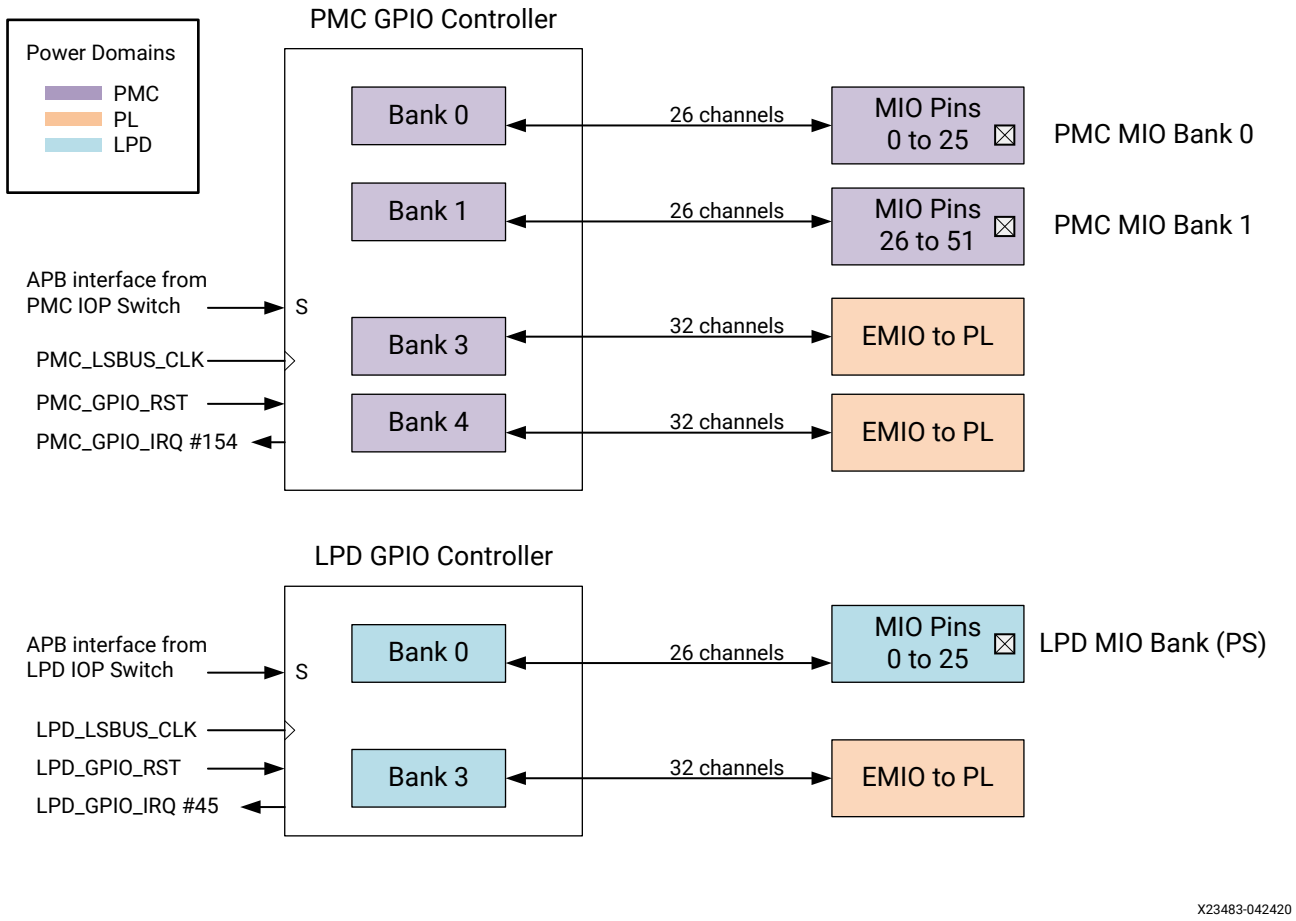
The GPIO controllers are addressed as 32-bit slaves. One controller is connected to the PMC APB switch. The other controller is attached to the LPD APB switch. The controller is clocked by its APB interface clock. Each controller can generate an IRQ system interrupt signal.

The high-level block diagram is shown below. The I/O interface is described in EMIO Signals.

Block Diagram

The MIO and EMIO banks for the two controllers are shown in the following figure. Not all banks within a controller are implemented.

Figure 78: **High-level Block Diagram**



System Interface

The controller is programmed with a 32-bit APB slave programming interface.

System Signals

Resets

The controllers are reset by the following register bits:

- PMC_GPIO_RST: CRP.RST_GPIO [RESET]

- LPD_GPIO_RST: CRL.RST_GPIO [RESET]

Clocks

The controllers are clocked by their APB interface clock using the following registers:

- PMC_LSBUS_CLK: CRP.PMC_LSBUS_REF_CTRL
- LPD_LSBUS_CLK: CRL.LPD_LSBUS_CTRL

Interrupts

Each controller accumulates an OR of all interrupt configured channels. If a input interrupt channel detects an event, the signal is routed to the mask register and OR'd with others to potentially assert a system interrupt signal. The system interrupt is routed to multiple destinations. The signals are listed in the IRQ System Interrupts table. The system IRQ number for each controller:

- LPD_GPIO IRQ#45
- PMC_GPIO IRQ#154

Errors

The controller detects APB address decode errors. When there is an access violation, the controller can optionally generate a SLVERR response to the master and/or generate a system error. There is no other event that can generate a system error.

I/O Interface

Each controller has multiple banks of GPIO channels routed to their local MIO multiplexers or to their PL EMIO. There are three signals from each GPIO channel: input, output, and output enable.

The functionality of a channel is programmed by the GPIO registers.

The PMC and LPD SLCR registers configure the MIO PSIO buffer, PMC and PS PSIO Buffers. This includes input and output characteristics, and tristate control. The SLCR registers can override the signaling from the GPIO controller.

The port interface signals to the PL EMIO consist of an input, output, and output enable for each channel.

MIO Banks

Each MIO bank:

- 26 channels, 26 pins

- Configurable IO buffer characteristics, routing (SLCR registers)
- Separate IO power rail
- Aligned with a GPIO bank

EMIO Banks

Each EMIO bank:

- 32 channels, 96 signals (input, output, enable)
- Aligned with a GPIO bank

The signals are listed in GPIO I/O Signals.

Programming Model

The GPIO controller includes a memory-mapped APB programming interface for software:

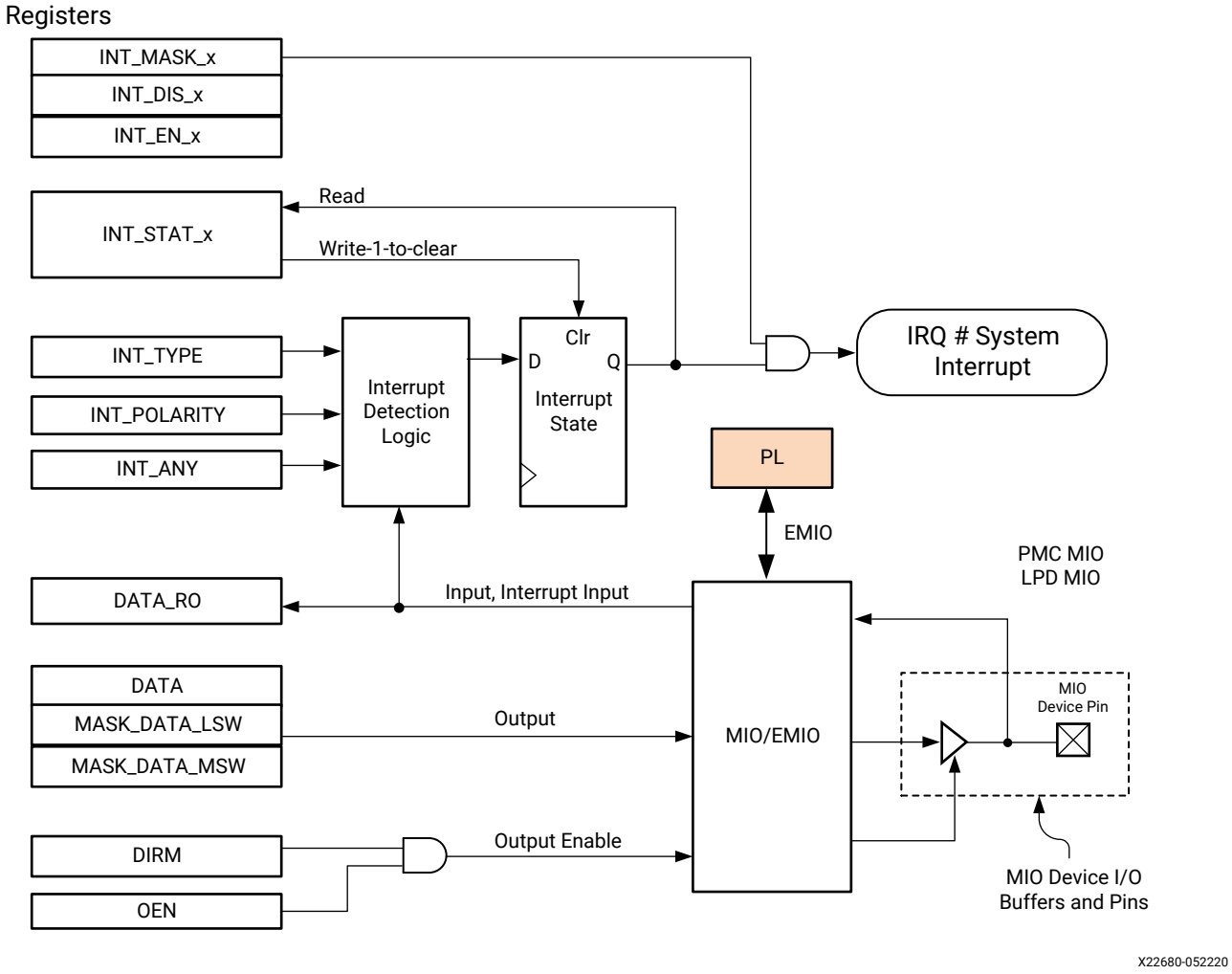
- Mode and status
- Output control
- Input state
- Interrupt configuration

The registers are included in the GPIO Register Descriptions.

Channel Block Diagram

The functionality of a GPIO channel is illustrated in the following figure.

Figure 79: **GPIO Channel Block Diagram**



Input Programming Model

In input mode, the pin voltage level is passed through a meta-stability protection circuit and translated to a logic level. The pin logic level is readable using the DATA_RO registers or the INT_STAT register.

The INTR_STAT regist In the latter case, the direction control must be set to 0 for the input from the I/O pad to be passed through to the register. There are two APB address locations allocated to the pin: A read only location for the dedicated path and a read/write location for the registered path. The pin value can be read from either location in the input mode. The two paths produce different values in the output mode with inactive output enable.

Interrupt Programming Model

The interrupt detection logic monitors the GPIO input signal. The interrupt trigger can be a positive edge, negative edge, either edge, Low-level or High-level. The trigger sensitivity is programmed using the INT_TYPE, INT_POLARITY and INT_ANY registers.

If an interrupt is detected, the GPIO's INT_STAT state is set true by the interrupt detection logic. If the INT_STAT state is enabled (unmasked), then the interrupt propagates through to a large OR function. This function combines all interrupts for all GPIOs in all four banks to one output (IRQ ID#52) to the interrupt controller. If the interrupt is disabled (masked), then the INT_STAT state is maintained until cleared, but it does not propagate to the interrupt controller unless the INT_EN is later written to disable the mask. As all GPIOs share the same interrupt, software must consider both INT_MASK and INT_STAT to determine which GPIO is causing an interrupt.

The interrupt mask state is controlled by writing a 1 to the INT_EN and INT_DIS registers. Writing a 1 to the INT_EN register disables the mask allowing an active interrupt to propagate to the interrupt controller. Writing a 1 to the INT_DIS register enables the mask. The state of the interrupt mask can be read using the INT_MASK register.

If the GPIO interrupt is edge sensitive, the INT state is latched by the detection logic. The INT latch is cleared by writing a 1 to the INT_STAT register. For level-sensitive interrupts, the source of the interrupt input to the GPIO must be cleared to clear the interrupt signal. Alternatively, software can mask that input using the INT_DIS register.

The state of the interrupt signal going to the interrupt controller can be inferred by reading the INT_STAT and INT_MASK registers. This interrupt signal is asserted if INT_STAT=1 and INT_MASK=0.

GPIO bank control is summarized as follows:

- INT_MASK: This register is read-only and shows which bits are currently masked and which are un-masked/enabled.
- INT_EN: Writing a 1 to any bit of this register enables/unmasks that signal for interrupts. Reading from this register returns an unpredictable value.
- INT_DIS: Writing a 1 to any bit of this register masks that signal for interrupts. Reading from this register returns an unpredictable value.
- INT_STAT: This registers shows if an interrupt event has occurred or not. Writing a 1 to a bit in this register clears the interrupt status for that bit. Writing a 0 to a bit in this register is ignored.
- INT_TYPE: This register controls whether the interrupt is edge sensitive or level sensitive.
- INT_POLARITY: This register controls whether the interrupt is active-Low or active High (or falling-edge sensitive or rising-edge sensitive).

- INT_ON_ANY: If INT_TYPE is set to edge sensitive, then this register enables an interrupt event on both rising and falling edges. This register is ignored if INT_TYPE is set to level sensitive.

Table 133: **GPIO Interrupt Trigger Settings**

Type	gpio.INT_TYPE_x	gpio.INT_POLARITY_x	gpio.INT_ANY_x
Rising edge-sensitive	1	1	0
Falling edge-sensitive	1	0	0
Both rising and falling edge-sensitive	1	1	1
Level sensitive, asserted High	0	x	x
Level sensitive, asserted Low	0	1	x

***Note*:** Register writes must be 32 bits.

Output Programming Model

In output mode, the output is driven by a register bit setting. The direction control and the output enable must both be set = 1 for the register setting to appear as an output on the MIO pin.

The direction control can be used to disable the input logic level from being recorded in the data read only (DATA_n_RO) register.

The output enable can be used to control whether an output value is driven on the pin. The actual I/O pad direction is the logical combination of both these signals; the output enable value is ignored when the direction mode is set to input.

Registers

The two GPIO controllers have different registers sets: PMC_GPIO and LPD_GPIO. There are a total of 174 channels.

- PMC GPIO controller
  - Registers 0, 1 control 52 MIO pins for 52 channels
  - Registers 3, 4 connect to 192 EMIO signals to 64 channels

- LPD GPIO controller
  - Register 0 controls 26 MIO pins for 26 channels
  - Register 3 connects to 96 EMIO signals for 32 channels

The register base address for each controller:

- PMC GPIO `0xF102_0000`
- LPD GPIO `0xFF0B_0000`

Table 134: **GPIO Register Overview**

PMC GPIO Controller		LPD GPIO Controller		Access Type
PMC_GPIO Register Set	**Address Offset**	**LPD_GPIO Register Set**	**Address Offset**	
Data Read and Write				
MASK_DATA_{0, 1, 3, 4}_LSW MASK_DATA_{0, 1, 3, 4}_MSW	0x000 ... 0x004 ...	MASK_DATA_{0, 3}_LSW MASK_DATA_{0, 3}_MSW	0x000, 0x018 0x004, 0x01C	R/W, W
DATA_{0, 1, 3, 4}	0x040 ...	DATA_{0, 3}	0x040, 0x04C	R/W
DATA_{0, 1, 3, 4}_RO	0x060 ...	DATA_{0, 3}_RO	0x060, 0x06C	R
I/O Buffer Control				
BYPM_{0, 1, 3, 4}	0x200 ...	BYPM_{0, 3}	0x200, 0x2C0	R/W
DIRM_{0, 1, 3, 4}	0x204 ...	DIRM_{0, 3}	0x204, 0x2C4	R/W
OEN_{0, 1, 3, 4}	0x208 ...	OEN_{0, 3}	0x208, 0x2C8	R/W
Interrupt Control				
INT_MASK_{0, 1, 3, 4}	0x20C ...	INT_MASK_{0, 3}	0x20C, 0x2CC	R
INT_EN_{0, 1, 3, 4}	0x210 ...	INT_EN_{0, 3}	0x210, 0x2D0	W
INT_DIS_{0, 1, 3, 4}	0x214 ...	INT_DIS_{0, 3}	0x214, 0x2D4	W
INT_STAT_{0, 1, 3, 4}	0x218 ...	INT_STAT_{0, 3}	0x218, 0x2D8	W1C
INT_TYPE_{0, 1, 3, 4}	0x21C ...	INT_TYPE_{0, 3}	0x21C, 0x2DC	R/W
INT_POLARITY_{0, 1, 3, 4}	0x220 ...	INT_POLARITY_{0, 3}	0x220, 0x2E0	R/W
INT_ANY_{0, 1, 3, 4}	0x224 ...	INT_ANY_{0, 3}	0x224, 0x2E4	R/W

GPIO Register Descriptions

Data Read and Write

- **DATA_RO**, write-only:

  For MIOs, this register always returns the state of the GPIO MIO pin. If the GPIO is configured as an output, this normally reflects the value being driven on the output regardless of the DIRM_x setting.

Note: If the MIO is not configured for this pin as a GPIO, the DATA_RO returns unpredictable results.

- **DATA**, read-write:

  This register controls the value to be output when the GPIO signal is configured as an output. All 32 bits of this register are written at one time. Reading from this register returns the previous value written to either DATA or MASK_DATA_{LSW,MSW}, and does not return the current value on the device pin.

- **MASK_DATA_LSW**:

  This register enables more selective changes to the desired output value. Any combination of up to 16 bits can be written. Those bits that are not written are unchanged and hold their previous value. Reading from this register returns the previous value written to either DATA or MASK_DATA_{LSW,MSW}; it does not return the current value on the device pin. This register avoids the need for a read-modify-write sequence for unchanged bits.

- **MASK_DATA_MSW**:

  This register is the same as MASK_DATA_LSW, except it controls the upper 16 channels of the bank.

I/O Buffer Control

Software configures each GPIO as either an input, output, or interrupt input.

- **DIRM**

  Direction mode controls whether the I/O pin is acting as an input or an output. Because the input logic is always enabled, this effectively controls the output driver. When DIRM = 0, the output driver is disabled.

- **OEN**

  When the I/O is configured as an output, the OEN controls whether the output is enabled (OEN = 1) or in tristate (OEN = 1).

  Note: There are overriding tristate control registers in the PMC_IOP_SLCR and LPD_IOP_SLCR register sets. If a bit in the MIO_MST_TRIn register is set = 1, the output buffer is put in a tristate mode regardless of the state of the OEN signal state from the GPIO controller.

Interrupt Control

There are several interrupt control registers.

- INT_MASK masks the latched INT_STAT value. To generate an interrupt:
  - INT_MASK must = 0 (enable interrupt)
  - INT_STAT must = 1 (active interrupt)
- INT_EN is write-only. Write 1 to enable the interrupt; sets the INT_MASK bit = 0.

- INT_DIS is write-only. Write 1 to disable the interrupt; set the INT_MASK bit = 1.
- INT_STAT indicates if an interrupt event occurred, latched before INT_MASK.
- INT_TYPE is programmed by software to set level (0) or edge (1) sensitivity.
- INT_POLARITY selects between active-Low/falling (0) and active-High/rising (1) sensitivity.
- INT_ANY selects single edge sensitivity defined by INT_POLARITY (0) or either edge event (1).

GPIO I/O Signals

There are two types of GPIO banks:

- MIO pins
- PL EMIO port signals

Each I/O pin can be individually programmed.

The MIO pins are selected with PMC_IOP_SLCR.MIO_PIN_0 register, for example.

Each GPIO channel consists of data in, data out, and 3-state output control. For MIO, these signals control the I/O buffer on the pin pad. For PL EMIO, all three signals connect between the GPIO controller and the PL. The I/O buffer AC and DC characteristics are programmed on a per bank basis.

The LPD GPIO controller attaches to the LPD MIO.

Table 135: **GPIO MIO Signals**

MIO Pin Signals					
GPIO Bank	**Signal Name**	**I/O**	**PMC MUX Pin**	**PS MUX Pin**	**MIO-at-a-Glance Tables**
PMC Bank 0	PMC_GPIO[0:25]	I/O	0:25	~	0:25
PMC Bank 1	PMC_GPIO[26:51]	I/O	26:51	~	26:51
LPD Bank 0	LPD_GPIO[0:25]	I/O	~	0:25	0:25

Table 136: **GPIO PL EMIO Signals**

EMIO Port Signals		
GPIO Bank	**Signal Name**	**I/O**
PMC Bank 3 [0:31]	Input_x	I
	Output_x	O
	Output_En_x	O

Table 136: **GPIO PL EMIO Signals** *(cont'd)*

EMIO Port Signals		
GPIO Bank	**Signal Name**	**I/O**
PMC Bank 4 [0:31]	Input_x	I
	Output_x	O
	Output_En_x	O
LPD Bank 3 [0:31]	Input_x	I
	Output_x	O
	Output_En_x	O

Assigned MIO Signals

Software Allocated Signals

The PMC_GPIO [12] channel is assigned to the reset for the OSPI flash memory device.

EMIO Signals

For each GPIO channel, there are three signals routed to the PL: input, output, and output enable.

The register interface for the EMIO banks is the same as for the MIO banks. However, the EMIO interface is simply wires between the PS and the PL, so there are a few differences:

- The inputs are wires from the PL and are unrelated to the output values or the OEN register. They can be read from the DATA_RO register when DIRM is set to 0, making it an input.
- The output wires are not 3-state capable, so they are unaffected by OEN. The value to be output is programmed using the DATA, MASK_DATA_LSW, and MASK_DATA_MSW registers. DIRM must be set to 1, making it an output.
- The output enable wires are simply outputs from the PS. These are controlled by the DIRM/OEN registers as follows: EMIOGPIOTN[x] = DIRM[x] & OEN[x].

The EMIO I/Os are not connected to the MIO I/Os in any way. The EMIO inputs cannot be connected to the MIO outputs and the MIO inputs cannot be connected to the EMIO outputs. Each bank is independent and can only be used as software observable/controllable signals.

I2C Controller

The I2C controllers can function as a master or a slave in a multi-master design. They can operate over a clock frequency range up to 400 kb/s. The controller supports multi-master mode for 7-bit and extended 10-bit addressing.

In master mode, a transfer can only be initiated by software writing the slave address into the address register. The software is notified of any available received data by a data interrupt or a transfer complete interrupt. If the hold bit is set, the I/O interface holds the clock signal (SCL) Low until after the data is transmitted to support slow software response. The master can be programmed to use both normal addressing and extended addressing. The extended addressing is only supported in master mode.

In slave monitor mode, the controller is set up as a master and continues to attempt a transfer to a particular slave until the slave device responds with an ACK or until the timeout occurs.

The controller supports repeated start functionality. After the start condition, the master can generate a repeated start. This is equivalent to a normal start and is usually followed by the slave I2C address.

A common feature between master mode and slave mode is the timeout interrupt flag bit. If at any point the SCL clock signal is held Low by the master or the accessed slave for more than the period specified in the timeout register, the timeout interrupt bit is set. This can generate an interrupt to the software to avoid stall conditions.

In slave mode, the controller responds to the external master device. A slave cannot initiate a transfer over the I2C bus, only a master can initiate transfers. Both master and slave can transfer data over the I2C bus, but that transfer is always controlled by the master.

There are multiple instances of the I2C controller. They are all similar in functionality.

- LPD_I2C0 in PS
- LPD_I2C1 in PS
- PMC_I2C in PMC

Note: There is a SYSMON_I2C unit dedicated to the SYSMON voltage and temperature monitoring, see *Versal ACAP System Monitor Architecture Manual* (AM006).

Features

The features of the I2C controllers include the following:

- Programmable bus data rates
  - Normal
  - Fast
- 16-byte FIFO
- Master mode
  - Transmit and receive with 8 or 10-bit addressing
  - Clock stretching by allowing hold for slow processor service
  - [TO] interrupt flag to avoid stall condition
  - Repeated start
  - Slave monitor mode
- Slave mode
  - Transmit and receive with 7-bit addressing
  - Fully programmable slave response address
  - [HOLD] bit helps to prevent the overflow condition
  - [TO] interrupt flag to avoid stall condition
  - Clock stretching helps to delay communication if data is not readily available
- Software driven controller
  - Status polling
  - Interrupt driven with programmability

Comparison to Previous Generation Xilinx Devices

The I2C controllers are similar to the controller in the Zynq® UltraScale+™ MPSoCs with the addition of the following:

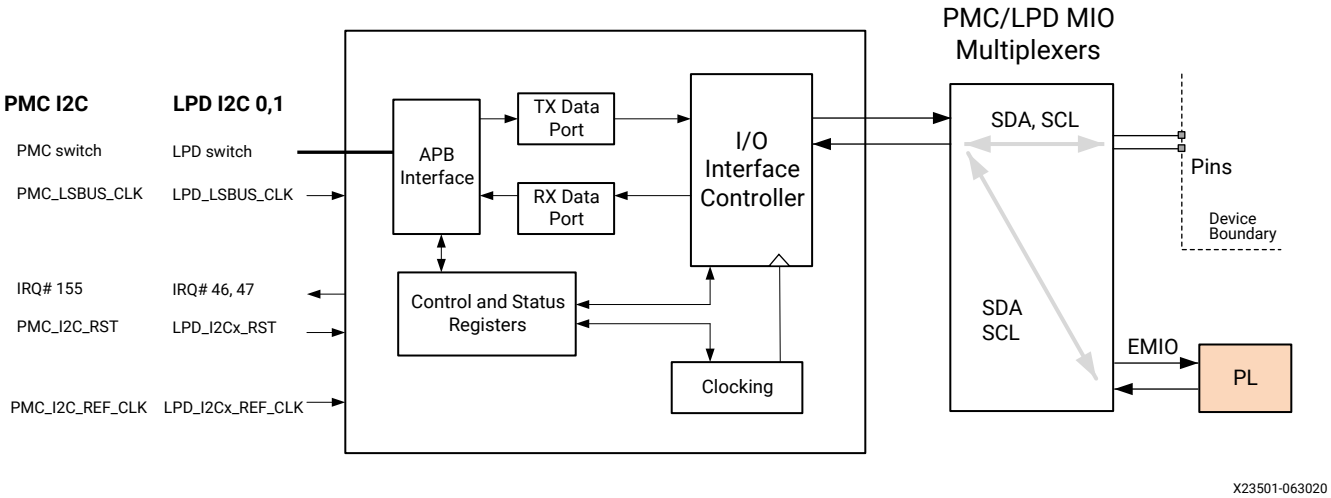
- Data_hold_control register is added to extend the SDA output hold time

System Perspective

Block Diagram

The three instances of I2C in the PMC and PS are shown in the following figure.

Figure 80: **I2C System Block Diagram**



System Interface

The I2C controller has a single 32-bit APB slave programming interface.

APB Programming Interface

The programming interface provides access to the configuration, control, and status registers, as well as data ports for the RX and TX message buffers.

- LPD controllers are attached to the LPD IOP slave switch
- PMC controller is attached to the PMC IOP slave switch

An overview of the controller registers is shown in Register Reference.

APB Interface Clock

The APB interface clock is used by the control and status registers, and the glitch filter.

- PMC_LSBUS_CLK
- LPD_LSBUS_CLK (both controllers)

System Signals

Reference Clock

Each controller receives its own reference clock.

- PMC_I2C_REF_CLK controlled by CRP.PMC_I2C_REF_CTRL
- LPD_I2C0_REF_CLK controlled by CRL.LPD_I2C0_REF_CTRL
- LPD_I2C1_REF_CLK controlled by CRL.LPD_I2C1_REF_CTRL

Controller Reset

The controller has one reset state that is entered when the device, PS, or LPD are reset. The local resets for LPD I2C controllers are included in the CRL.RST_I2C0 and CRL.RST_I2C1 registesr. The PMC I2C controller is reset by the CRP.RST_I2C register.

System Interrupt

Each I2C controller generates an interrupt based on the LPD_I2C.ISR register. The IRQ number are shown in Block Diagram. These IRQ numbers are included in the IRQ System Interrupts table.

I/O Interface

The two wire I/O interface can be routed to the LPD MIO, PMC MIO, or PL EMIO. The specific options are shown in I2C I/O Interface.

Programming Model

The I2C controllers includes a simple memory-mapped APB programming interface for software:

- Control and status
- Transmit data
- Receive data

Programming Model

The following sections describe how to program and use the I2C controller.

Reset Controller

The following register bits can be used to reset a I2C controller.

- PMC I2C: CRP.RST_I2C [RESET]
- LPD I2C0: CRL.RST_I2C0 [RESET]
- LPD I2C1: CRL.RST_I2C1 [RESET]

The [RESET] bits must be toggled by software:

- 0: Run mode
- 1: Reset state

Configure I/O Signals

The SCL and SDA signals can be routed to one of many sets of MIO pins or to the PL EMIO port signal interface by default.

The signal for each MIO pin is routed using SLCR registers:

- PMC_IOP_SLCR.MIO_PIN_0 through MIO_PIN_51
- LPD_IOP_SLCR.MIO_PIN_0 through MIO_PIN_25

The IOP_SLCR.LPD_MIO_SEL [CANx] register bit selects between the PMC and LPD MIO pin multiplexers.

If a MIO PIN register does not map an I2C I/O pin, then the signal is available as an EMIO port interface signal. The SLCR registers also configure the MIO pin buffer input and output characteristics. The I2C I/O signals are listed in I2C I/O Interface.

Configure Clocks

Clocks are generated by the PMC and LPD clock controllers. The clock generators are described in System Perspective.

The I2C controller receives two clocks.

APB Programming Interface Clock

The APB programming interface is driven by the low-speed interconnect clock. Each register includes the three fields [SRCSEL], [DIVISOR], and [CLKACT].

Control registers:

- PMC_I2C: PMC_LSBUS_CLK

- LPD_I2C{0, 1}: LPD_LSBUS_CLK

Reference Clock

The reference clock is used for the controller logic, registers, and memories. It is also an input to the baud rate generator. Each I2C controller has its own reference clock that comes from the PMC or LPD clock generator. Each register includes the three fields [SRCSEL], [DIVISOR], and [CLKACT].

Control registers:

- PMC_I2C: CRP.I2C_REF_CTRL
- LPD_I2C0: CRL.I2C0_REF_CTRL
- LPD_I2C1: CRL.I2C1_REF_CTRL

Controller Configuration

I2C transfer parameters are programmed using the Control registers.

Configure Interrupts

Interrupts help to control data in the FIFO.

Initiate Data Transfers

Transfers are achieved in polled mode or interrupt-driven mode. The limitation on data count while performing a master read transfer is 255 bytes. The next sections show examples of read and write transfer in master mode and an example in slave monitor mode.

Master Read Using Polled Method

1. Set the transfer direction as read and clear the FIFOs. Write `41h` to the Control register.
2. Clear the interrupts. Read and write back the read value of the IRS status register.
3. Write the read data count to the transfer size register and hold bus, if required. Write the read data count value to the Transfer_Size register. If the read data count is greater than the FIFO depth, set Control [HOLD] = `1`.
4. Write the slave address. Write the address to the Address register.
5. Wait for data to be received into the FIFO. Poll on Status [RXDV] = `1`.
    a. If Status [RXDV] = `0`, and any of the following interrupts are set: Interrupt_Status [NACK], Interrupt_Status [ARB_LOST], Interrupt_Status [RX_OVF], or Interrupt_Status [RX_UNF], then stop the transfer and report the error, otherwise continue to poll on the Status [RXDV].

   b. If Status [RXDV] = `1`, and if any of the following interrupts are set: Interrupt_Status [NACK], Interrupt_Status [ARB_LOST], Interrupt_Status [RX_OVF], or Interrupt_Status [RX_UNF], then stop the transfer and report the error. Otherwise, go to step 6.

6. Read the data and update the count. Read the data from the FIFO until Status [RXDV] = `1`. Decrement the read data count and if it is less than or equal to the FIFO depth, clear the Control [HOLD] register.
7. Check for the completion of transfer. If the total read count reaches zero, poll on Interrupt_Status [COMP] = `1`. Otherwise, continue from step 5.

Master Read Using Interrupt Method

1. Set the direction of the transfer as read and clear the FIFOs. Write `41h` to the Control register.
2. Clear the interrupts. Read and write back the read value to the Interrupt_Status register.
3. Enable the timeout, NACK, RX overflow, arbitration lost, DATA, and completion interrupts. Write `22Fh` to the I2C.IER register.
4. Write the read data count to the transfer size register and hold bus, if required. Write the read data count value to the Transfer_Size register. If the read data count is greater than the FIFO depth, set the Control [HOLD] register bit.
5. Write the slave address. Write the address to the Address register.
6. Wait for data to be received into the FIFO.
   a. If the read data count is greater than the FIFO depth, wait for ISR [DATA] bit = `1`. Read 14 bytes from the FIFO. Decrement the read data count by 14 and if it is less than or equal to the FIFO depth, clear the Control [HOLD] register bit.
   b. Otherwise, wait for ISR [COMP] bit = `1` and read the data from the FIFO based on the read data count.
7. Check for the completion of the transfer. Check if the read count reaches zero. Otherwise, repeat from step 6.

Master Write Using Interrupt Method

1. Set the direction of transfer as write and clear the FIFOs. Write `40h` to the Control register.
2. Clear the interrupts. Read and write back the read value to the ISR status register.
3. Enable the timeout, NACK, TX overflow, arbitration lost, DATA, and completion interrupts. Write `24Fh` to the IER interrupt enable register.
4. Enable the bus hold logic. Set Control [HOLD] bit if the write data count is greater than the FIFO depth.
5. Calculate the space available in the FIFO. Subtract the Transfer_Size register value from the FIFO depth.

6. Fill the data into the FIFO. Write the data to the Data register based on the count obtained in step 5.
7. Fill the data into the FIFO. Write the data to the Data register based on the count obtained in step 5.
8. Wait for the data to be sent. Check that the ISR [COMP] bit is set.
    a. If writing further data, repeat steps 5, 6, and 8.
    b. If there is no further data, set Control [HOLD] bit = `0`.
9. Wait for the completion of transfer. Check that the ISR [COMP] register bit is set = `1`.

Slave Monitor Mode

The slave monitor mode helps to monitor when the slave is in the busy state. The slave ready interrupt occurs only when the slave is not busy. This process can only be performed in master mode.

1. Select slave monitor mode and clear the FIFOs. Write `60h` to the Control register.
2. Clear the interrupts. Read and write back the read value to the ISR status register.
3. Enable the interrupts. Set the IER [SLV_RDY] bit = `1`.
4. Set the slave monitor delay. Write `Fh` to the Slave_Mon_Pause register.
5. Write the slave address. Write the address to the Address register.
6. Wait for the slave to be ready. Poll on ISR [SLV_RDY] status register bit until = `1`.

Programming Sequences

The flow diagram for the I2C controller programming sequence is shown in the following two figures.

Figure 81: **I2C Master Interrupt Example Flowchart**

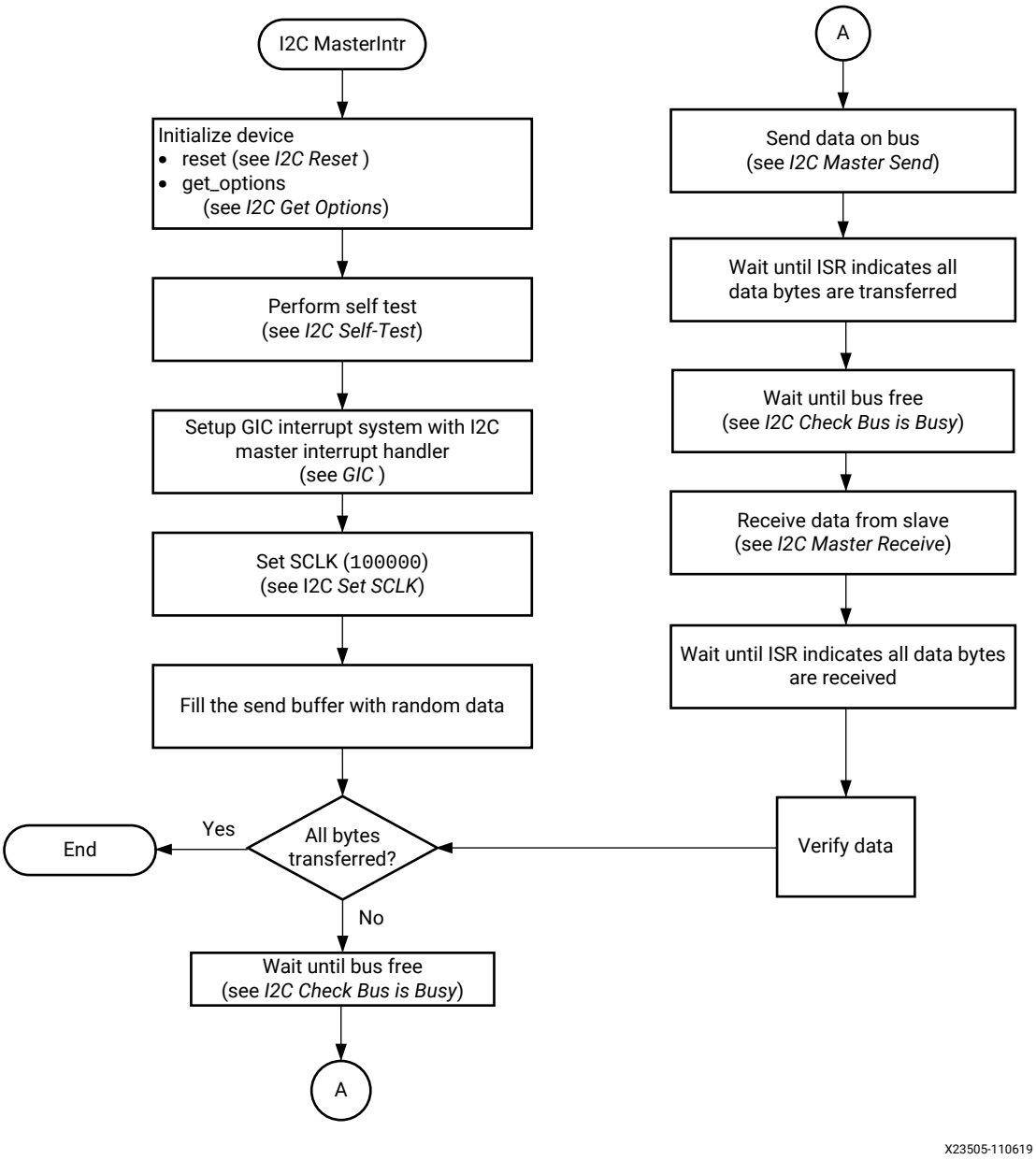
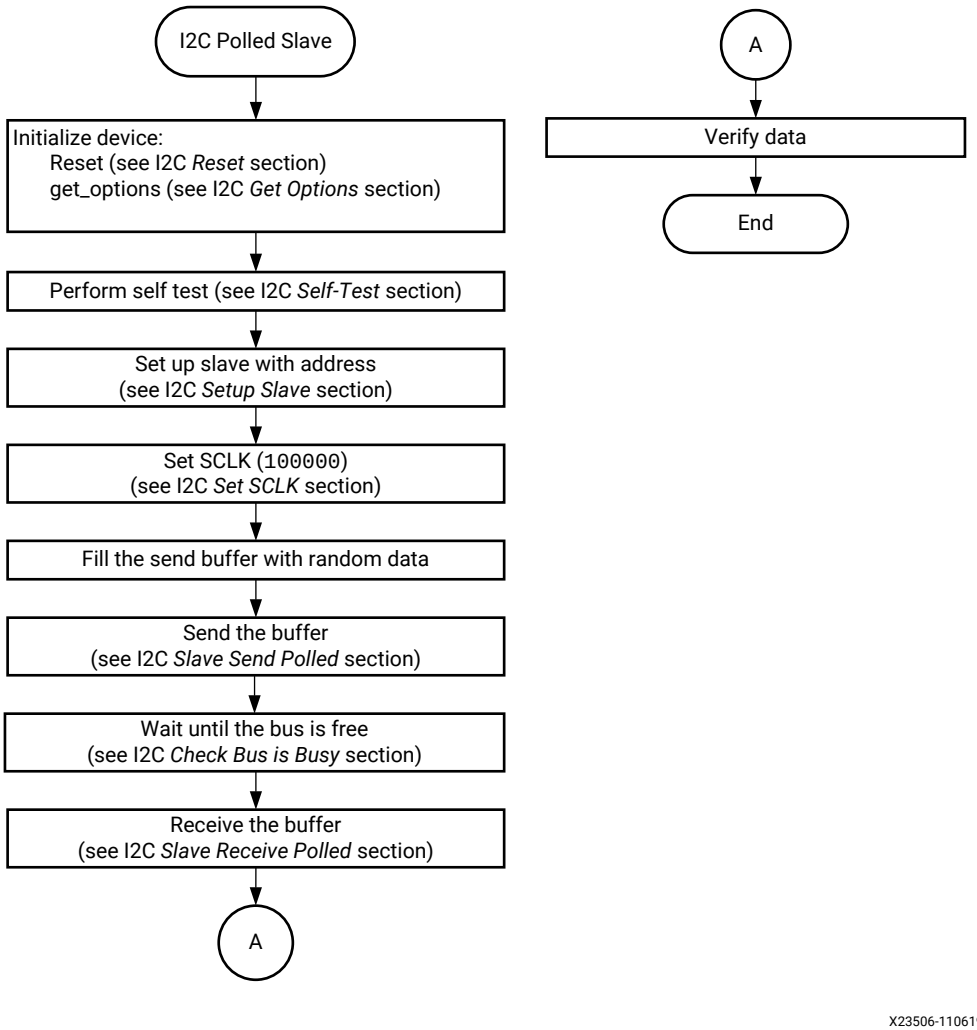


Figure 82: **I2C Slave Polled Example Flowchart**



Software Routines

Reset

Table 137: **I2C Reset**

Task	Register	Register Field	Bits	Operation
Abort Start				
Save interrupt mask register	IMR, 0x20	All	9:0	Read operation
Disable all interrupts	IDR, 0x28	All	9:0	Write 2FFh

Table 137: **I2C Reset** *(cont'd)*

Task	Register	Register Field	Bits	Operation
Reset configuration and clear FIFOs	Control, 0x00	All	15:0	Write 40h
Read interrupt status register	ISR, 0x10	All	9:0	Read operation
Write back interrupt status register	ISR, 0x10	All	9:0	Clear bits detected as set
Restore interrupt state	IER, 0x24	All	9:0	0x2FF and ~IMR
Abort End				
Reset configuration	Control, 0x00	All	15:0	Write 0h
Reset time out	Time_Out, 0x1C	All	7:0	Write FFh
Disable all interrupts	IDR, 0x28	All	9:0	Write 2FFh

Get Options

Table 138: **I2C Get Options**

Task	Register	Register Field	Bits	Operation
Read control register	Control, 0x00	All	15:0	Read operation

Check Bus is Busy

Table 139: **I2C Check Bus is Busy**

Task	Register	Register Field	Bits	Operation
Read bus active state	Status, 0x04	BA	8	Read operation
If set bus is busy, else bus is free.				

Transmit FIFO Fill

Table 140: **I2C Transmit FIFO Fill**

Task	Register	Register Field	Bits	Operation
Read transfer size register	Transfer_Size, 0x14	Transfer_Size	7:0	Read operation
Calculate available bytes = FIFO DEPTH(16) – Transfer_Size.				
Fill data register with the data until available bytes count is reached. See Send Byte.				

Send Byte

Table 141: **I2C Send Byte**

Task	Register	Register Field	Bits	Operation
Write byte into data register	Data, 0x0C	DATA	7:0	Write data

Reset Hardware

Table 142: **I2C Reset Hardware**

Task	Register	Register Field	Bits	Operation
Disable all interrupts	IDR, 0x28	All	9:0	2FFh
Clear Interrupt Status				
Read interrupt status register	ISR, 0x10	All	9:0	Read operation
Write back interrupt status register	ISR, 0x10	All	9:0	Clear bits detected as set
Clear hold, master enable, and acknowledge bits.				
Read control register	Control, 0x00	All	15:0	Read operation
Clear bits	Control, 0x00	CLR_FIFO, HOLD, ACK_EN, MS	6, 4, 3, and 1	(~(0x0015) | 0x0040) (hex)
Reset time out	Time_Out, 0x1C	All	7:0	FFh
Clear transfer size register	Transfer_Size, 0x14	Transfer_Size	7:0	Write 00h
Clear status register				
Read status register	ISR, 0x04	All	8:0	Read operation
Write back status register	ISR, 0x04	All	8:0	Read value
Reset configuration register	Control, 0x00	All	15:0	Write 0000h

Setup Master

Table 143: **I2C Setup Master**

Task	Register	Register Field	Bits	Operation
Read control register	Control, 0x00	All	15:0	Read operation
If [HOLD] is set = 1, then check if bus is busy (see Check Bus is Busy). If bus is busy, return.				
Setup master	Control, 0x00	CLR_FIFO, HOLD, ACK_EN, NEA, MS	6, 4, 3, 2,	
			and 1	5Eh
For Receiver Role				
Enable master receiver	Control, 0x00	RW	0	1

Table 143: **I2C Setup Master** *(cont'd)*

Task	Register	Register Field	Bits	Operation
For Transmitter Role				
Enable master transmitter	Control, 0x00	RW	0	0
Disable all interrupts	IDR, 0x28	All	9:0	2FFh

Master Send

Table 144: **I2C Master Send**

Task	Register	Register Field	Bits	Operation
Set repeated start if data is more than FIFO depth				
Set hold bit	Control, 0x00	HOLD	4	1
Setup master for transmitter role (see Setup Master).				
Transmit FIFO full (see Transmit FIFO Fill).				
Program transfer address	Address, 0x08	ADD	9:00	Address
Enable interrupts	IER, 0x24	ARB_LOST, NACK, COMP	9, 2, and 0	205h

Master Receive

Table 145: **I2C Master Receive**

Task	Register	Register Field	Bits	Operation
Set repeated start if data is more than FIFO depth.				
Set hold bit	Control, 0x00	HOLD	4	1
Setup master for receiver role (see Setup Master).				
Program transfer address	Address, 0x08	ADD	9:00	Write address
Setup transfer size	Transfer_Size, 0x14	Transfer_Size	7:00	Required transfer size
Enable interrupts	IER, 0x24	ARB_LOST, RX_OVF, NACK, COMP	9, 5, 2, and 1	
			0	227h

Master Send Polled

Table 146: **I2C Master Send Polled**

Task	Register	Register Field	Bits	Operation
Set repeated start if data is more than FIFO depth.				
Set hold bit	Control, 0x00	HOLD	4	1

Table 146: **I2C Master Send Polled** *(cont'd)*

Task	Register	Register Field	Bits	Operation
Setup master for transmitter role (see Setup Master).				
Read interrupt status register	ISR, 0x10	All	9:0	Read operation
Write back interrupt status register	ISR, 0x10	All	9:0	Clear bits detected as set
Transmit first FIFO full of data (see Transmit FIFO Fill).				
Program transfer address	Address, 0x08	ADD	9:0	Address
Read interrupt status register	ISR, 0x10	All	9:0	Read operation
Perform the following steps as long as no errors are reported by hardware from the status register read and total bytes are sent.				
Read status register	Status, 0x04	All	8:0	Read operation
Read interrupt status register	ISR, 0x10	All	9:0	Read operation
Transmit first FIFO full of data (see Transmit FIFO Fill).				
Check for transfer completion				
Read interrupt status register	ISR, 0x10	All	9:0	Read operation
If any error reported by hardware transfer failed.				
Clear hold bit if not repeated start operation	Control, 0x00	HOLD	4	0

Master Receive Polled

Table 147: **I2C Master Receive Polled**

Task	Register	Register Field	Bits	Operation
Set repeated start if data is more than FIFO depth.				
Set hold bit	Control, 0x00	HOLD	4	1
Setup master for receiver role (see Setup Master).				
Read interrupt status register	ISR, 0x10	All	9:0	Read operation
Write back interrupt status register	ISR, 0x10	All	9:0	Clears bits detected as set
Transfer address	Address, 0x08	ADD	9:0	Address
Program transfer size	Transfer_Size, 0x14	Transfer_Size	7:0	Required transfer size
Read interrupt status register	ISR, 0x10	All	9:0	Read operation
Start Loop 1: perform the following steps as long as receiving bytes and no errors reported from hardware.				
Read status register	Status, 0x04	All	8:0	Read operation
Start Loop 2: perform the following steps as long as RXDV bit is non zero in SR.				

Table 147: **I2C Master Receive Polled** *(cont'd)*

Task	Register	Register Field	Bits	Operation
Clear repeat start if receive byte count is less than 14	Control, 0x00	HOLD	4	0
Receive byte	Data, 0x0C	DATA	7:0	Read operation
Read status register	Status, 0x04	All	8:0	Read operation
End Loop 2				
If receive byte count is >0 and bytes still need to be received.				
Read interrupt status register	ISR, 0x10	All	9:0	Read operation
Write back interrupt status register	ISR, 0x10	All	9:0	Clears bits detected as set
If receive byte count > maximum transfer size, then program transfer size	Transfer_Size, 0x14	Transfer_Size	7:0	Maximum transfer size
Else program with required transfer size	Transfer_Size, 0x14	Transfer_Size	7:0	Required transfer size
Read interrupt status register	ISR, 0x10	All	9:0	Read operation
End Loop 1				
Clear hold bit if not repeated start operation	Control, 0x00	HOLD	4	0
If any error reported by hardware transfer failed else transfer success.				

Enable Slave Monitor

Table 148: **I2C Enable Slave Monitor**

Task	Register	Register Field	Bits	Operation
Clear transfer size register	Transfer_Size, 0x14	Transfer_Size	7:0	0
Enable slave monitor mode	Control, 0x00	MS | NEA| CLR_FIFO | SLVMON	15:0	0066h
Enable slave monitor interrupt	IER, 0x24	SLV_RDY	4	1
Initialize slave monitor register	Slave_Mon_Pause, 0x18	Pause	3:0	Fh
Program transfer address	Address, 0x08	ADD	9:0	Address

Disable Slave Monitor

Table 149: **I2C Disable Slave Monitor**

Task	Register	Register Field	Bits	Operation
Disable slave monitor mode	Control, 0x00	SLVMON	5	0
Disable slave monitor interrupt	IER, 0x24	SLV_RDY	4	0

Master Send Data

Table 150: **I2C Master Send Data**

Task	Register	Register Field	Bits	Operation
Transmit first FIFO full of data (see Transmit FIFO Fill).				
Set repeated start bit if requested	Control, 0x00	HOLD	4	1

Master Interrupt Handler

Table 151: **I2C Master Interrupt Handler**

Task	Register	Register Field	Bits	Operation
Read interrupt status register	ISR, 0x10	All	9:0	Read operation
Write back interrupt status register	ISR, 0x10	All	9:0	Clear bits detected as set
Get the enabled interrupts	IMR, 0x20	All	9:0	Read operation
ISR & IMR				
Check if hold bit is set (isHold)	Control, 0x00	HOLD	4	Read operation
If send operation && (ISR & [COMP])				
Send data (see Master Send Data).				
If receive operation && (ISR & [COMP]) || (ISR & [DATA).				
Perform the following operations until receive data valid mask is set (loop-1 started).				
Read status register	Status, 0x04	All	8:0	Read operation
Clear hold bit if not needed	Control, 0x00	HOLD	4	0
Receive byte	Data, 0x0C	DATA	7:0	Read operation
Loop-1 Ended				
If receive byte count is >0 and bytes still need to be received.				
Read interrupt status register	ISR, 0x10	All	9:0	Read operation

Table 151: **I2C Master Interrupt Handler** *(cont'd)*

Task	Register	Register Field	Bits	Operation
Write back interrupt status register	ISR, 0x10	All	9:0	Clear bits detected as set
If receive byte count > maximum transfer size then setup transfer size	Transfer_Size, 0x14	Transfer_Size	7:0	Maximum transfer size
Else program with required transfer size	Transfer_Size, 0x14	Transfer_Size	7:0	Required transfer size
Enable interrupts	IER, 0x24	ARB_LOST, RX_OVF, NACK, DATA, COMP	9, 5, 2, 1 and 0	227h
Clear hold bit if all interrupts attended	Control, 0x00	HOLD	4	0
Clear hold bit if slave ready interrupt is triggered	Control, 0x00	HOLD	4	0
Clear hold bit if any other interrupts occurred (event errors)	Control, 0x00	HOLD	4	0

Setup Slave

Table 152: **I2C Setup Slave**

Task	Register	Register Field	Bits	Operation
Clear ack_en, nea, FIFO, and set master in slave mode	CONTROL, 0x00	CLR_FIFO, ACK_EN, NEA, MS	6, 3, 2, and 1	2Ch
Disable all interrupts	IDR	All	9:0	2FFh
Transfer address	ADDR, 0x08	ADD	9:0	Address

Slave Send

Table 153: **I2C Slave Send**

Task	Register	Register Field	Bits	Operation
Enable interrupts	IER, 0x24	TX_OVF, TO, NACK, DATA, COMP	6, 3, 2, 1, and 0	4Fh

Slave Receive

Table 154: **I2C Slave Receive**

Task	Register	Register Field	Bits	Operation
Enable interrupts	IER, 0x24	RX_UNF, RX_OVF, TO, NACK, DATA, COMP	7, 5, 3, 2, 1, and 0	AFh

Slave Send Polled

Table 155: **I2C Slave Send Polled**

Task	Register	Register Field	Bits	Operation
Use RXRW bit in status register to wait master to start a read.				
Read status register	Status, 0x04	All	8:0	Read operation
Check the RXRW bit is set by reading status register continuously. If master tries to send data, it is an error.				
Read interrupt status register	ISR, 0x10	All	9:0	Read operation
Write back interrupt status register	ISR, 0x10	All	9:0	Clear bits detected as set
Send data as long as there is more data to send and there are no errors (see Send Byte).				
Read status register	Status, 0x04	All	8:0	Read operation
Wait for master to read the data out of the TX FIFO; [SR] & [TXDV] != 0 and there are no errors.				
Read interrupt status register	ISR, 0x10	All	9:0	Read operation
If master terminates the transfer before all data is sent, it is an error (interrupt status register and NACK).				
Write back interrupt status register	ISR, 0x10	All	9:0	Clear bits detected as set

Slave Receive Polled

Table 156: **I2C Slave Receive Polled**

Task	Register	Register Field	Bits	Operation
Read status register	Status, 0x04	All	8:0	Read operation
Read interrupt status register	ISR, 0x10	All	9:0	Read operation
Write back interrupt status register	ISR, 0x10	All	9:0	Clear bits detected as set
Read status register	Status, 0x04	All	8:0	Read operation
Write back status register	Status, 0x04	All	8:0	Write status
Read status register	Status, 0x04	All	8:0	Read operation
Perform the following operations until all bytes received (Loop-1 started).				
Perform the following operations as long as SR and RXDV = 0 (Loop-2 started).				
Read status register	Status, 0x04	All	8:0	Read operation
If (status register and (DATA | COMP) != 0) && (status register and RXDV ==0) && receive byte count >0) then it is a failure.				
Write back interrupt status register	ISR, 0x10	All	9:0	Clear bits detected as set
Loop-2 ended				
Perform the following operations until status register and RXDV!= 0 and receive byte count!=0 (Loop-3 started).				
Receive byte	Data, 0x0C	DATA	7:0	Read operation

Table 156: **I2C Slave Receive Polled** *(cont'd)*

Task	Register	Register Field	Bits	Operation
Read status register	Status, 0x04	All	8:0	Read operation
Loop-3 ended				
Loop-1 ended				

Receive Data

Table 157: **I2C Receive Data**

Task	Register	Register Field	Bits	Operation
Read status register	Status, 0x04	All	8:0	Read operation
Until (status register and RXDV) && receive byte count !=0 (Loop -1 started).				
Receive byte	Data, 0x0C	DATA	7:0	Read operation
Read status register	Status, 0x04	All	8:0	Read operation
Loop-1 ended				

Slave Interrupt Handler

Table 158: **I2C Slave Interrupt Handler**

Task	Register	Register Field	Bits	Operation
Read interrupt status register	ISR, 0x10	All	9:0	Read operation
Write the status back to clear the interrupts so no events are missed while processing this interrupt.				
Write back interrupt status register	ISR, 0x10	All	9:0	Clear bits detected as set
Get the enabled interrupts (imr)	IMR, 0x20	All	9:0	Read operation
Use the mask register AND with the interrupt status register so disabled interrupts are not processed (~(imr) and IntrStatusReg). Data interrupt (if interrupt status register and data): • Master wants to perform more data transfers. • Check for completion of transfer; signal upper layer if done. For sending transmit FIFO fill (see Transmit FIFO Fill). Else receive slave data (see Slave Receive).				

Set and Clear Options

Table 159: **I2C Set and Clear Options**

Task	Control Register Field (offset 0x00)	Bits	Set Option	Clear Option
For 7-bit address option	NEA	2	1	0
For 10-bit address option	NEA	2	0	1
Slave monitor option	SLVMON	5	1	0
For repeated start option	HOLD	4	1	0

Set SCLK

Table 160: **I2C Set SCLK**

Task	Register	Register Field	Bits	Operation
Read transfer size register	Transfer_Size, 0x14	Transfer_Size	7:0	Read operation
If the Transfer_Size register is not = 0, then stop here. If the device is currently transferring data, the transfer must complete or be aborted before setting options.				
Make sure clock option is within range I2C_SCL >0. Calculate values for divisor_a (best_divA) and divisor_b (best_divB).				
Program the divisor values	Control, 0x00	divisor_a | divisor_b	15:8	Best_divA | best_divB

Get CLK

Table 161: **I2C Get CLK**

Task	Register	Register Field	Bits	Operation
Read the divisor values (Div_a | Div_b)	Control, 0x00	divisor_a | divisor_b	15:8	Read operation
Calculate actual clock value = (input clock / (22U x (Div_a + 1U) x (Div_b + 1U)).				

Self-Test

Table 162: **I2C Self-Test**

Task	Register	Register Field	Bits	Operation
All I2C registers should be in their default state.				
Read control register (CR)	Control, 0x00	All	15:0	Read operation

Table 162: **I2C Self-Test** *(cont'd)*

Task	Register	Register Field	Bits	Operation
Read interrupt mask register (imr)	IMR, 0x20	All	9:00	Read operation
If (CR != 0) OR if (IMR != 0x2FF), stop here.				
Perform reset (see Reset Hardware).				
Write test value (0x05) into slave monitor register	Slave_Mon_Pause, 0x18	Pause	3:0	5h
Read back slave monitor register	Slave_Mon_Pause, 0x18	Pause	3:0	Read operation
Verify the value with the written value. If not the same, test failed; else passed.				
Reset slave monitor register	Slave_Mon_Pause, 0x18	Pause	3:0	0h

Register Reference

Register overview tables:

- I2C Registers
- SLCR I/O Interface Registers
- System-level Clock and Reset Registers

The registers are programmed by software through the APB slave interface.

Interrupt status and control registers detect events and monitor system state to generate system interrupts.

I2C Registers

Each controller has a set of I2C registers. The PMC has a separate register set, PMC_I2C from the LPD controllers, LPD_I2C. The controllers have identical functionality with separate memory-mapped register base address locations:

- PMC_I2C base address is 0xF100_0000
- LPD_I2C0 base address is 0xFF02_0000
- LPD_I2C1 base address is 0xFF03_0000

The registers are listed in the following table.

Table 163: **I2C Register Overview**

Register Name	Address Offset	Access Type	Description
CTRL	0x000	R/W	I/O protocol, clock divider
STATUS	0x004		Read data available
ADDR DATA	0x008 0x00C	R/W	Address; 7 or 10-bit field 8-bit data field
TRANS_SIZE	0x014	R/W	0 to 255 transfer size
SLAVE_PAUSE	0x018	R/W	0 to 7 pause interval
TIME_OUT	0x01C	R/W	32 to 127 timeout interval
ISR IMR IER IDR	0x010 0x020 0x024 0x028	R, W1C R W W	Interrupts: status is after mask. Enabled interrupts are OR'ed together and generate a system interrupt.
GLITCH_CTRL	0x02C	R/W	Glitch filter control
DATA_HOLD_CTRL	0x030	R/W	Data hold control

SLCR I/O Interface Registers

The system-level registers are summarized in the following table. These registers are used to route the two I/O signals to MIO device pins.

Table 164: **I2C SLCR Registers**

Register Name	Address	Access Type	Description
PMC_IOP_SLCR.MIO_PIN_0 +	0xF106_0000 +	RW	52 Registers: PMC MIO pin 0 to PMC MIO pin 51
LPD_IOP_SLCR.MIO_PIN_0 +	0xFF08_0000 +	RW	26 Registers: LPD MIO pin 0 to LPD MIO pin 25

System-level Clock and Reset Registers

The clock and reset control registers are summarized in the following table.

Table 165: **I2C Clock and Reset Registers**

Register Name	Address	Access Type	Description	Signal Name
PMC_I2C				
CRP.I2C_REF_CTRL	0xF126_0130	RW	Reference clock	PMC_I2C_REF_CLK
CRP.RST_I2C	0xF126_0314	RW	Controller reset	PMC_I2C_RESET

Table 165: **I2C Clock and Reset Registers** *(cont'd)*

Register Name	Address	Access Type	Description	Signal Name
LPD_I2C0				
CRL.I2C0_REF_CTRL	0xFF5E_0140	RW	Reference clock	LPD_I2C0_REF_CLK
CRL.RST_I2C0	0xFF5E_0330	RW	Controller reset	LPD_I2C0_RESET
LPD_I2C1				
CRL.I2C1_REF_CTRL	0xFF5E_0144	RW	Reference clock	LPD_I2C1_REF_CLK
CRL.RST_I2C1	0xFF5E_0334	RW	Controller reset	LPD_I2C1_RESET

I2C I/O Interface

The I2C controller I/O interface is routed to both the PMC and LPD MIOs, and the EMIO. The MIO signals are shown in the MIO-at-a-Glance Tables and all signals are detailed in the following table.

Table 166: **I2C Controller I/O Signals**

MIO					EMIO Signals	
Signal Name	**I/O**	**PMC MIO Pin**	**LPD MIO Pin**	**MIO-at-a-Glance Table**	**Signal Name**	**I/O**
LPD I2C0 Controller						
LPD_I2C0_SCL	I/O	MIO-at-a-Glance Tables		0		
LPD_I2C0_SDA	I/O			1		
LPD I2C1 Controller						
LPD_I2C1_SCL	I/O	MIO-at-a-Glance Tables		0		
LPD_I2C1_SDA	I/O			1		
PMC I2C Controller						
PMC_I2C_SCL	I/O	MIO-at-a-Glance Tables		0		
PMC_I2C_SDA	I/O			1		
SYSMON I2C Controller						
SYSMON_I2C_SCL	I/O	MIO-at-a-Glance Tables		0		
SYSMON_I2C_SDA	I/O			1		
SYSMON_I2C_ALERT	I/O			2		

SPI Controller

The SPI bus controller enables communications with a variety of peripherals such as memories, temperature sensors, pressure sensors, analog converters, real-time clocks, displays, and any SD card with serial mode support. The SPI controller can function in master mode with multi-master feature, slave mode, or loop-back test mode.

There are two instances of the SPI controller. Both the controllers are identical and independent. The SPI controllers are located in the LPD power domain of the PS.

The I/O interfacing options include the PMC and LPD MIO pins.

Software accesses the control and status registers via the APB slave programming interface on the LPD IOP switch.

Features

The controller provides these features:

- Full-duplex operation offers simultaneous receive and transmit
- Four wire SPI bus: MISO, MOSI, SCLK, SS_b
- Master with multi-master feature, slave, and loop-back modes
- Three slave selects in master mode with expansion to eight with external 3:8 decoder
- Multi-master environment: identifies an error condition if more than one master detected
- Control and status registers are accessible via the APB programming interface
- Data ports for RX and TX data mapped to the register set
- Buffered operations with separate RX and TX FIFOs
- Programmable master-mode clock frequencies
- Serial clock with programmable polarity
- Programmable transmission format
- FIFO level status read registers
- FIFO level interrupts with programmable RX and TX FIFO thresholds

Comparison to Previous Generation Xilinx Devices

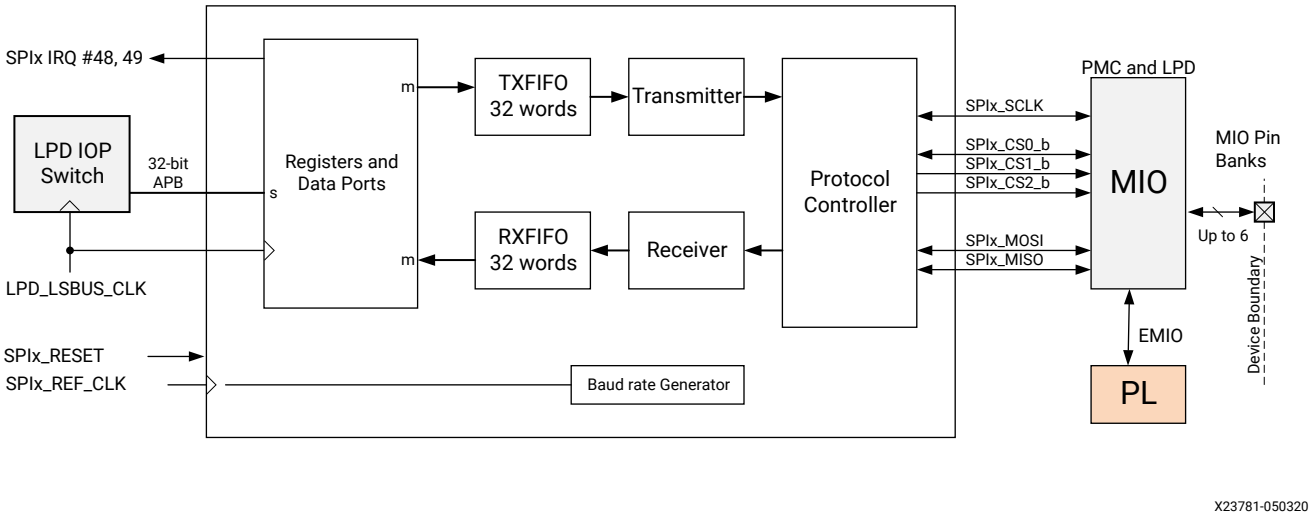
The functionality of the SPI controllers in the Versal™ ACAP is the same as in the Zynq® UltraScale+™ MPSoC.

System Perspective

Block Diagram

The following figure shows the SPI high-level block diagram.

Figure 83: **SPI High-Level Block Diagram**



The controller includes several functional units:

- TX FIFO and transmitter datapath
- RXFIFO and receiver datapath
- Master and slave protocol units

System Interface

The SPI controller has a single 32-bit APB programming interface.

APB Programming Interface

The programming interface provides access to the configuration, control, and status registers. An overview of the controller registers is shown in Register Reference.

Interface Clock

The controller is clocked by the APB programming interface. The interface is clocked by the LPD_LSBUS_CLK clock.

System Signals

Reference Clock

The controller receives its reference clock from the LPD clock controller:

- SPI0_REF_CLK
- SPI1_REF_CLK

The reference clock operates the controller, and for master mode, it provides an input to the baud rate divider for the SPI_SCLK I/O output clock. Refer to Clocking for an explanation of all the SPI clocks.

Controller Reset

The controller has a single reset state. The SPI_RESET signal is controlled by the LPD reset controller. See the CRL CRL.RST_SPI0 register.

I/O Interface

Master Mode SCLK

In master mode, the interface is clocked by the controller-generated SCLK that is derived from the SPIx_REF_CLK using the baud rate divider. The divider is programmed using the SPI.Config [BAUD_RATE_DIV] bit field. The range of the baud rate divider is from a minimum of 4 to a maximum of 256 in binary steps (i.e., divide by 4, 8, 16, 32,... 256).

Master Mode Clock Requirement

The external device must synchronously drive the signal inputs to the SCLK output clock. The clock frequency specifications are defined in the Versal ACAP data sheets.

Slave Mode SCLK

In slave mode, the external device generates the SCLK. The controller samples the input signals and drives the MISO signal using the SCLK from the attached master. The input signals are synchronized to the SPIx_REF_CLK and then interpreted by the protocol controller.

Programming Model

The programming model is defined in the SPI register set, which includes control and status plus TX and RX data ports.

The programming model can be divided into these sections:

- Configuration and mode control
- Master mode
- Slave mode
- Data Loopback Mode

Modes and States

The SPI controller operates in three modes:

- Master mode with multi-master feature
- Slave mode
- Data loopback mode

***Note*:** There are clock ratio requirements for the SPI_REF_CLK and the APB programming interface clock. The ratio depends on the operating mode and are defined in Clocking.

Master Mode

In master mode, the SPI I/O interface transmits data to a slave or initiates a transfer to receive data from a slave. In this mode, the controller drives the serial clock and slave selects with an option to provide a multi-master functionality. The serial clock is derived from the SPI_REF_CLK from the LPD clock controller.

The SPI selects one slave device at a time using one of the three slave select lines. If more than three slave devices need to be connected to the master, a 3-to-8 decoder can be added on the MIO or EMIO interface. The multiplexer is enabled using the SPI.CONFIG [PERI_SEL] bit.

The controller initiates messages using up to three individual slave select output signals that can be externally expanded. The controller reads and writes to the slave devices by writing bytes to the 32-bit read/write data port register.

Multi-master Functionality

For multi-master, the controller is programmed for master mode [MODE_SEL] and can initiate transfers on any of the slave selects. When the software is ready to initiate a transfer, it enables the controller using the [SPI_EN] bit. When the transaction is finished, the software disables the controller. The controller cannot be selected by an external master when the controller is in master mode.

When the multi-master feature is enabled, the controller's output signals are 3-stated when the controller is not active. The controller detects another master on the bus by monitoring the open-drain slave select signal (active-Low). The detection mechanism is enabled by the [Modefail_gen_en]. When the controller detects another master:

- Tristates the I/O outputs
- Sets the SPI.ISR [MODE_FAIL] interrupt status bit to indicate the fault
- Clears the SPI.ENABLE [SPI_EN] control bit

The [MODE_FAIL] interrupt enables the software to abort the transfer, reset the controller, and resend the transfer.

Slave Mode

In slave mode, the controller receives the serial I/O clock from the master device and uses the SPI_REF_CLK to synchronize data capture in the controller.

The slave mode includes a programmable start detection mechanism when the controller is enabled while the slave select (SS) signal is asserted. The read and write FIFOs provide buffering between the SPI I/O interface and the software servicing the controller via the APB slave interface. The FIFOs are used for both slave and master I/O modes.

Data Loopback Mode

For data loopback, the I/O signals of the two controllers are connected together: the clock, slave select, MISO, and MOSI signals from one controller are connected to the other controller's clock, slave, MISO, and MOSI signals, respectively. This connection is internal to the controller and does not use any MIO pins.

The loopback mode is selected by setting the LPD_IOP_SLCR.MIO_LOOPBACK [SPI0_LOOP_SPI1] bit = 1.

Clocking

There are three clocks associated with a SPI controller:

- SPI_REF_CLK reference clock
- LPD_LSBUS_CLK programming interface, APB
- I/O SCLK on SPI bus

Reference Clock

Each controller is provided a SPI_REF_CLK from the LPD clock controller. This clock is used by the majority of the controller logic. Reference clocks are programmed by the LPD clock controller using the CRL.LPD_SPI0_REF_CTRL and CRL.LPD_SPI_REF_CTRL registers.

I/O SCLK Clock

The I/O SCLK is generated by dividing down the SPI_REF_CLK using the SPI.CONTROL [BAUD_DIV] bit field. The divide-down ratio options include /4, /8, /16, ..., /256.

APB Programming Interface Clock

The controller is also clocked by the LPD_LSBUS_CLK for the APB programming interface; this clock is common to all LPD controllers on the IOP switch and is controlled by the CRL.LPD_LSBUS_CTRL register.

Ratio Requirements for Reference and APB Clocks

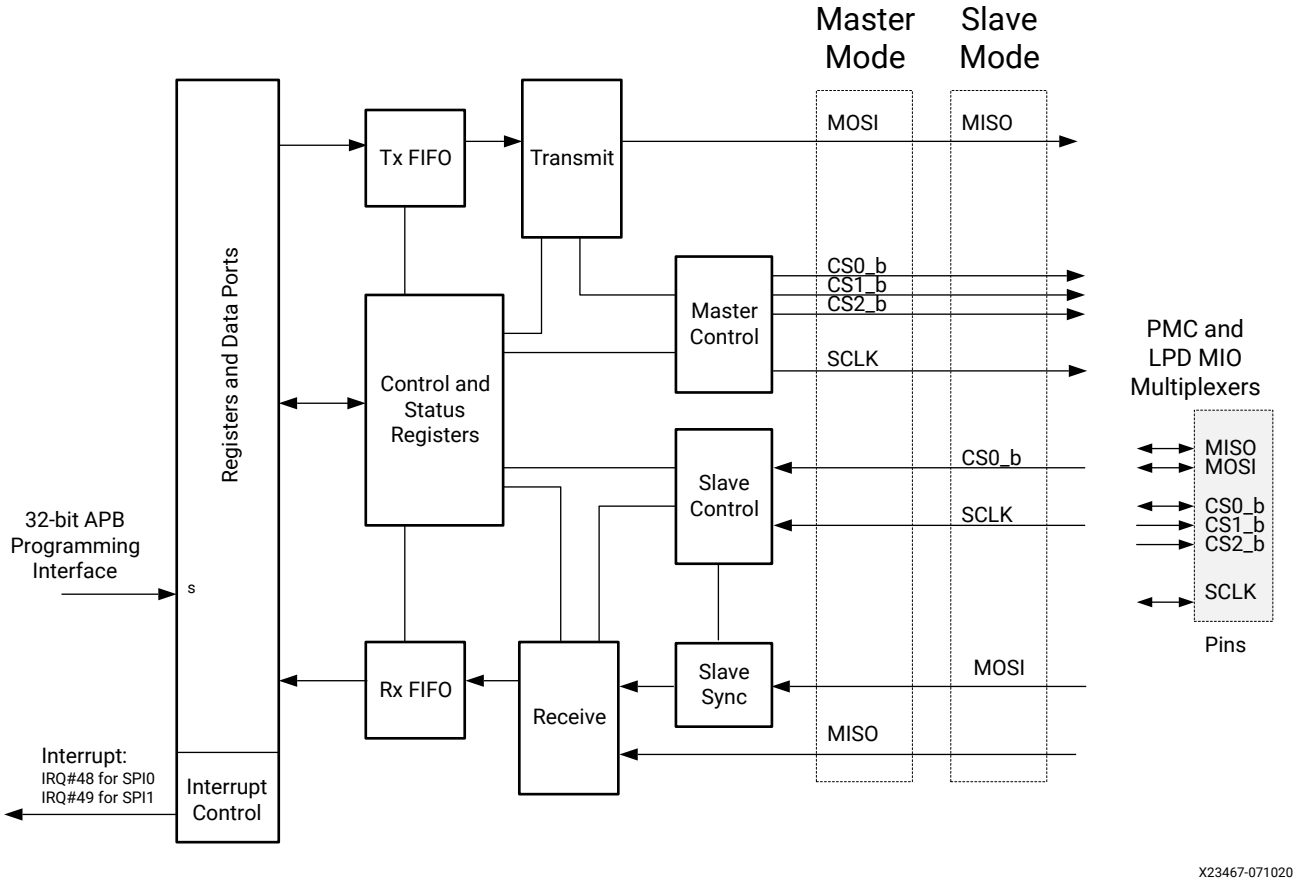
There is a minimum clock frequency ration between the SPI_REF_CLK and LPD_LSBUS_CLK. The ration depends on the operating mode:

- Master mode clock ratio: 4:1, minimum
- Slave mode clock ration 2:1, minimum

Functional Diagram

The following figure shows the SPI controller functional block diagram. There is also a higher-level system Block Diagram.

Figure 84: **SPI Functional Diagram**



FIFOs

The RX and TX FIFOs are each 128-bytes deep. Software reads and writes these FIFOs using the data port registers RX_data and TX_data.

RXFIFO

If the controller hardware attempts to push data into a full RXFIFO, the data is lost and the sticky overflow interrupt flag is set. No data is added to a full RXFIFO. Software writes a 1 to the interrupt to clear the SPI.ISR [RX_OVERFLOW] bit.

TXFIFO

If software attempts to write data into a full TXFIFO, the write is ignored. No data is added to a full TXFIFO. The SPI.ISR [TX_FIFO_full] bit is asserted until the TXFIFO is read and the TXFIFO is no longer full. If the TXFIFO overflows, the sticky [RX_OVERFLOW] bit is set = 1.

Data Transfer

The SPI controller follows a specific series of operations to initiate and control the data transfers on the SPI bus. This section details the data transfer handshake mechanisms.

Data Transfer

The SCLK clock and MOSI signals are under control of the master. Data to be transmitted is written into the TXFIFO by software using register writes and then unloaded for transmission by the controller hardware in a manual or automatic start sequence. Data is driven onto the master output (MOSI) data pin. Transmission is continuous while there is data in the TXFIFO. Data is received serially on the MISO data pin and is loaded eight bits at a time into the RXFIFO. Software reads the RXFIFO using register reads. For every n bytes written to the TXFIFO, there are n bytes stored in RXFIFO that must be read by software before starting the next transfer.

Auto/Manual Slave Select and Start

Data transfers on the I/O interface can be manually started using software or automatically started by the controller hardware. Also, the slave select assertion/deassertion can be controlled by the hardware or the software.

- Manual slave select

  Software selects the manual slave select method by setting the SPI.CONFIG [Manual_CS] bit = 1. In this mode, software must explicitly control the slave select assertion/deassertion. When the [Manual_CS] bit = 0, the controller hardware automatically asserts the slave select during a data transfer.

- Automatic slave select

  Software selects the auto slave select method by programming the [Manual_CS] bit = 0. The SPI controller asserts/deasserts the slave select for each transfer of TXFIFO content on to the MOSI signal. Software writes data to the TXFIFO and the controller asserts the slave select automatically, transmits the data in the TXFIFO, and then deasserts the slave select. The slave select gets deasserted after all the data in the TXFIFO is transmitted. This is the end of the transfer. Software ensures the following in automatic slave select mode.

  - Software continuously fills the TXFIFO with the data bytes to be transmitted, without the TXFIFO becoming empty, to maintain an asserted slave select
  - Software continuously reads data bytes received in the RXFIFO to avoid overflow

Software uses the TXFIFO and RXFIFO threshold levels to avoid FIFO under- and over-flows. The TXFIFO's not-full condition is flagged when the number of bytes in TXFIFO is less than the TXFIFO threshold level. The RXFIFO full condition is flagged when the number of bytes in RXFIFO is equal to 128.

Manual Start

This section describes how to start data transfers in manual mode.

Enable

Software selects the manual transfer method by setting the SPI.CONFIG [Man_start_en] bit = 1. In this mode, software must explicitly start the data transfer using the manual start command mechanism. When the [Man_start_en] bit = 0, the controller hardware automatically starts the data transfer when there is data available in the TXFIFO.

Command

Software starts a manual transfer by writing a 1 to the [Man_start_com] bit. When the software writes the 1, the controller hardware starts the data transfer and transfers all the data bytes present in the TXFIFO. The [Man_start_com] bit is self-clearing. Writing a 1 to this bit is ignored if [Man_start_en] = 0. Writing a 0 to [Man_start_com] has no effect, regardless of mode.

Clocking

The slave select input pin must be driven synchronously with respect to the SCLK input. The controller operates in the SPI_REF_CLK clock domain. The input signals are synchronized and analyzed in the SPI_REF_CLK domain.

Word Detection

The start of a word is detected in the SPI_REF_CLK clock domain.

- *Detection when controller is enabled*: if the controller is enabled (from a disabled state) at a time when the slave select is active-Low, the controller ignores the data and waits for the SCLK to be inactive (a word boundary) before capturing data. The controller counts SCLK inactivity in the SPI_REF_CLK domain. A new word is assumed when the SCLK idle count reaches the value programmed into the SPI.SLV_IDLE [Slave_Idle_count] bit field.
- *Detection when slave select is asserted*: with the controller enabled and slave select is detected as High (inactive), the controller assumes the start of the word occurs on the next active edge of SCLK after slave select transitions active-Low.

Start Condition

The start condition must be held active for at least four SPI_REF_CLK cycles to be detected. If slave mode is enabled at a time when the master is very close to starting a data transfer, there is a small probability that false synchronization will occur, causing packet corruption. This issue is avoided by ensuring any of the following:

- External master does not initiate a data transfer until at least ten SPI_REF_CLK cycles are complete after slave mode is enabled,
- Slave mode is enabled before the attached master is enabled, or

- Slave select input signal is not active when the slave is enabled.

Register Reference

SPI controller and system-level registers:

- Controller Registers
- System Level Registers

Controller Registers

The SPI register set is summarized in the following table.

Table 167: **SPI Controller Register Set Overview**

Register Name	Offset Address	Access	Description
SPI.CONFIG	0x000	R/W	Controller configuration.
SPI.ISR SPI.IER SPI.IDR SPI.IMR	0x004 0x008 0x00C 0x010	WTC W W W	Interrupts; status is after the mask.
SPI.ENABLE	0x014	R/W	Controller enable.
SPI.DELAY	0x018	R/W	Master mode only. Extends bus timing related to the SPIx_CS_b output.
SPI.TXD SPI.RXD	0x01C 0x020	W R	TX and RX FIFO data ports.
SPI.SLV_IDLE	0x024	R/W	Slave mode only. SCLK idle count for hardware to measure to detect a transaction start.
SPI.TX_THRESH SPI.RX_THRESH	0x028 0x02C	R/W R/W	Defines the fill level for TX FIFO not full and RX FIFO not empty.

System Level Registers

The system level registers include the CRL clock and reset, and I/O routing SLCR registers.

- CRL.SPI0_REF_CTRL (reference clock)
- CRL.SPI1_REF_CTRL (reference clock)
- CRL.RST_SPI0 (reset)
- CRL.RST_SPI1 (reset)

I/O Interface

The SPI controller I/O interface is routed to both the PMC and LPD MIOs, and the EMIO. The MIO signals are shown in the MIO-at-a-Glance Tables and all signals are detailed in the following table. The I/O signals are shown in the Functional Diagram.

Table 168: **SPI Controller I/O Signals**

MIO					EMIO	
Signal Name	**I/O**	**PMC MIO Pin**	**LPD MIO Pin**	**MIO-at-a-Glance Table**	**Signal Name**	**I/O**
Master and Slave Signals						
SPI0_SCLK SPI1_SCLK	I/O	MIO-at-a-Glance Tables		5		
SPI0_MISO SPI1_MISO	I/O			1		
SPI0_MOSI SPI1_MOSI	I/O			0		
SPI0_SS0_b SPI1_SS0_b	I/O			2		
Master-only Signals						
SPI0_SS1_b SPI1_SS1_b	O	MIO-at-a-Glance Tables		3		
SPI0_SS2_b SPI1_SS2_b	O			4		

UART SBSA Controller

The UART controller is a full-duplex asynchronous receiver and transmitter that supports a wide range of programmable baud rates. The server-based system applications (SBSA) functionality is defined by the Arm® architecture.

There are two UART controllers, and they are located in the LPD IOP.

The UART performs the following:

- Serial-to-parallel conversion on data received from a peripheral device
- Parallel-to-serial conversion on data transmitted to a peripheral device

The software performs reads and writes of data and control/status information via the APB slave interface. The transmit and receive register ports are buffered with internal RX and TX FIFOs with up to 32B of storage.

The UART includes a programmable baud rate generator that generates a common transmit and receive clock from the UARTX_REF_CLK.

The maximum baud rates in different modes are as follows:

- 921600 bps, in UART mode
- 460800 bps, in IrDA mode
- 115200 bps, in low-power IrDA mode

Features

- 32 deep ×8-bit wide transmit FIFO
- 32 deep ×12-bit wide receive FIFO
- Standard asynchronous communication bits (start, stop and parity)
- Independent interrupt masking:
  - Transmit and receive FIFOs
  - Receive timeout, modem status, and error condition

- False start bit detection
- Line break generation and detection
- Modem control functions CTS, DCD, DSR, RTS, DTR, and RI

Programmable Parameters

- Programmable hardware flow control
- Fully-programmable serial interface characteristics
  - 5, 6, 7, or 8-bit data
  - Even, odd, stick, or no-parity bit generation and detection
  - 1 or 2 stop bit generation
  - Baud rate generator; DC up to UARTx_REF_CLK/16
- Communication baud rate, integer, and fractional parts
- FIFO enable (32 deep) or disable (1 deep)
- FIFO trigger levels selectable between 1/8, 1/4, 1/2, 3/4, and 7/8

Modem Operation

The UART can be used to support the data terminal equipment (DTE) and the data communication equipment (DCE) modes of operation.

Comparison to Previous Generation Xilinx Devices

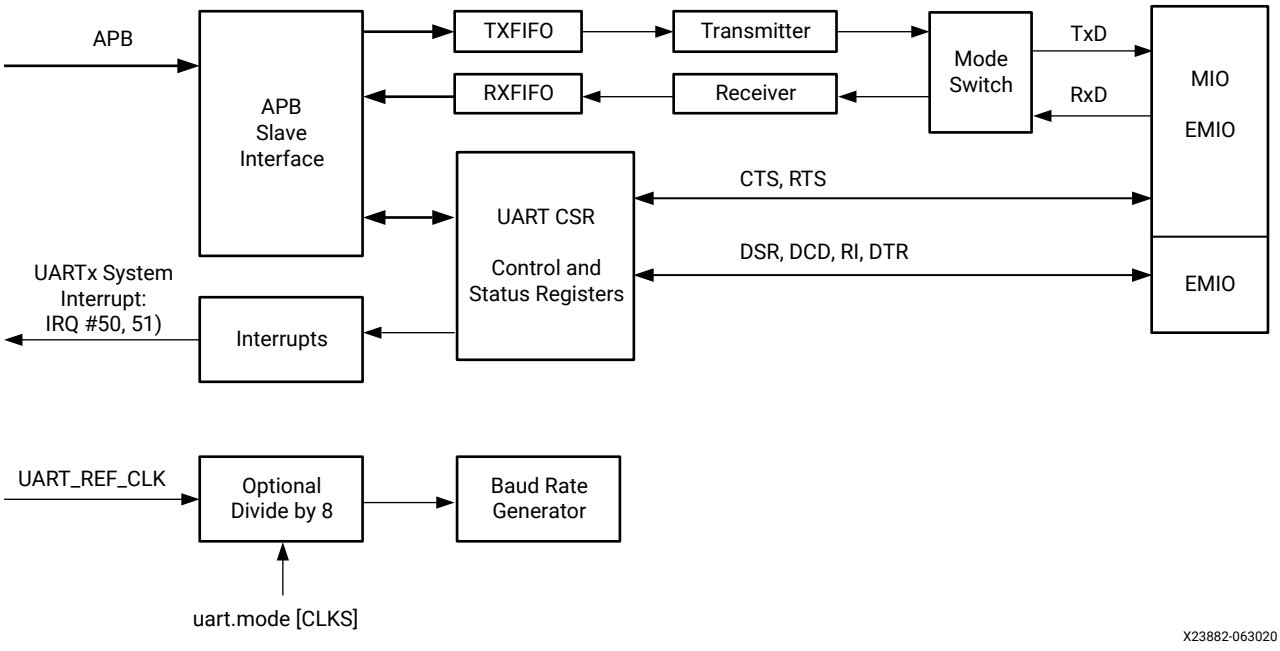
The UART is based on Xilinx IP. It includes enhancements for the server-based system applications, SBSA, defined by the Arm® architecture. Additional changes include the RTS and CTS flow control signals available on the MIO pins and IrDA mode. The UART controller in Zynq® UltraScale+™ MPSoC is based on Cadence IP.

System Perspective

Block Diagram

The high-level block diagram for the UART controller is shown in the following figure.

Figure 85: **UART Controller High-level Block Diagram**



Functional Units

System Interface

The single APB programming interface provides access to the read/write control and status registers, and the transmit and receive FIFOs. An APB interface, connected to the IOP slave switch 32-bit APB bus is used for all controller configuration, control, and data transfer operations.

System Signals

Reference Clock

The controller and I/O interface are driven by the reference clock UARTx_REF_CTRL. The controller's interconnect also requires an APB interface clock, LPD_LSBUS_CLK. Both of these clocks always come from the PS clock subsystem. The UARTx_REF_CLK can be derived from any of the PLLs as described in Clocks.

Controller Reset

A single active-Low reset is used by the controller. The UART controller is reset by configuring the registers CRL.RST_UART0 [RESET] and RST_UART1 [RESET].

Modes and States

UART Mode

The operation and baud rate values are controlled by the line control register, UART.LCR_H, and the baud rate divisor registers

- Integer baud rate register, UART.BAUD_INTEGER
- Fractional baud rate register, UART.BAUD_FRACT

The UART generates individual, maskable interrupts:

- Receiver (including timeout)
- Transmitter
- Modem status
- Error conditions

The interrupts are OR'd together to generate the system interrupt: IRQ #50 for UART0 and IRQ51 for UART1.

When a framing, parity, or break error occurs during reception, the appropriate error bit is set. When an overrun condition occurs, the overrun register bit is set immediately and FIFO data is prevented from being overwritten.

The FIFOs can be programmed to be 1-byte deep providing a conventional double-buffered UART interface.

The modem status input signals Clear To Send (CTS), Data Carrier Detect (DCD), Data Set Ready (DSR), and Ring Indicator (RI) are supported. The output modem control lines, Request To Send (RTS), and Data Terminal Ready (DTR) are also supported.

Hardware flow control feature uses the UARTx_CTS_b input and the UARTx_RTS_b output to automatically control the serial data flow.

IrDA Mode

The serial infrared (SIR) controller contains an IrDA SIR ENDEC. The SIR ENDEC can be enabled for serial communication through nSIROUT and SIRIN.

When the SIR ENDEC is enabled, the UARTx_TXD line is held in the passive state (High logic level) and transitions of the modem status, or the UARTx_RXD line have no effect. The SIR ENDEC can receive and transmit, but it is half-duplex only, so it cannot receive while transmitting, or transmit while receiving.

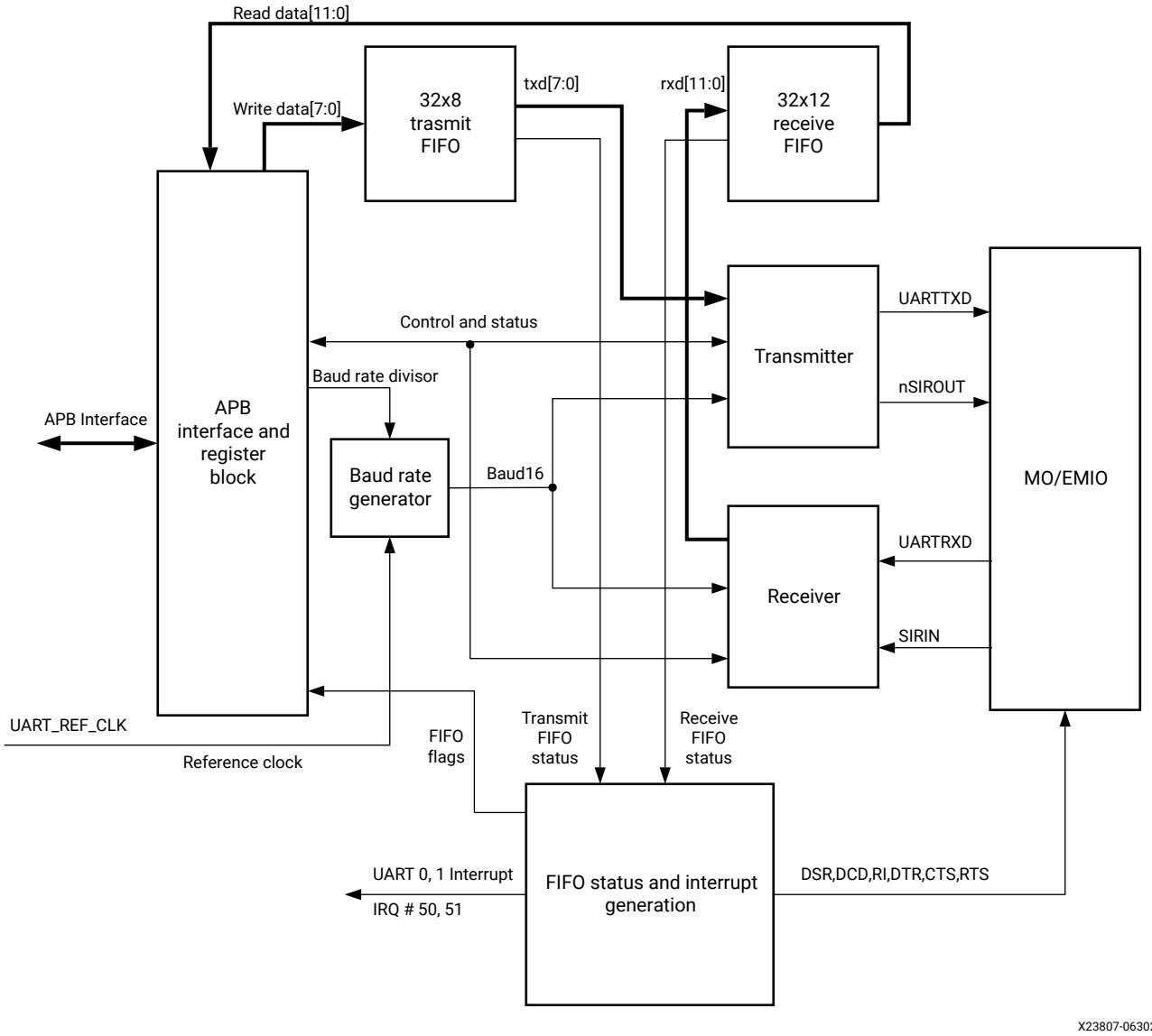
The IrDA SIR physical layer specifies a minimum 10 ms delay between transmission and reception.

UART Functionality

Block Diagram

The UART functional description is illustrated in the following figure.

Figure 86: **UART Functional Block Diagram**



Baud Rate Generator

The baud rate generator contains free-running counters that generate the internal ×16 clocks, Baud16 and IrLPBaud16. Baud16 provides timing information for UART transmit and receive control. Baud16 is a stream of pulses with a width of one UARTx_REF_CLK clock period and a frequency of 16 times the baud rate. IrLPBaud16 provides timing information to generate the pulse width of the IrDA encoded transmit bit stream when in low-power IrDA mode.

Transmit FIFO

The transmit FIFO is an 8-bit wide and 32 location deep FIFO memory buffer. Data written across the APB interface is stored in the FIFO until read out by the transmit logic. The transmit FIFO can be disabled to act like a one-byte holding register.

Receive FIFO

The receive FIFO is a 12-bit wide and 32 location deep FIFO memory buffer. Received data and corresponding error bits are stored in the receive FIFO by the receive logic until read out by the CPU across the APB interface. The receive FIFO can be disabled to act like a one-byte holding register.

Transmit Logic

The transmit logic performs parallel-to-serial conversion on the data read from the transmit FIFO. The control logic outputs the serial bitstream beginning with a start bit, data bits with the Least Significant Bit (LSB) first, followed by the parity bit, and then the stop bits according to the programmed configuration in the control registers.

Receive Logic

The receive logic performs serial-to-parallel conversion on the received bitstream after a valid start pulse has been detected. Overrun, parity, frame error checking, and line break detection are also performed, and their status accompanies the data that is written to the receive FIFO.

Interrupts

Individual maskable active-High interrupts are generated by the UART. A system interrupt output is generated as an OR function of the individual interrupt requests.

Operation

The following descriptions are for one instance of UART and the same applies to other instances.

The control data is written to the UART Line Control register, UART.LCR.

LINE_CTRL defines the:

- Transmission parameters
- Word length
- Buffer mode
- Number of transmitted stop bits
- Parity mode
- Break generation

BAUD_INTEGER defines the integer baud rate divider.

BAUD_FRACT defines the fractional baud rate divider.

Data Transmission and Reception

Data received or transmitted is stored in two 32-byte FIFOs. The receive FIFO has an extra four bits per character for status information.

Transmission

During transmission, data is written into the transmit FIFO. When the UART is enabled, it causes a data frame to start transmitting with the parameters indicated in the Line Control register, UART.LCR_H. Data continues to be transmitted until there is no data left in the transmit FIFO. The BUSY signal goes High as soon as data is written to the transmit FIFO (that is, the FIFO is non-empty) and remains asserted High while data is being transmitted. BUSY is negated only when the transmit FIFO is empty, and the last character has been transmitted from the shift register, including the stop bits.

Reception

When the receiver is idle (UARTRXD continuously 1, in the marking state) and a Low is detected on the data input (a start bit has been received), the receive counter, with the clock enabled by Baud16, begins running and data is sampled on the eighth cycle of that counter in UART mode, or the fourth cycle of the counter in SIR mode to allow for the shorter logic 0 pulses (half way through a bit period).

The start bit is valid if UARTRXD is still Low on the eighth cycle of Baud16, otherwise a false start bit is detected and it is ignored. When a valid start bit is detected, successive data bits are sampled on every 16th cycle of Baud16 (that is, one bit period later according to the programmed length of the data characters. The parity bit is then checked if parity mode was enabled.

A valid stop bit is confirmed if UARTRXD is High, otherwise a framing error has occurred. When a full word is received, the data is stored in the receive FIFO, with any error bits associated with that word.

Error Bits

Three error bits are stored in bits [10:8] of the receive FIFO and are associated with a particular character. There is an additional error that indicates an overrun error and this is stored in bit 11 of the receive FIFO.

Overrun Bit

The overrun bit is not associated with the character in the receive FIFO. The overrun error is set when the FIFO is full, and the next character is completely received in the shift register. The data in the shift register is overwritten, but it is not written into the FIFO. When an empty location is available in the receive FIFO, and another character is received, the state of the overrun bit is copied into the receive FIFO along with the received character. The overrun state is then cleared. The following table lists the bit functions of the receive FIFO.

Table 169: **Receive FIFO Bit Functions**

FIFO bit	Function
11	Overrun indicator
10	Break error
9	Parity error
8	Framing error
7:0	Received data

System and Diagnostic Loopback Testing

Loopback testing is done for data by setting the loopback enable bit to 1 in UART.CTRL [LBE]. Data transmitted on UARTx_TXD output signal is received on the UARTx_RXD input signal of the same controller.

Baud Rate Divider

The baud rate divisor is a 22-bit number consisting of a 16-bit integer and a 6-bit fractional part. This is used by the baud rate generator to determine the bit period. The fractional baud rate divider enables the use of any clock with a frequency >3.6864 MHz to act as UARTx_REF_CLK, while it is still possible to generate all the standard baud rates.

The 16-bit integer is written to the Integer Baud Rate bit field, UARTx.BAUD_INTEGER [DIVINT]. The 6-bit fractional part is written to the Fractional Baud Rate bit field, UARTx.BAUD_FRACT [DIVFRAC]. The baud rate divisor has the following relationship to UARTx_REF_CLK:

Baud Rate Divisor = UARTx_REF_CLK/(16× Baud Rate) = [DIVINT] . [DIVFRAC]

Figure 87: **Baud Rate Divisor**

The 6-bit number (m) can be calculated by taking the fractional part of the required baud rate divisor and multiplying it by 64 (that is, 2^[DIVFRAC]) and adding 0.5 to account for rounding errors:

$$m = \text{integer}\ ([BAUD\_FRACT] \times 2\text{^}n + 0.5)$$

An internal clock enable signal, Baud16, is generated, and is a stream of one UARTx_REF_CLK wide pulses with an average frequency of 16 times the required baud rate. This signal is then divided by 16 to give the transmit clock. A low number in the baud rate divisor gives a short bit period, and a high number in the baud rate divisor gives a long bit period.

Baud Rate Clock

The frequency selected for UARTx_REF_CLK must accommodate the required range of baud rates:

- $F_{UART\_REF\_CLK} \geq 16 \times baud\_rate(max)$
- $F_{UART\_REF\_CLK} \leq 16 \times 65535 \times baud\_rate(max)$

For example, for a range of baud rates from 110 baud to 460800 baud the UARTx_REF_CLK frequency must be between 7.3728 MHz to 115.34 MHz. The frequency of UARTx_REF_CLK must also be within the required error limits for all baud rates to be used. There is also a constraint on the ratio of clock frequencies for LPD_LSBUS_CLK to UARTx_REF_CLK. The frequency of UARTx_REF_CLK must be no more than 5/3 times faster than the frequency of LPD_LSBUS_CLK:

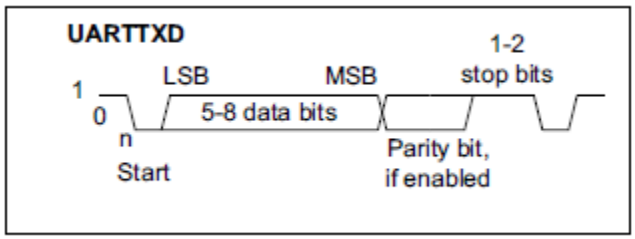
- $F_{UART\_REF\_CLK} \leq \frac{5}{3} \times F_{LPD\_LSBUS\_CLK}$

For example, in UART mode, to generate 921600 baud when UARTx_REF_CLK is 14.7456 MHz, LPD_LSBUS_CLK must be greater than or equal to 8.85276 MHz. This ensures that the UART has sufficient time to write the received data to the receive FIFO.

Character Frame

The following figure shows the character frame.

Figure 88: **UART Character Frame**

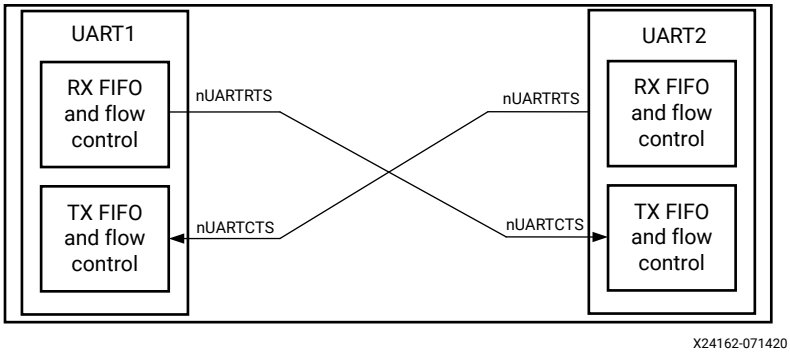


Hardware Flow Control

The hardware flow control feature is fully selectable. This feature enables the control of the serial data flow by using the UARTX_CTS_b output and UARTX_RTS_b input signals.

The following figure shows communication between two devices using the hardware flow control.

Figure 89: **Hardware Flow Control Between Two Similar Devices**



When the RTS flow control is enabled, UARTX_RTS_b is asserted until the receive FIFO is filled up to the programmed watermark level. When the CTS flow control is enabled, the transmitter can only transmit data when UARTX_CTS_b is asserted.

The hardware flow control is selectable using the [RTSEn] and [CTSEn] bits in the Control register, UART.CTRL. The following table lists the bit settings used to enable RTS and CTS flow control both simultaneously and independently.

Table 170: **Control Bits to Enable and Disable Hardware Flow Control**

CTSEn	RTSEn	Description
1	1	Both RTS and CTS flow control enabled
1	0	Only CTS flow control enabled
0	1	Only RTS flow control enabled
0	0	Both RTS and CTS flow control disabled

***Note*:** When RTS flow control is enabled, the software cannot use the [RTSEn] bit in the Control register, UART.CTRL to control the status of UARTX_RTS_b.

RTS Flow Control

The RTS flow control logic is linked to the programmable receive FIFO watermark levels. When RTS flow control is enabled, the UARTx_RTS_b is asserted until the receive FIFO is filled up to the watermark level. When the receive FIFO watermark level is reached, the UARTx_RTS_b signal is deasserted, indicating that there is no more room to receive any more data. The transmission of data is expected to cease after the current character has been transmitted.

The UARTx_RTS_b signal is reasserted when data has been read out of the receive FIFO so that it is filled to less than the watermark level. If RTS flow control is disabled and the UART is still enabled, then data is received until the receive FIFO is full, or no more data is transmitted to it.

CTS Flow Control

When the CTS flow control is enabled, the transmitter checks the UARTx_CTS_b signal before transmitting the next byte. When the UARTx_CTS_b signal is asserted, it transmits the byte; otherwise, the transmission does not occur.

The data continues to be transmitted while UARTx_CTS_b is asserted, and the transmit FIFO is not empty. When the transmit FIFO is empty and the UARTx_CTS_b signal is asserted no data is transmitted.

When the UARTx_CTS_b signal is deasserted and CTS flow control is enabled, the current character transmission is completed before stopping. When the CTS flow control is disabled and the UART is enabled, the data continues to be transmitted until the transmit FIFO is empty.

IrDA Functionality

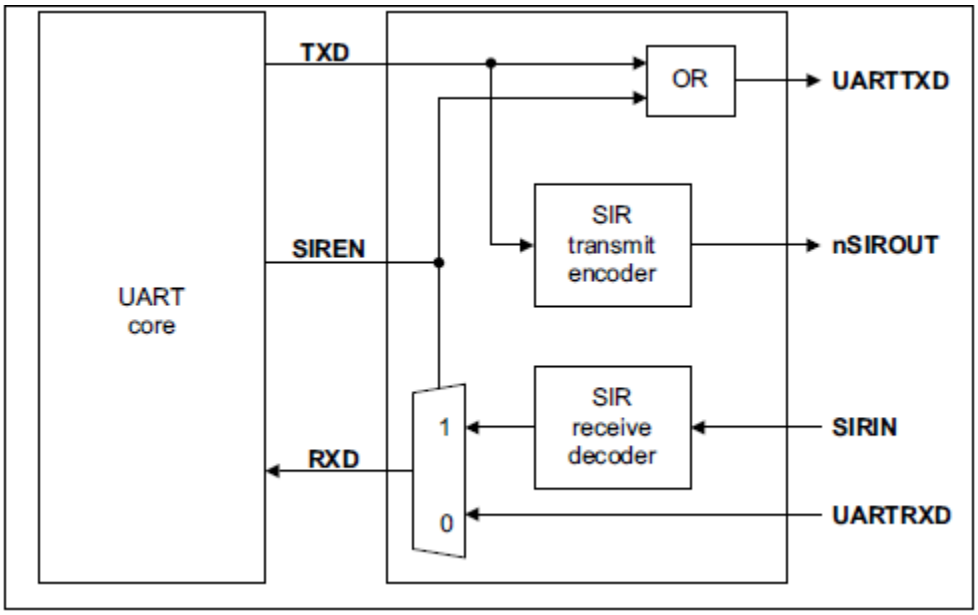
The IrDA SIR ENDEC comprises:

- IrDA SIR transmit encoder
- IrDA SIR receive decoder

Block Diagram

The IrDA SIR ENDEC block diagram is shown in the following figure.

Figure 90: **IrDA SIR ENDEC Block Diagram**



Transmit Encoder

The SIR transmit encoder modulates the non return-to-zero (NRZ) transmit bit stream output from the UART. The IrDA SIR physical layer specifies the use of a return to zero, inverted (RZI) modulation scheme that represents logic 0 as an infrared light pulse. The modulated output pulse stream is transmitted to an external output driver and infrared light emitting diode (LED).

The frequency of IrLPBaud16 is set up by writing the appropriate divisor value to the IrDA Low-Power Counter register, UART.ILPR.

The active-Low encoder output is normally Low for the marking state (no light pulse). The encoder outputs a high pulse to generate an infrared light pulse representing a logic 0 or spacing state.

Receive Decoder

The SIR receive decoder demodulates the return-to-zero bitstream from the infrared detector and outputs the received NRZ serial bitstream to the UART received data input. The decoder input is normally High (marking state) in the idle state. The transmit encoder output has the opposite polarity to the decoder input.

A start bit is detected when the decoder input is Low.

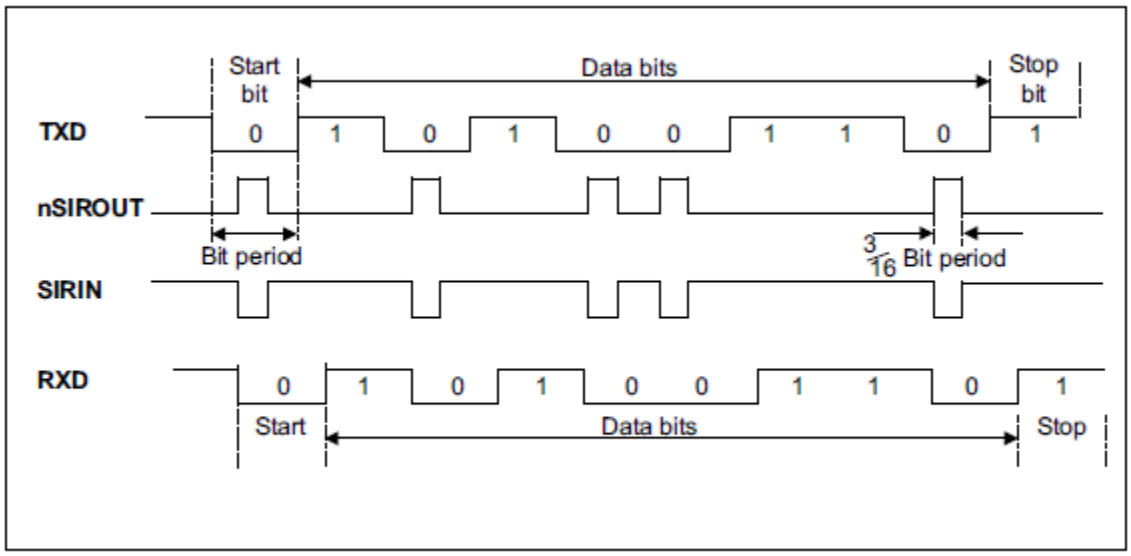
Note: To prevent the UART from responding to glitches on the received data input then it ignores SIRIN pulses that are less than:

- 3/16 of Baud16, in IrDA mode
- 3/16 of IrLPBaud16, in low-power IrDA mode

Data Modulation

The following figure shows the IrDA data modulation.

Figure 91: **UART IrDA 3/16 Data Modulation**



Interrupts

Eleven maskable interrupts are generated in the UART. These are combined to produce one system interrupt that is the OR of the individual outputs. Interrupts can be enabled or disabled individually by changing the mask bits in the interrupt mask set/clear register, UART.IMSC. Setting the appropriate mask bit High enables the interrupt.

Flow Control Interrupts

The modem status interrupt is asserted if any of the modem status signals (UARTx_CTS_b, DCD_b, DSR_b, and RI_b) change. It is cleared by writing a 1 to the corresponding bits in the Interrupt Clear register, UART.ICR, depending on the modem status signals that generated the interrupt.

Change State Interrupt

The transmit interrupt changes state when one of the following events occurs:

- If the FIFOs are enabled and the transmit FIFO is equal to or lower than the programmed trigger level, the transmit interrupt is asserted High. The transmit interrupt is cleared by writing data to the transmit FIFO until it becomes greater than the trigger level, or by clearing the interrupt.
- If the FIFOs are disabled (have a depth of one location) and there is no data present in the transmitters single location, the transmit interrupt is asserted High. It is cleared by performing a single write to the transmit FIFO, or by clearing the interrupt.
- To update the transmit FIFO, write data to the transmit FIFO, either prior to enabling the UART and the interrupts, or after enabling the UART and interrupts.

***Note*:** The transmit interrupt is based on a transition through a level, rather than on the level itself. When the interrupt and the UART is enabled before any data is written to the transmit FIFO the interrupt is not set. The interrupt is only set, after written data leaves the single location of the transmit FIFO and it becomes empty.

Timeout Interrupt

The receive timeout interrupt is asserted when the receive FIFO is not empty and no more data is received during a 32-bit period. The receive timeout interrupt is cleared either when the FIFO becomes empty through reading all the data (or by reading the holding register), or when a 1 is written to the corresponding bit of the Interrupt Clear register, UART.ICR.

Error Interrupt

The error interrupt is asserted when an error occurs in the reception of data by the UART. The interrupt can be caused by a number of different error conditions:

- Framing
- Parity
- Break
- Overrun

The cause of the interrupt can be determined by reading the Raw Interrupt Status register, UART.RIS or the Masked Interrupt Status register, UART.MIS. It can be cleared by writing to the relevant bits of the Interrupt Clear register, UART.ICR (bits 7 to 10 are the error clear bits).

Registers

UART Registers

The UART controller core registers are listed in the following table. The base address for each UART register module:

- UART0: 0xFF00_0000
- UART1: 0xFF01_0000

Table 171: **UART Controller Registers**

Register Name	Offset Address	Type	Description
Data Ports			
DATA	0x000	RW	Read/write data port
Miscellaneous Control			
CTRL	0x030	RW	Configuration and control
BAUD_INTEGER BAUD_FRACT	0x024 0x028	RW	BAUD rate integer and fractional divider
LINE_CTRL	0x02C	RW	Line control
IR_LOWPWR	0x020	RW	Low power counter divisor
Status/Clear and Flags			
ERR_STAT_CLR FLAG	0x004 0x018	RW R	Interface flags
Interrupts			
INTR_MASK INTR_RIS INTR_MIS INTR_CLR	0x038 0x03C 0x040 0x044	RW R R W	Read/write interrupt mask Raw interrupt status Masked interrupt status Clear interrupt status
FIFO Interrupt Levels			
FIFO_LEVEL	0x034	RW	RX and TX FIFO interrupt trigger levels

SLCR Registers

The UART system-level control registers (SLCR) are listed in the following table. The base address for the SLCR registers:

- LPD_IOP_SLCR register module is 0xFF08_0000.
- PMC_IOP_SLCR register module is 0xF106_0000 (for PMC MIO).

Table 172: **UART System-Level Clock and Reset Registers**

Register Name	Bit Field	Offset Address	Access Type	Description
LPD_IOP_SLCR APB Programming Interface Access Error Interrupt				
PARITY_ISR PARITY_IMR PARITY_IER PARITY_IDR	[perr_uart0_apb] [perr_uart1_apb]	0x0714+	W1C, R R W W	Parity error detected on APB programming interface write data
LPD_IOP_SLCR MIO Select				
LPD_MIO_SEL	[UART0_SEL] [UART1_SEL]	0x0410	RW	Select between PMC and LPD MIO muxes
LPD_IOP_SLCR MIO Pin Routing				
MIO_PIN_0 etc MIO_PIN_25	[L0_SEL} [L1_SEL] [L2_SEL] [L3_SEL]	0x000+	RW	LPD mux MIO routing
PMC_IOP_SLCR MIO Pin Routing				
MIO_PIN_0 etc MIO_PIN_51	[L0_SEL} [L1_SEL] [L2_SEL] [L3_Sel]	0x000+	RW	PMC mux MIO routing

Clock and Reset Registers

The UART reference clock and core reset are controlled by the CRL register module. The base address for the CRL register module is 0xFF5E_0000.

Table 173: **UART Clock and Reset Registers**

Register Name	Offset Address	Access Type	Description
Reference Clock			
UART0_REF_CTRL UART1_REF_CTRL	0x0128 0x012C	RW	Reference clock control from LPD clock controller
Controller Reset			
RST_UART0 RST_UART1	0x0318 0x031C	RW	Controller reset from LPD reset controller

UART I/O Signals

The UART controller I/O signals are routed to both the PMC and LPD MIOs, and the EMIO. Each set of I/O signals can be located on one of six sections of pins as shown in MIO-at-a-Glance Tables. The CTS_b and RTS_b signals are available on the MIO or EMIO. The UART flow control signals are available on EMIO.

Table 174: **UART Controller I/O Signals**

MIO					EMIO	
Signal Name	**I/O**	**PMC MIO Pin**	**LPD MIO Pin**	**MIO-at-a-Glance Table**	**Signal Name**	**I/O**
UART0_RXD UART1_RXD	I	MIO-at-a-Glance Tables[1]		0		I
UART0_TXD UART1_TXD	O			1		O
UART0_CTS_b UART1_CTS_b	I			2	CTS_b	I
UART0_RTS_b UART1_RTS_b	O			3	RTS_b	O
Not available on MIO					DCD_b	O
					DSR_b	I
					RI_b	I
					DTR_b	I
					SIRIN	O
					SIROUT_b	I

Notes:

1. The RXD, TXD, CTS_b, and RTS_b signals are routed to the MIO as a group. The groups are shown in the MIO-at-a-Glance Tables. Unused signals (e.g., CTS_b and RTS_b) do not need to be routed through the MIO.

USB 2.0 Controller

The controller is compliant with the USB 2.0 specification to support high, full, and low-speed modes in all configurations. It can be configured as a host, a device, or in on-the-go (OTG). Both host and device modes can operate at the same time as a dual role device (DRD).

In host mode, the controller is compatible with the Intel extensible host controller interface (xHCI) specification. In device mode, it supports up to 12 endpoints (6 in and 6 out). The controller's I/O uses an 8-bit universal low peripheral interface (ULPI) to connect the Versal™ device to an external PHY via the PMC MIO pins. The controller operates at 20 MHz using the USB_REF_CLK from the LPD clock controller. The ULPI interface is clocked at 60 MHz by the PHY. The controller provides transfer rates up to 480 Mb/s for high-speed mode. The 32-bit AXI slave programming interface is accessed by system software to control the modes. The programming interface provides access to the USB_2_REGS controller and USB_2_XHCI core register sets. The AXI slave is attached to the LPD IOP slave switch.

The 64-bit AXI master interface is used by the DMA unit to read descriptor tables and access data buffers. The AXI master is attached to the LPD IOP master switch. The controller includes a single dual-port RAM to store RX FIFO data, TX FIFO data, and to cache descriptors. The AXI master port and the protocol layers access the RAMs using the buffer management unit. The RAM provides buffering of transaction data between the ULPI interface and system memory. The host controller is a schedule driven environment for data transfers of periodic (interrupt and isochronous) and asynchronous (control and bulk) types.

Device mode includes a simple pair of descriptors to respond to USB data transfers in a timely manner between the software and the USB. The transfer descriptors of the host schedules and device endpoints control the DMA engine to move data between the 64-bit AXI master system bus interface and the RX and TX data FIFOs in RAM that respond in real time to the USB. The controller makes strategic use of software for tasks that do not require time-critical responses. This reduces the amount of hardware logic. At the same time, the controller includes hardware assistance logic to enable the controller to respond quickly to USB events and simplify the software.

The controller includes the hibernation and low-power modes.

Features

The USB controller includes these features:

- Host, device, and dual-role device options
- Power management features: hibernation mode
- DMA master with 44/48-bit addressing and 64-bit data
- 12 endpoints (six out and six in)
- Compatible with xHCI standard 1.1
- 32-bit AXI programming interface
- ULPI interface routed to PMC MIO pins

Comparison to Previous Generation Xilinx Devices

The USB controller in the Versal™ ACAP is based on newer Synopsys IP and is implemented as USB 2.0. The Zynq UltraScale+ MPSoC included a USB 3.0 controller based on older Synopsys IP.

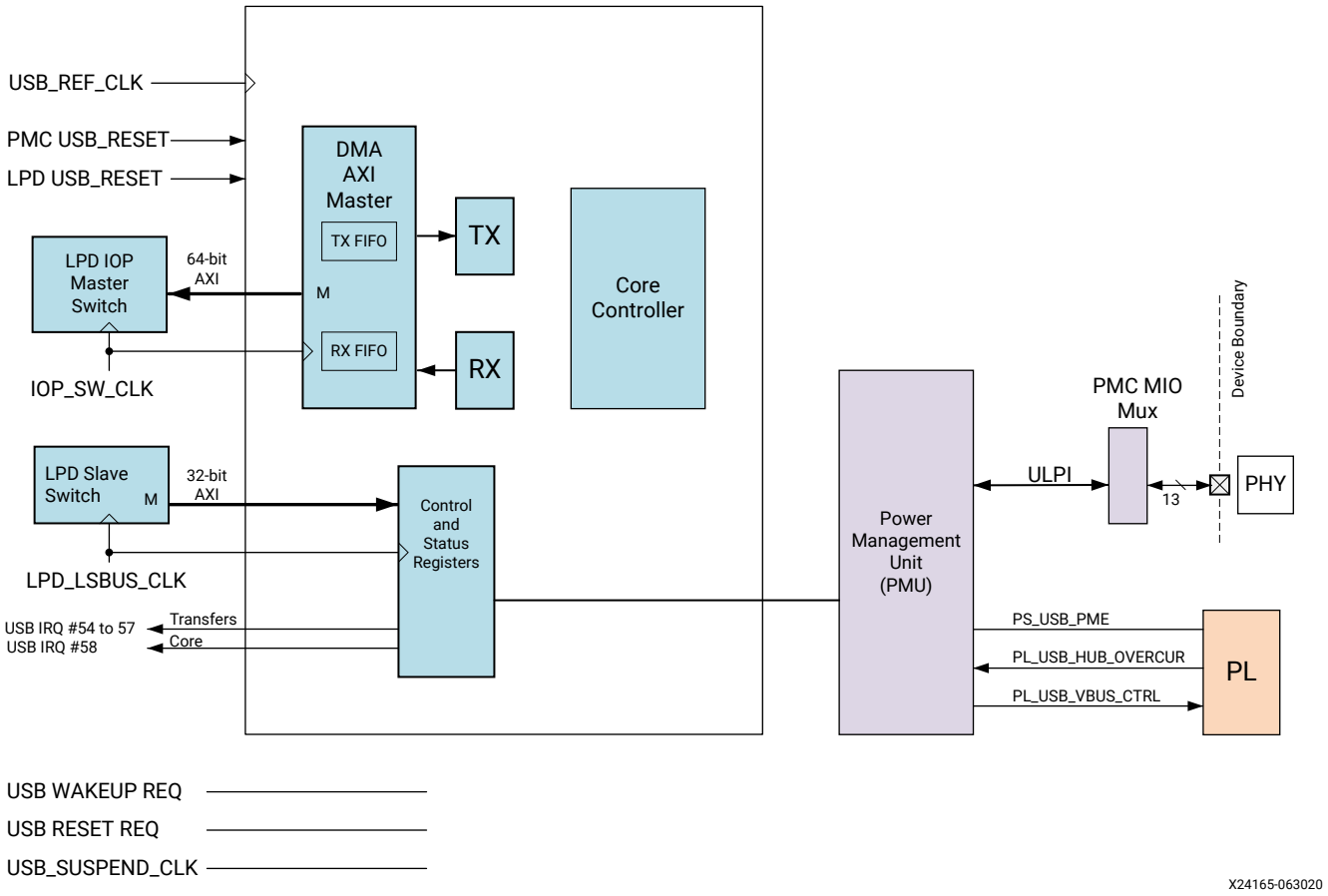
In device mode, the Zynq UltraScale+ MPSoC USB 2.0 controller can be used as the primary boot mode. In Versal™ ACAP, primary boot mode is not supported. Software can configure the USB controller as a secondary boot device.

System Perspective

High-Level Block Diagram

The USB controller block diagram is shown in the following figure.

Figure 92: **USB Controller Block Diagram**



System Interfaces

- AXI slave programming interface
- AXI DMA master interface for descriptor read and data read/write accesses

AXI Slave Programming Interface

The USB register module is accessed by software using the 32-bit AXI slave programming interface attached to the LPD IOP slave switch.

AXI DMA Master

The DMA AXI transaction includes several attributes for coherency and QoS. These are controlled by the following registers:

- LPD_IOP_SLCR.USB_INT_ROUTE for coherency
- LPD_IOP_SLCR.USB_INT_QOS for quality of service (QoS)

- LPD_IOP_SLCR.USB_INT_SMID for system master ID (SMID)

When the DMA transaction is routed to the CCI, it first passes through the TBU0 of the SMMU. This translation unit is also used by the SMMU TCU for translation table lookups.

***Note*:** This datapath will impose a bandwidth load to the CCI and FPD interconnect. In very high traffic loads through TBU0, a deadlock situation can occur. To ensure this does not happen, do not route DMA transactions through the CCI.

System Signals

System signals include:

- Clocks
- Controller Resets
- System Interrupts
- System Error Signal

Clocks

The USB controller includes several clocks from the system.

- USB_REF_CLK from LPD clock controller
- LPD_IOU_REF_CLK for the interconnect interfaces

The USB_REF_CLK is used by the controller and the ULPI interface.

Controller Resets

The controller has three reset domains.

- Controller wrapper
- Controller core
- External ULPI PHY

Reset Matrix

The controller receives one reset input from the SoC and several local register-controlled resets. These are summarized in the following table.

Table 175: **USB Reset Matrix**

Description	Register Control	Controller Wrapper	Controller Core	External ULPI PHY[1]
		USB_RESET	USB_CORE_RST	USB_ULPI_RST
Controller reset	CRL.RST_USB [RESET]	Yes	Yes	Yes
Core soft reset	USB_XHCI.GCTL [CORESOFTRESET]	Yes	Yes	Yes
Internal logic	USB_XHCI.USB_CMD [HCRST]	Yes	Yes	Yes
Core soft reset	USB_XHCI.DCTL [CSFTRST]	Yes	Yes	Yes
Light host reset	USB_XHCI.USB_CMD [LHCRST]	~	Yes	~

Notes:
1. The USB_ULPI_RST signal can be masked using the USB_2_CSR.PHY_RESET_MASK register.

System Interrupts

The USB controller includes four transfer system interrupts and one core system interrupt:

- IRQ# 54 to 57 for transfers (USB_2_XHCI)
- IRQ#58 for OTG (USB_2_CSR)
- IRQ# 106 for PME

These are listed with the other system interrupts in the IRQ System Interrupts table.

XCHI Interrupts

Four transfer interrupts:

- Bulk transfer
- Isochronous transfer
- Controller interrupt
- Control transfer

Control and Status Registers Interrupt

The interrupt is sourced from several register controls:

- USB_2_CSR.ISR register:
  - [addr_dec_err] for APB programming interface address decode error
  - [host_sys_err] for host system error
- USB_2_XHCI.GSTS register:
  - [CURMOD], [BUSERRADDRVLD]

- [CRSTimeout]
- [Device], [Host], [ADP], [BC], [DRD], [SSIC]
- [CBELT]

Power Management Interrupt

The USB power management interrupt (PME) is generated by power management unit in the PMC power domain. This system interrupt is assigned to IRQ# 106.

System Error Signal

The APB interface includes an address decode error detector. If an error is detected a system error is generated and the APB programming interface can optionally assert the SLVERR error signal back to the source and assert the address decode error interrupt.

I/O Interface

The I/O interface implements ULPI. These signals are routed to the PMC MIO. The signals are listed in USB I/O Signals.

ULPI PHY

The controller interfaces to the external ULPI PHY via 12 MIO pins: 8 data I/Os, direction input, control input, clock input, and a stop output.

The PHY is external to the device and is reset by the USB_ULPI_RST output signal. The reset output signal can be masked using the USB_2_CSR.PHY_RESET_MASK register.

GPIO

A LPD GPIO signal can be used to reset the external PHY.

Port Indicator and Power Signals via PL EMIO

The USB port indicator outputs, power select output, and power fault input signals are normally routed through the EMIO to the PL SelectIO pins that connect to the external board logic.

Power

The USB2 controller resides in the LPD.

The power management control and ULPI interface reside in the PMC.

Programming Model

The controller has two sets of registers.

- USB_2_CSR control and status
- USB_2_XHCI

The controller uses system memory for transaction and descriptor data structures. The system memory usage includes:

- TX data buffers
- RX data buffers
- Descriptor tables

Programming Interface

The controller includes a 32-bit AXI slave interface to program the USB_2_CSR and USB_2_XHCI register sets.

Host Mode Data Structures

To operate the USB controller, a set of data structures are defined by the xHCI specification. The application software gives information to the xHCI driver that takes care of the programming and interaction with the data structures. The data structures are used to communicate control, status, and data between the xHCI stack (software) and the USB controller.

Context Data Structures

The USB context data structures are described in the following table. The PAGESIZE is 4 KB.

Table 176: **USB 2.0 Context Data Structures**

Context Data Structure	Maximum Size (bytes)	Boundary	Byte Alignment	Description
Device context	2048	PAGESIZE	64	Slot context and endpoint contexts (up to 32). An array of device contexts is prepared and maintained by the xHCI embedded RAM and software. This array contains a maximum of 256 device contexts. The first entry (slot ID = 0) in the device context base address array is used by the xHCI scratchpad mechanism.
Slot context	64	PAGESIZE	32	Information that applies to a device as a whole. The slot context data structure of a device context is also referred to as an output slot context.

Table 176: **USB 2.0 Context Data Structures** *(cont'd)*

Context Data Structure	Maximum Size (bytes)	Boundary	Byte Alignment	Description
Endpoint context	64	PAGESIZE	32	Information that applies to a specific endpoint
Stream context	16	PAGESIZE	16	Specific stream associated with an endpoint.
Input context	132	PAGESIZE	64	Endpoints and the operations to be performed on those endpoints by the address device, configure endpoint, and evaluate context commands.
Input control context	64	PAGESIZE	64	Device context data structures are affected by a command and the operations to be performed on those contexts
Port bandwidth context	#ports * 4	PAGESIZE	32	Provides software with the percentage of periodic bandwidth available on each root hub port, at the speed indicated by the device speed field of the get port bandwidth command. Software allocates the context data structure and the xHCI updates the context data structure during the execution of a get port bandwidth command.

Ring Data Structures

The USB ring data structures are defined in the following table.

Table 177: **Ring Data Structures**

Context Data Structure	Maximum Size (KB)	Boundary (KB)	Byte Alignment	Description
Transfer ring segments	64	64	16	A transfer request block (TRB) ring is an array of TRB structures, that are used by the xHCI as a circular queue to communicate with the host. Transfer rings provide data transport to and from USB devices. There is a 1:1 mapping between transfer rings and USB pipes. They are defined by an endpoint context data structure contained in a device context, or the stream context array pointed to by the endpoint context.
Command ring segments	64	64	64	The command ring provides system software the ability to issue commands to enumerate USB devices, configure the xHCI to support those devices, and coordinate virtualization features. The command ring is managed by the command ring control register that resides in the operational registers.
Event ring segments	64	64	64	The event ring provides the xHCI with a means of reporting to system software: data transfer and command completion status, root hub port status changes, and other xHCI related events. An event ring is defined by the event ring segment table base address, segment table size, and dequeue pointer registers which reside in the run time registers.
Event ring segment table	512	None	64	Table of event ring segments.

Table 177: **Ring Data Structures** *(cont'd)*

Context Data Structure	Maximum Size (KB)	Boundary (KB)	Byte Alignment	Description
Scratchpad buffers	PAGESIZE	PAGESIZE	PAGESIZE	A scratchpad buffer is allocated from system memory for storing internal state.

Register Reference

The register sets that affect the USB controller include:

- USB_2_CSR register set
- USB_2_XHCI register set
- System-level registers
  - CRL (reference clocks and resets)
  - CRP
  - LPD_IOP_SLCR (system-level controls; AXI transaction attributes and routing)
  - LPD_IOP_SECURE_SLCR
  - LPD_IOP_SLCR
  - Miscellaneous system interrupt, QoS registers
- USB_2_XHCI registers

Controller Registers

The USB_2_CSR registers provide general control and status, transaction controls, and manages APB and host system error interrupts. The registers are located at base address `0xFF9D_0000` and are summarized in the following table.

Table 178: **USB 2.0 Control and Status Registers**

Register Name	Offset Address	Access Type	Description
USB2_PHY_RESET	0x01C	RW	PHY reset output mask
PORT	0x034	RW	Device characteristics
JITTER_ADJUST	0x038	RW	High-speed jitter adjustment
BIG_ENDIAN	0x040	RW	Set = 0; little endian
COHERENCY	0x044	RW	DMA transaction via CCI
XHC_BME	0x048	RW	
APB_ERR_CTRL	0x060	RW	APB slave error enable

Table 178: **USB 2.0 Control and Status Registers** *(cont'd)*

Register Name	Offset Address	Access Type	Description
ISR IMR IER IDR	0x064 0x068 0x06C 0x070	W1C R W W	APB address decode and host system error interrupts

XHCI Registers

The USB_2_XHCI registers provide functionality for the xHCI specification. The base address for these registers is 0xFE20_0000 and they are summarized in the following table.

Table 179: **USB_2_XHCI Register Address Map**

Description	Offset Address Range		Detailed Register Table
	Start	End	
xHCI capabilties, offsets, operations	0x0_0000	0x0_0058	
Ports, run time, host interrupter, event ring, doorbells	0x0_0420	0x0_05E0	
Miscellaneous control, status, capabilities	0x0_08E0	0x0_09C0	
Miscellaneous configuration, control, and user	0x0_C100	0x0_C19C	
ULPI PHY	0x0_C200	0x0_C280	
FIFOs	0x0_C300	0x0_C388	
Event buffer	0x0_C400	0x0_C43C	
DMA	0x0_C600	0x0_C630	
Device CSRs	0x0_C700	0x0_C720	
Device endpoints	0x0_C800	0x0_C8BC	
Device interrupt moderation	0x0_CA00	0x0_CA0C	

Host Capabilities, Offset, and Operations Registers

The USB host capabilities, offset, and operations registers are located in the USB_2_XHCI register set at base address 0xFE20_0000. They are summarized in the following table.

- CONFIG register

  This register is in the AUX power well. It is only reset by the platform during a cold reset or in response to a host controller reset (HCRST).

Table 180: **USB Host Capabilities, Offsets, and Operations Registers**

Register Name	Offset Address	Access Type	Description
CAPLENGTH	0x0000	R	Length capability

Table 180: **USB Host Capabilities, Offsets, and Operations Registers** *(cont'd)*

Register Name	Offset Address	Access Type	Description
HCSPARAMS1 HCSPARAMS2 HCSPARAMS3	0x0004 0x0008 0x000C	R	Host controller structural parameters
HCCPARAMS1 HCCPARAMS2	0x0010 0x001C	R	Host controller capability parameters
DBOFF	0x0014	R	Doorbell offset
RTSOFF	0x0018	R	Run time offset
USBCMD	0x0020	RW	USB command
USBSTS	0x0024	R, W1C	USB status
PAGESIZE	0x0028	R	Page size
DNCTRL	0x0034	RW	Device notification
CRCR_LO CRCR_HI	0x0038 0x003C	RW, R	
DCBAAP_LO DCBAAP_HI	0x0050 0x0054	RW	Device context BAAP
CONFIG	0x0058	RW	Configure

Port Status, Control, Host Interrupter, Event Ring, and Doorbell Registers

The USB port status, control, host interrupter, event ring, doorbell registers are located in the USB_2_XHCI register set at base address 0xFE20_0000. They are summarized in the following table.

- PORTPMSC_20 register

  This register is in the AUX power well. It is only reset by platform hardware during a cold reset or in response to a host controller reset (HCRST).

- IMOD_0, 1, 2, 3 register

  Software can use this register to pace (or even out) the delivery of interrupts to the host CPU. This register provides an inter-interrupt delay between interrupts asserted by the xHCI, regardless of USB traffic conditions. To independently validate configuration settings, software can use the algorithms recommended by the xHCI specification to convert the inter-interrupt interval value to the common interrupts/sec performance metric.

Table 181: **Port Status, Control, Host Interrupter, Event Ring, Doorbell Registers**

Register Name	Offset Address	Access Type	Description
PORTSC_20	0x00420	RW, R, W1C	Port status and control

Table 181: **Port Status, Control, Host Interrupter, Event Ring, Doorbell Registers** *(cont'd)*

Register Name	Offset Address	Access Type	Description
PORTPMSC_20	0x00424	RW, R	Port power management status and control
PORTHLPMC_20	0x0042C	RW	LPM hardware control
MFINDEX	0x00440	Read	Microframe index
IMAN_0 to 3	0x00460 incr	RW, W1C	Interrupter management
IMOD_0 to 3	0x00464 incr	RW	Interrupter moderation
ERSTSZ_0 to 3	0x00468 incr	RW	Event ring segment table size
ERSTBA_LO_0 to 3 ERSTBA_HI_0 to 3	0x00470 incr 0x00474 incr	RW	
ERDP_LO_0 to 3 ERDP_HI_0 to 3	0x00478 incr 0x0047C incr	RW	
DB0 DB{1 to 63}	0x004E0 incr	RW	Doorbells

Miscellaneous Control, Status, and Capabilities Registers

The USB miscellaneous control, status, and capabilities registers are located in the USB_2_XHCI register set at base address 0xFE20_0000. They are summarized in the following table.

Table 182: **USB Miscellaneous Control, Status, and Capabilities Registers**

Register Name	Offset Address	Access Type	Description
USBLEGSUP	0x08E0	RW, R	
USBLEGCTLSTS	0x08E4	W1C, R	
SUPTPRT2_DW0 SUPTPRT3_DW0	0x08F0 0x0900	R	
SUPTPRT2_DW1 SUPTPRT3_DW1	0x08F4 0x0904	R	
SUPTPRT2_DW2 SUPTPRT3_DW2	0x08F8 0x0908	R	xHCI supported protocol capability
SUPTPRT2_DW3 SUPTPRT3_DW3	0x08FC 0x090C	R	

Miscellaneous Configuration, Control, and User Registers

The USB miscellaneous configuration, control, and user registers are summarized in the following table.

Table 183: **USB 2.0 Miscellaneous Configuration, Control, and User Registers**

Register Name	Offset Address	Access Type	Description
GSBUSCFG0 GSBUSCFG1	0x0_C100 0x0_C104	RW	Bus configuration
GTXTHRCFG GRXTHRCFG	0x0_C108 0x0_C10C	RW	TX and RX threshold control
GCTL	0x0_C110	RW	Common control
GSTS	0x0_C118	R	Status
GUCTL1 GUCTL2	0x0_C11C 0x0_C19C	RW	User controls
GSNPSID	0x_C120	Read only	ID register
GGPIO	0x_C124	Mixed	General purpose I/O
GUID	0x_C128	Read/Write	User ID
GUCTL	0x_C12C	Read/Write	Global user control
GBUSERRADDRLO GBUSERRADDRHI	0x_C130 0x_C134	R	Bus address error
GHWPARAMS0 to GHWPARAMS7	0x_C140 to 0x_C15C	R	Implementation parameters
GDBGFIFOSPACE	0x_C160	RW, R	Queue/FIFO space available
ULPI PHY			
GUSB2PHYCFG	0x_C200	Mixed	ULPI PHY configuration
GUSB2PHYACC_ULPI	0x_C280	RW, R	ULPI PHY vendor control
RX/TX FIFO Depths			
GTXFIFOSIZ0 GTXFIFOSIZ1 GTXFIFOSIZ2	0x_C300 0x_C304 0x_C308	RW	RXFIFO 0, 1, 2 depths
GRXFIFOSIZ0 GRXFIFOSIZ1 GRXFIFOSIZ2	0x_C380 0x_C384 0x_C388	RW	TXFIFO 0, 1, 2 depths
Event			
GEVNTADRLO_0 GEVNTADRLO_1 GEVNTADRLO_2 GEVNTADRLO_3	0x_C400 0x_C410 0x_C420 0x_C430	RW	
GEVNTADRHI_0 GEVNTADRHI_1 GEVNTADRHI_2 GEVNTADRHI_3	0x_C404 0x_C414 0x_C42 0x_C434	RW	
GEVNTSIZ_0 GEVNTSIZ_1 GEVNTSIZ_2 GEVNTSIZ_3	0x_C408 0x_C418 0x_C428 0x_C438	RW	

Table 183: **USB 2.0 Miscellaneous Configuration, Control, and User Registers** *(cont'd)*

Register Name	Offset Address	Access Type	Description
GEVNTCOUNT_0 GEVNTCOUNT_1 GEVNTCOUNT_2 GEVNTCOUNT_3	0x_C40C 0x_C41C 0x_C42C 0x_C43C	RW	
Host Controls			
GHWPARAMS8	0x_C600	R	Implementation parameters
GTXFIFOPRIDEV	0x_C610	RW	Device TXFIFO DMA priority
GTXFIFOPRIHST	0x_C618	RW	Host TXFIFO DMA priority
GRXFIFOPRIHST	0x_C61C	RW	Host RXFIFO DMA priority
GDMAHLRATIO	0x_C624	RW	Host FIFO DMA high-low priority ratio
GFLADJ	0x_C630	RW	Frame length adjustment

Device and Command Registers

The device and command registers are summarized in the following table.

Table 184: **USB 2.0 Device and Command Registers**

Register Name	Offset Address	Access Type	Description
Device Registers			
DCFG	0x0000C700	RW	Device configuration
DCTL	0x0000C704	RW, W	Device control
DEVTEN	0x0000C708	RW, R	Device event enable
DSTS	0x0000C70C	R, WTC	Device status
DGCMDPAR	0x0000C710	RW	Device generic command parameter
DGCMD	0x0000C714	RW, R	Device generic command
DALEPENA	0x0000C720	RW	Device active USB endpoint enable
Command Registers			
DEPCMDPAR2_0 DEPCMDPAR2_{1 to 11}	0x0000C800 0x0000C810 incr	RW	Physical endpoint parameter, reg 2
DEPCMDPAR1_0 DEPCMDPAR1_{1 to 11}	0x0000C804 0x0000C814 incr	RW	Physical endpoint parameter, reg 1
DEPCMDPAR0_0 DEPCMDPAR2_{1 to 11}	0x0000C808 0x0000C818 incr	RW	Physical endpoint parameter, reg 0
DEPCMD_0 DEPCMD_{1 to 11}	0x0000C80C 0x0000C81C incr	RW	Physical endpoint command

System-Level Registers

The USB controller is included two clock and reset register modules (CRL and CRP). The controller is located in the LPD, but the PHY controller interface is in the PMC.

- LPD_IOP_SLCR at 0xFF08_0000.
- PMC_IOP_SLCR at 0xF106_0000

LPD System-Level Registers

The LPD_IOP_SLCR registers associated with the controller are listed in the table

Table 185: **USB LPD_IOP_SLCR Registers**

Register Name	Address	Access Type	Description
USB_INT_ROUTE	0xFF08_0428	RW	Select direct path or CCI path to memory
USB_INT_QOS	0xFF08_042C	RW	Define QoS bit values
USB_INT_SMID	0xFF08_0430	RW	Select one SMID bit [0] value

PMC System-Level Registers

The PMC_IOP_SLCR registers associated with the controller are listed in the table

Table 186: **USB PMC_IOP_SLCR Registers**

Register Name	Address	Access Type	Description
USB_PWR_STATE	0xF106_0600	R	Power state of the core

Clock and Reset Registers

The USB reference clock and core reset are controlled by the CRL register module. The base address for the CRL register module is 0xFF5E_0000.

Table 187: **USB Clock and Reset Registers**

Register Name	Offset Address	Access Type	Description
Reference Clock			
USB_BUS_REF_CTRL	0x0124	RW	Reference clock control from LPD clock controller
Controller Reset			
RST_USB	0x0314	RW	Controller reset from LPD reset controller

USB I/O Signals

ULPI I/O Signals

The USB 2.0 controller is attached to an external PHY via the PMC MIO. The MIO interface signals are shown with all other I/O's in MIO-at-a-Glance Tables and detailed in the following table.

The USB controller is located in the LPD, but the ULPI I/O signals are routed to the PMC MIO.

Table 188: **USB 2.0 Controller ULPI I/O Interface**

MIO			
Signal Name	**I/O**	**PMC MIO Pin**	**MIO-at-a-Glance Table**
USB_ULPI_RST	Output	13	12
USB_ULPI_DATA[0] USB_ULPI_DATA[1] USB_ULPI_DATA[2] USB_ULPI_DATA[3]	I/O	14 15 16 17	4 5 6 7
USB_ULPI_CLK	Input	18	0
USB_ULPI_DATA[4] USB_ULPI_DATA[5] USB_ULPI_DATA[6] USB_ULPI_DATA[7]	I/O	19 20 21 22	8 9 10 11
USB_ULPI_DIR	Input	23	1
USB_ULPI_STP	Output	24	2
USB_ULPI_NXT	Input	25	3

Port Indicator, Fault, and Power Select Signals

The following table lists the USB port indicator and power signals on the EMIO.

Table 189: **USB Port Indicator and Power Signals on EMIO**

Port Signals	EMIO Signals		Default Input Value to Controller
	Name	I/O	
Port indicator	EMIOUSB{0,1}PORTINDCTL{0,1}	O	~
Power fault	EMIOUSB{0,1}VBUSPWRFAULT	I	0
Power select	EMIOUSB{0,1}VBUSPWRSELECT	O	~

Flash Memory Controllers

This section includes these chapters:

- Octal SPI Controller
- Quad SPI Controller
- SD/eMMC Controller

The three flash memory controllers are located in the PMC. Their I/O signals are routed to device pins via the PMC MIO multiplexer. Only the SD/eMMC controller I/O signals can be routed to the PL EMIO, but this route requires the LPD to be powered-up.

Each of the flash memories can be a primary boot device on PMC MIO as described in Boot Modes.

OSPI and QSPI Restriction

The OSPI and QSPI are mutually exclusive; only one of the controllers can be used in a system. The selection is done using PMC_IOP_SLCR registers. Program the PMC_IOP_SLCR.MIO_PIN_[0:12] registers to defined the I/O pin connections for these controllers.

Octal SPI Controller

The octal SPI (OSPI) controller can access one or two flash devices using several different methods. The controller is located with the other flash memory controllers in the PMC. The I/O interface is routed to the PMC MIO pin bank 0. OSPI is commonly used as a boot device; see Octal SPI Boot Mode. The controller provides multiple ways to read and write the flash memory:

- STIG/PIO read/write (software triggered instruction generator)
- Direct read/write with address remap
- Non-DMA indirect read/write via AXI slave interface
- DMA indirect read using AXI master interface

STIG/PIO access enables software to read and write 64-bit flash memory data via the APB programming interface.

Direct access allows software to read/write flash memory within a 512 MB memory block starting at `0xC000_0000`. This window is mapped to the flash device memory space. This enables software to perform normal reads and writes within this memory-mapped window. Processor software cannot execute code directly from the controller; execute-in-place is not supported.

In DMA mode, data is autonomously read from the flash memory and written to system memory via the TXFIFO. The DMA master is on the PMC main AXI switch. The DMA includes a 128-word TXFIFO.

Software sends commands to the controller using the flash command register. Commands include configuration, SPI commands (opcode, address, mode, dummy), and single byte reads and writes.

The controller also includes a programmable polling features to read the flash device status and report when a certain value is received. The polling feature also includes an expiration timeout.

The interface works with up to two flash devices that are connected to the PMC MIO mux and pin bank 0. The signals are listed in OSPI I/O Interface. The I/O signals are not available on the LPD MIO pins or as PL EMIO port signals.

Software accesses the OSPI register module via the 32-bit APB programming interface. All of the OSPI related registers are listed in the Register Reference.

Features

The OSPI features include:

- Interface to one or two devices
- DLL for accurate I/O clocking at high speed
- DDR and SDR I/O

Boot Device

The OSPI controller can be used as a boot device. For more information, see Octal SPI Boot Mode.

Nomenclature

The OSPI controller uses slightly different nomenclature for some terms as noted in the following table.

Table 190: **OSPI Nomenclature**

Functionality	Common Nomenclature	OSPI Chapter Nomenclature
Processor addressable flash memory	Linear access mode	Direct mode
Data clocking on positive and negative clock edges	Double data rate (DDR)	Double data rate (DDR) Dual transfer rate (DTR) [1]

Notes:

1. Both the DDR and DTR terms are used in the OSPI chapter. The DTR term is used when referring to the Micron flash memory devices.

Comparison to Previous Generation Xilinx Devices

The OSPI controller is new to Xilinx® devices.

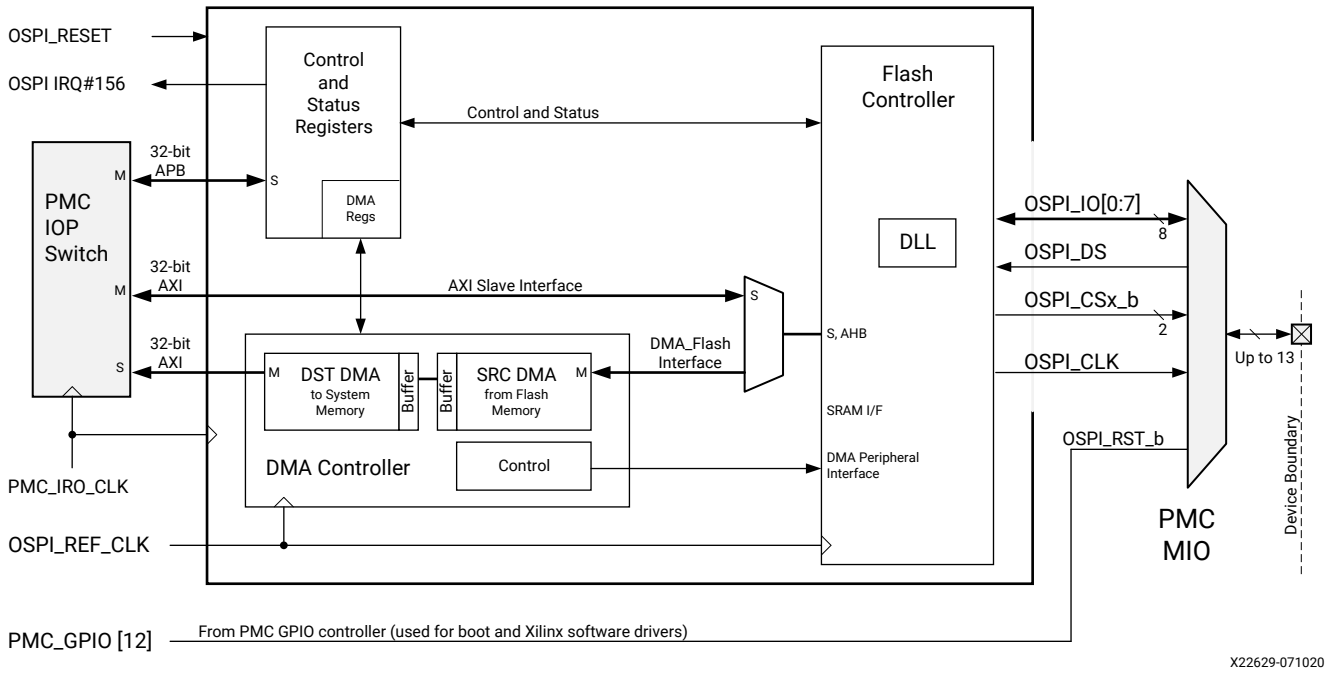
System Perspective

The OSPI is located in the PMC IOP. The controller includes several system interfaces, signals and an I/O interface routed to the PMC MIO pins.

Block Diagram

The high-level block diagram is shown in the following figure. The location of the OSPI is shown in the PMC Interconnect Diagram.

Figure 93: **OSPI High-level Block Diagram**



Functional Units

The main components of the OSPI controller are introduced in this section.

Flash Controller

The flash controller has several modes. Each mode represents a different way for the controller to interface with the flash memory device. See Access Modes.

DMA Controller

The DMA controller is a one-way path to access memory in the flash device and write it to system memory via the 32-bit AXI master interface. The DMA is buffered and supports 64-byte burst transactions on the AXI interface to the PMC IOP switch. See DMA Programming Model for the SRC DMA that fetches data from the flash device and the DST DMA that writes the data to system memory.

Control and Status Registers

The OSPI control and status registers provide access to the DMA and flash controller. In addition to the OSPI registers, there are system-level registers associated with the OSPI controller. See Register Reference for more information.

System Interfaces

The controller has three system interfaces attached to the PMC IOP switch.

AXI Master DMA Interface

The AXI master DMA interface enables the controller to burst data from the flash device to system memory in DMA access mode. For more information, see DMA Programming Model.

AXI Slave Interface

The AXI slave interface enables software to read and write flash memory data. The interface is used in several controller operating modes.

- Direct access mode
- Indirect mode non-DMA

APB Slave Programming Interface

The APB programming interface enables software to access the memory mapped control and status registers. These are listed in Register Reference.

System Signals

System signals include:

- Clocks
- Controller Reset
- System Interrupt
- System Error

Clocks

The reference clock is used by the controller logic and DLL for the I/O clock. The reference clock is generated by the PMC clock controller using the CRP register set.

- OSPI_REF_CLK controlled by the CRP.OSPI_REF_CTRL register

The controller also receives the PMC_IRO_CLK for the system interfaces.

Controller Reset

The controller has two resets. The controller reset, OSPI_RESET, comes from the PMC reset controller. This signal asserts when software writes to the CRP.RST_OSPI register or by any one of a number of system-level resets described in the Resets chapter.

The controller also has a separate PHY reset for the DLL. This is controlled by the OSPI.PHY_CONFIG register.

Asserting Reset

Resets should only be asserted when the controller is inactive, [IDLE] = 1. Doing so at other times produces unknown results.

Reset Condition

The following states are set by the controller reset:

- Transactions on the I/O interface are abruptly terminated (must be avoided, check [IDLE] to confirm the I/O is quiescent)
- Pending requests from the software are canceled
- All registers are set to their reset value

System Interrupt

OSPI has one system interrupt signal, IRQ#156. The system interrupt can be asserted by any one of the three interrupt status registers contained with the OSPI registers set.

- Controller interrupt register:
  - OSPI.IRQ_STATUS register, status is after masking
- DMA source interrupt register:
  - OSPI.DMA_SRC_I_STS, status is before masking
- DMA destination interrupt register:
  - OSPI.DMA_DST_I_STS

The system interrupt is routed to several places as described in System Interrupts.

System Error

The APB programming interface can generate an address decode error if it detects a software access violation. The error signal is routed to the system error accumulator for processing as described in System Errors.

I/O Interface

The I/O signals are routed to the PMC MIO. The 8-bit data bus is supported with two chip selects. The signals are shown in OSPI I/O Interface.

I/O Wiring Diagrams

The I/O wiring connections for boot modes are shown in Octal SPI Boot Mode.

Programming Model

The OSPI has several ways to access flash memory. These modes and their controls are all done using the OSPI register module. The registers are summarized in Register Reference. All modes and controls are done via the OSPI register set. The modes are introduced in Access Modes.

Access Modes

There are several access modes:

- Software triggered instruction generation (STIG read/write)
- Direct read/write mode
- Non-DMA indirect read/write mode
- DMA indirect read mode

Memory Access Modes

There are several flash memory access modes. Only one mode is available at a time. The mode is controlled by the OSPI.xxx register.

- STIG/PIO software triggered instruction generator and programmed I/O
- Direct mode read/write via AXI slave interface
- Software non-DMA indirect read/write via AXI slave interface
- DMA indirect read via AXI master interface

Flash Memory Access Modes Table

Each operating mode uses the system interfaces differently as summarized in the following table. The methods to access the OSPI controls and data differ depending on the operating mode. The features that apply for each mode are identified in the following table.

Table 191: **OSPI Flash Memory Access Modes**

Entity	STIG	Direct Mode	Indirect Mode	
			Non-DMA	With DMA
Interfaces				
AXI system slave	~	Data	Data	~
AXI system master	~	~	~	Data
APB programming	Control and data	Control	Control	Control
Internal DMA peripheral	~	~	Data	
Hardware Features				
Address remap	~	Yes	~	~
Data write protection	~	Yes	~	~
DMA	~	~	~	Yes
Flash Memory Accesses				
Memory access	RW	RW	RW	R
Data transfer size per command	Up to 8 bytes, or "read memory" up to 128 bytes	Reads up to device size Writes based on device command		

The programmed I/O must not be attempted when there is DMA activity.

STIG, Programmed I/O

Software can trigger individual flash instructions using register reads and writes.

Direct Mode

Memory-mapped direct mode generates flash data transfers.

Non-DMA Indirect

The controller can be programmed to read or write a block of data using register read/write operations. For reads, the controller prefetches data in preparation for the software read. For writes, it buffers data for the flash memory.

DMA Indirect

The DMA unit operations are performed by SRC DMA interfaces to the controller's DMA flash interface to read the flash memory via the controller's buffer and the DST DMA to write data to the system memory via the AXI master interface on the PMC IOP switch. The DMA peripheral interface decodes requests from the SRC DMA to read flash memory data.

Polling Feature

The OSPI flash auto-polling feature operates in all access modes. Polling is controlled by the expiration and flash status registers.

- OSPI.POLL_EXPIRE cycle count to polling expiration
- OSPI.POLL_STATUS with dummy cycle count

Start-up Sequences

Controller Reset

After reset, the controller is disabled and can be configured as needed.

There are several ways to reset the controller. These are described in the Controller Reset section of this chapter.

Controller Enable

The controller enable/disable function is used by software for several purposes.

Idle Status Bit

After a flash memory access has been initiated, software must wait for it to be completed before another access is initiated. The [IDLE] status bit is asserted after the controller has finished performing the flash memory access.

- Initiate a new access mode (e.g., from direct to DMA mode)
- Set-up polling of the flash memory status

DMA Programming Model

The DMA reads from the flash memory and writes to system memory; a flash memory read-only operation.

The DMA controllers are divided between the SRC DMA for data reads from the flash memory and the DST DMA for data writes to addressable system memory via the PMC IOP AXI switch.

DMA Features

The DMA includes the following features:

- Separate read channel (SRC) and write channel (DST)

- Simple DMA, no scatter-gather
- DST connects to 32-bit AXI to PMC IOP interconnect
- 128-word FIFO
- SRC DMA reads from flash memory controller when space is available in FIFO
- DST DMA writes data when data is available in the FIFO
- DMA start address is 4-byte aligned
- DMA transfer length is in 4-byte words
- DST DMA INCR burst type
- Timeout mechanisms for SRC and DST DMA
- Automatic DST DMA hardware management for 4 KB boundary crossing
- Configurable AXI AxUSER bits for coherency and QoS

Programming Steps

This section provides the steps for configuring the DMA controller. Be sure to follow the guidance provided in the Configuration Restrictions section.

1. Configure the I/O interface.
2. Configure the DMA controller:
    a. Program the OSPI.DMA_DST_ADDR_L and DMA_DST_ADDR_H registers with the destination address in main memory; must be word aligned.
    b. Program the OSPI.DMA_SRC_ADDR register to be the same as the OSPI.IND_AHB_ADDR_TRIG register.
    c. Program the OSPI.DMA_DST_SIZE with the number of words to be transferred (word aligned). This should be the same as the OSPI.IND_READ_NUM register.
    d. Program the OSPI.DMA_DST_CTRL_1 and OSPI.DMA_DST_CTRL2 register as required.
3. Start the indirect read in flash memory I/O controller by setting OSPI.IND_READ_CTRL [start] = 1. Wait until the DST_I_STS [DONE] bit is set to check if the AXI command transfer has completed before accessing the data transferred to memory.

Source DMA

The SRC DMA generates an address to the flash controller via its interface to read data from the flash memory device.

The SRC DMA writes the data from flash device to the DMA buffer. The DST DMA then writes the data in the buffer to system memory. Single transactions are four bytes and burst transactions are always 64 bytes.

Control Settings

The SRC DMA settings are shown in the following table.

Table 192: **OSPI Source DMA Control Settings**

Register	Address Offset	Write Value	Description
OSPI.DMA_PERIPH_CFG	0x0020	0000_0602h	Transfer size: four bytes for single AXI transfer and 64 bytes for AXI burst transfer. The DMA does not support any other transfer sizes.
OSPI.IND_READ_THRESH	0x0064	0000_0000h	To indicate data availability as and when sufficient data is available in the buffer.
OSPI.IND_TRIG_ADDR	0x0080	0000_0006h	The SRC DMA can execute only a fixed burst of 64-bytes.
OSPI.SRAM_PARTITION_CFG	0x0018	0000_00FEh	Allocate the 1 KB buffer for SRC DMA read operations.

Source DMA Interrupts

The SRC DMA interrupts in the OSPI.DMA_SRC_ISR register are summarized in the following table. These interrupts are not used during normal operation, but can provide information for test and debug.

Table 193: **OSPI Source DMA Interrupts**

Interrupt	Bit	Description
[MEM_DONE]	0	The DMA has completed current command of all reads of the flash memory
[DONE]	1	DMA has completed a command
[AXI_RDERR]	2	Error reading data from flash controller
[TIMEOUT_STRM]	3	Timeout counter 1 expired; flash controller is stalled
[TIMEOUT_MEM]	4	Timeout counter 2 expired; DMA is stalled
[THRESH_HIT]	5	FIFO watermark hit
[INVALID_APB]	6	APB programming interface address decode error

Destination DMA

The DST DMA generates an address to the system memory via its 32-bit AXI master interface to write data from the flash memory device that is in the DMA buffer to system memory.

Destination DMA Interrupts

The DST DMA interrupts in the OSPI.DMA_DST_ISR register are summarized in the following table. The status bits show the raw (before the mask) interrupt event. Each interrupt is cleared by writing a 1 to the bit (W1C).

Table 194: **OSPI DST DMA Interrupts**

Interrupt	Bit	Description
[DONE]	1	DMA is done and all data is sent; BRESP received
[AXI_BRESP_ERR]	2	DMA write generated a BRESP error on AXI
[TIMEOUT_STRM]	3	Timeout counter 2 expired; data from SRC DMA stalled
[TIMEOUT_MEM]	4	Timeout counter 1 expired; AXI interface stalled
[THRESH_HIT]	5	FIFO reached threshold limit
[INVALID_APB]	6	APB programming interface address decode error
[FIFO_OVERFLOW]	7	FIFO overflow detected

Configuration Restrictions

- In addition to the above registers, the OSPI.IND_AHB_ADDR_TRIG register must be programmed so there is not a 4 KB boundary crossing between the value programmed and the value + 63.
- The OSPI.IND_READ_NUM register should be programmed so that the number of bytes read is word-aligned. Bits 1 and 0 should always be programmed to 0 for this register.
- Program the DMA_TOP and then trigger the indirect read transfer using IND_READ_CTRL [start].
- For direct transfers, all the AXI transactions must be aligned.
- For direct write transfers, only the following WSTRB are supported.

AxSIZE	WSTRB values supported
`2'b10`	`4'b1111`
`2'b01`	`4'b1100, 4'b0011`
`2'b00`	`4'b0001, 4'b0010, 4'b0100, 4'b1000`

- Program the PMC_IOP_SLCR.IOP_AXI_MUX_SEL register with `3h` to enable direct mode and AXI slave interface.
- For direct read and writes, pipeline mode needs to be always enabled.
- STIG mode supports only 1-1/0-1/0 (command- address-data) and 8-8/0-8/0 commands.
- STIG mode does not support 1-8-8 and 1-1-8 commands.
- Only one indirect mode operation can be triggered at a time. Next indirect operation can be triggered after the first indirect operation is complete.

Architecture

The AHB interface is used to transfer data, either in a memory mapped direct fashion (for example software wishing to execute code directly from flash memory), or in an indirect fashion where the controller is setup via configuration registers to silently perform some requested operation, signaling its completion via interrupts or status registers.

For indirect operations, data is transferred between system memory and flash memory via an internal SRAM. Interrupts or status registers are used to identify the specific times at which this SRAM should be accessed using user programmable configuration registers.

The DMA peripheral bus optimizes data transfers to the flash memory and PHY during indirect transfers.

DMA Controller Implementation

The SRC and DST DMA controller has the same programming model as the CSU DMA in the Zynq® UltraScale+™ MPSoC.

The data read from the flash memory by the SRC DMA is put in a buffer for the DST DMA to access and write out to system memory using the controller's AXI master interface.

Interrupts

There are several interrupt register sets:

- Controller Interrupts
- Source DMA Interrupts
- Destination DMA Interrupts

Controller Interrupts

The controller interrupts are latched into the OPSI.IRQ_STATUS register. The source for each interrupt is indicated in the following table.

Table 195: **OSPI Flash Controller Interrupts**

Interrupt	Bit	Interrupt Source					Description
		STIG	Direct	Indirect Non-DMA	Indirect DMA	Polling	
IND_OP_DONE	2	~	~	Yes	Yes	~	Indirect operation complete

Table 195: **OSPI Flash Controller Interrupts** *(cont'd)*

Interrupt	Bit	Interrupt Source					Description
		STIG	Direct	Indirect Non-DMA	Indirect DMA	Polling	
WPROT_ATTEMPT	4	Yes					Write protect access attempted
ILLEGAL_ACCESS_DET	5						Illegal AHB slave interface access attempted
IND_RD_SRAM_FULL	12	~	Yes				Indirect read partition overflow
POLL_EXP	13	~	~	~	~	Yes	Polling time period counter expired
STIG_REQ_RDY	14	Yes	~	~	~	~	

Register Reference

The OSPI registers are divided into several overview tables:

- OSPI Controller Registers
- OSPI SRC DMA Registers
- OSPI DST DMA Registers
- System-Level Registers: from PMC_IOP_SLCR register set

OSPI Controller Registers

The OSPI flash memory I/O controller registers are listed in the following table. The base address for these registers is `0xF101_0000`.

Table 196: **OSPI Flash Memory I/O Controller Register Overview**

Register Name	Address Offset	Access Type	Description
CONFIG	0x0000	RW	Controller configuration
DEV_INSTR_RD_CFG DEV_INSTR_WR_CFG	0x0004 0x0008	RW	Device read and write instruction configurations
DEV_DELAY	0x000C	RW	I/O timing delay
RD_DATA_CAPTURE	0x0010	RW	Read data capture
DEV_SIZE_CFG	0x0014	RW	Device size configuration
SRAM_PARTITION_CFG	0x0018	RW	SRAM partition configuration

Table 196: **OSPI Flash Memory I/O Controller Register Overview** *(cont'd)*

Register Name	Address Offset	Access Type	Description
IND_AHB_ADDR_TRIG	0x001C	RW	Indirect AHB address trigger
DMA_PERIPH_CFG	0x0020	RW	DMA peripheral configuration
REMAP_ADDR	0x0024	RW	Remap address
MODE_BIT_CFG	0x0028	RW	Mode bit configuration
SRAM_FILL	0x002C	R	SRAM fill
WRITE_COMPLETION_CTRL	0x0038	RW	Write completion control
NO_OF_POLLS_BEF_EXP	0x003C	RW	Polling expiration
IRQ_STATUS IRQ_MASK	0x0040 0x0044	WTC RW	Interrupt status, mask
W_PROT_START W_PROT_END	0x0050 0x0054	RW	Write protection, lower and upper boundaries
W_PROT_CTRL	0x0058	RW	Write protection control
IND_READ_CTRL IND_READ_THRESH IND_READ_START IND_READ_NUM	0x0060 0x0064 0x0068 0x006C	R, W, WTC RW RW RW	Indirect read transfer control Indirect read transfer watermark Indirect read transfer start address Indirect read transfer number bytes
IND_WRITE_CTRL IND_WRITE_THRESH IND_WRITE_XFER_START IND_WRITE_XFER_NUM_BYTES IND_WRITE_TRIG_ADDR	0x0070 0x0074 0x0078 0x007C 0x0080	R, W WTC RW RW RW	Indirect write transfer control Indirect write transfer watermark Indirect write transfer start address Indirect write transfer number bytes Indirect trigger address range
MEM_CMD_CTRL	0x008C	R, W, RW	Flash command control memory
FLASH_CMD_CTRL	0x0090	R, W, RW	Flash command control
FLASH_CMD_ADDR	0x0094	RW	Flash command address
FLASH_READ_L FLASH_READ_U	0x00A0 0x00A4	R	Flash command read data, lower and upper
FLASH_WRITE_L FLASH_WRITE_U	0x00A8 0x00AC	RW	Flash command write data, lower and upper
FLASH_POLL_STATUS	0x00B0	R	Polling flash status
PHY_CONFIG	0x00B4	W, RW	PHY configuration
PHY_MASTER_CTRL	0x00B8	RW	PHY DLL master control
DLL_OBSERVE_L DLL_OBSERVE_H	0x00BC 0x00C0	R	DLL observable, lower and upper
OPCODE_EXT_L OPCODE_EXT_U	0x00E0 0x00E4	RW	Opcode extension, lower and upper
SAFETY_CHK	0x1FF8	RW	Safety check register for OSPI register set

OSPI SRC DMA Registers

The DMA controller registers are listed in the following table. The base address for these registers is 0xF101_0000.

Table 197: **SRC DMA Register Overview**

Register Name	Address Offset	Access Type	Description
DMA_SRC_ADDR	0x1000	RW	Source DMA read address
DMA_SRC_SIZE	0x1004	R	Source DMA read payload size
DMA_SRC_STS	0x1008	R, WTC	Source DMA read status
DMA_SRC_CTRL1 DMA_SRC_CTRL2	0x100C 0x1024	RW	Source DMA read control Reg 1 and 2
DMA_SRC_ISR DMA_SRC_IER DMA_SRC_IDR DMA_SRC_IMR	0x1014 0x1018 0x101C 0x1020	WTC W W R	Source DMA read interrupt status, enable, disable, and mask

OSPI DST DMA Registers

The DMA controller registers are listed in the following table. The base address for these registers is 0xF101_0000.

Table 198: **OSPI SRC DMA Register Overview**

Register Name	Address Offset	Access Type	Description
DMA_DST_ADDR_L DMA_DST_ADDR_H	0x1800 0x1828	RW	Destination DMA write address to system memory, 32 LSBs and 17 MSBs
DMA_DST_SIZE	0x1804	RW	Destination DMA write payload size
DMA_DST_STS	0x1808	R, WTC	Destination DMA write status
DMA_DST_CTRL1 DMA_DST_CTRL2	0x180C 0x1824	RW	Destination DMA write control reg 1 and 2
DMA_DST_ISR DMA_DST_IER DMA_DST_IDR DMA_DST_IMR	0x1814 0x1818 0x181C 0x1820	WTC W W R	Destination DMA write interrupt status, enable, disable, and mask

System-Level Registers

The PMC IOP SLCR related registers are listed in the following table. The offset address for PMC_IOP_SLCR is relative to 0xF106_0000.

Table 199: **OSPI-related Registers in the PMC_IOP_SLCR Register Set**

Register Name	Address Offset	Access Type	Description
OSPI_IOP_AXI_SEL	0x0504	RW	Set AXI interface mode: 2: DMA mode 3: Linear mode
OSPI_IOP_COH_CTRL	0x0530	RW	Define transaction coherency and bufferability policy
OSPI_IOP_INT_ROUTE	0x0534	RW	Route through FPD CCI (for APU L2-cache coherency) or bypass it (non-coherent)
OSPI_IOP_INT_QOS	0x053C	RW	QoS traffic type

OSPI I/O Interface

The OSPI controller I/O signals are only available on the PMC MIO. The interface is not available on the LPD MIO or the PL EMIO interface.

I/O Signal Table

The I/O signals are shown in MIO-at-a-Glance Tables and detailed in the following table.

Two devices can be attached to the I/O interface. The OSPI_CS0_b and OSPI_CS1_b are used for the stacked device configuration. For non-stacked configurations, use OSPI_CS0_b.

Table 200: **OSPI Controller I/O Signals**

MIO				Description
Signal Name	I/O	PMC MIO Pin	MIO-at-a-Glance Table	
OSPI_CLK	Output	0	0	Clock output
OSPI_IO[0] OSPI_IO[1] OSPI_IO[2] OSPI_IO[3] OSPI_IO[4]	I/O	1 2 3 4 5	1 2 3 4 5	I/O signals
OSPI_DS	Input	6	6	Read data strobe
OSPI_IO[5] OSPI_IO[6] OSPI_IO[7]	I/O	7 8 9	7 8 9	I/O signals
OSPI_CS0_b	Output	10	10	Chip select 0, active-Low
OSPI_CS1_b	Output	11	11	Chip select 1, active-Low[1]

Table 200: **OSPI Controller I/O Signals** *(cont'd)*

MIO				Description
Signal Name	**I/O**	**PMC MIO Pin**	**MIO-at-a-Glance Table**	
OSPI_RST_b	Output	12	12	PMC GPIO controller output used for reset[2]

Notes:

1. When one device is connected, it can be connected to CS0_b or CS1_b.
2. The OSPI boot process and the Xilinx software drivers use an output signal from the PMC GPIO Controller (bank 0, channel 12) to reset the flash device.

Quad SPI Controller

The quad SPI (QSPI) controller can access one or two flash devices using several different methods. The controller is located with the other flash memory controllers in the PMC. The I/O interface is routed to the PMC MIO pin bank 0 and can drive one or two devices. QSPI is commonly used as a boot device, see Quad SPI Boot Mode. The controller provides multiple ways to read and write flash memory:

- SPI accesses
- Programmed I/O (PIO) protocol
- DMA indirect read using AXI master interface

SPI accesses enables software to control the bus protocol and read/write memory data via the APB programming interface.

Linear addressing is not supported. Processor software also cannot execute code directly from the controller; execute-in-place is not supported.

For DMA, data is autonomously read from the flash memory and written to system memory via the TXFIFO. The DMA master is on the PMC main AXI switch. The DMA includes a TXFIFO.

Software sends commands to the controller using the flash command register. The commands are buffered in the command FIFO. Commands include configuration, SPI commands (opcode, address, mode, dummy), and single byte reads and writes.

The controller also includes a programmable polling features to read the flash device status and report when a certain value is received.

The interface works with up to two flash devices. The I/O interface is routed to the PMC MIO multiplexer, bank 0. The I/O signals are not available on the LPD MIO pins or the PL EMIO port signal interface.

The data signals are divided between upper and lower signals; four data bits each with a clock and a chip select. The two-device implementation can be stacked with a 4-bit I/O interface, or connected in parallel with an 8-bit interface for higher performance.

Software accesses the QSPI register module via the 32-bit APB programming interface. All of the QSPI related registers are listed in the Register Reference.

Features

Interconnect

- 32-bit APB slave programming interface
- AXI master interface for DMA controller writes to system memory

I/O Configurations

The controller can interface with one or two flash memory devices. Basic connections include:

- 4 and 8-bit I/O signals
  - Single device with a x4 data bus width
  - Dual device, stacked configuration with a x4 data bus width
  - Dual device, parallel configuration with a x8 data bus width

Flash Memory Addressing

- 128 Mb and larger devices
- 16 MB addressing per device (32 MB for two devices)

Power Domain

The QSPI controller is in the PMC power domain.

Comparison to Previous Generation Xilinx Devices

The QSPI controller is similar to the Zynq® UltraScale+™ MPSoC except for the following:

- Legacy mode (LQSPI) including linear addressing is not supported
- AXI master DMA interface included enhanced control for coherency, buffer-ability, and quality of service

System Perspective

The controller is located in the PMC on the IOP interconnect switch and has the following interfaces and signals.

- 32-bit APB slave programming interface
  - Memory mapped, programming registers, PMC Address Map
  - Control, status, and interrupt registers
  - FIFO data ports
- AXI master interface for DMA controller writes to system memory
  - 44-bit physical address with data bursts transfers
  - AxCACHE bit settings for coherency and bufferability
- QSPI_REF_CLK input for controller logic
- QSPI_RESET reset input
- Flash device interface
  - 4 and 8-bit I/O signals
  - QSPI_CLK clock, chip select, and loopback clock

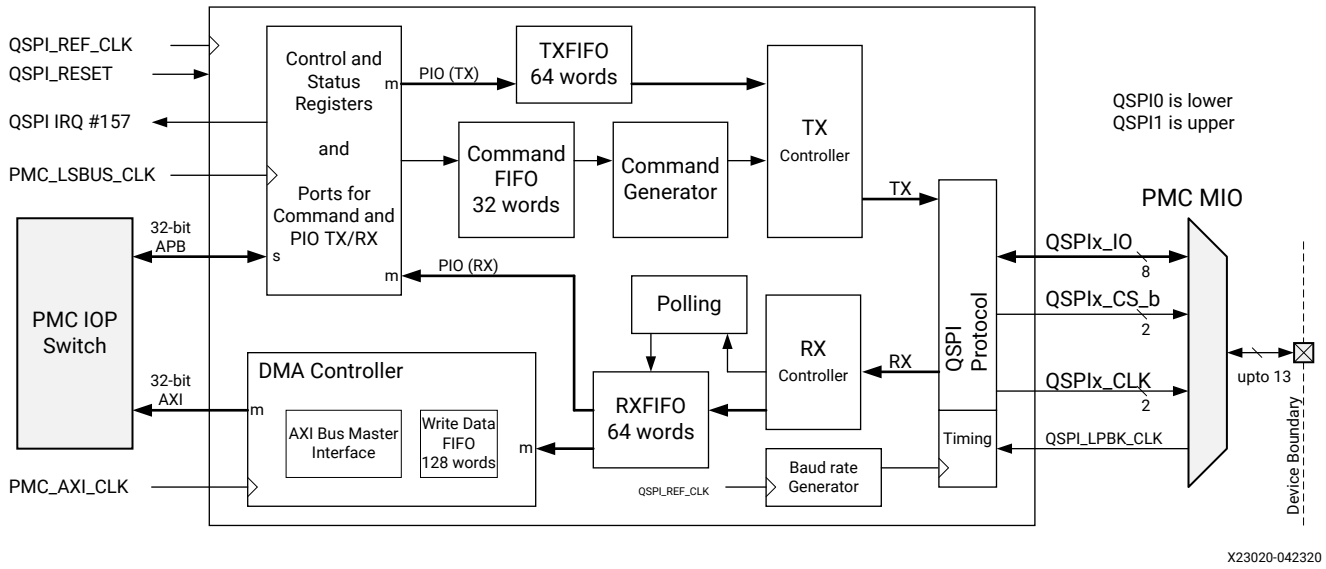
The QSPI controller can be a primary boot device. For more information, see Quad SPI Boot Mode.

Block Diagram

The I/O interface is routed to the PMC MIO bank 0 pins. The I/O pins are listed in MIO Signal Tables. The PMC MIO bank is shown in MIO-at-a-Glance Tables.

The high-level block diagram is shown in the following figure.

Figure 94: **QSPI High-level Block Diagram**



Functional Units

The high-level block diagram includes several major functional units.

TXFIFO and RXFIFO

The controller has a 64-word TXFIFO for sending content to the I/O interface and a 64-word RXFIFO for receiving data from the I/O interface. Use the QSPI.TXD register data port to write data to the TXFIFO and the RXD register data port to read data from the RXFIFO.

Command FIFO

Software writes 20-bit command words to QSPI.GEN_FIFO to configure and initiate transactions on the I/O interface. The command generator initiates transactions that are driven by the command fields. The controller transmits data written to the TXFIFO and receives data read from the RXFIFO.

Polling

The controller can repeatedly read the status of a the flash device looking for a specific pattern. This can be used to monitor the status of a flash device operation or other purpose.

DMA Controller

The DMA controller is used to move large blocks of data from the flash device to system memory. This is a master, write-only DMA controller on the AXI bus interface. It can only be used to read data from the flash device and write the data to system memory.

Interfaces

The controller has two system interfaces and a single I/O interface.

32-bit APB Slave Programming Interface

- Memory mapped, programming registers, PMC Address Map
- Control, status, and interrupt registers
- FIFO data ports

AXI Master Interface for DMA Controller Memory Writes

- 44-bit physical address with 32-word data bursts
- AxCACHE defines coherency and buffer-ability of the transaction

Flash Memory I/O Interface

- 4 and 8-bit data I/O (one or two devices)
- Chip select
- QSPI_CLK clock
- Loopback clock

System Interfaces

The DMA is a burst-enabled, 32-bit AXI master on the PMC IOP interconnect. There are several PMC_IOP_SLCR registers available to control the AxCACHE and AxUSER transaction parameters.

The AXI write transactions include three options:

- Coherent or non-coherent with the FPD CCI
- Buffer-ability in the system
- Quality of service (QoS) settings

Coherent and Bufferable Transactions

The AXI coherency and bufferable transaction attribute can be programmed using the QSPI_AXI_COH register. If hardware coherency with the APU L2 cache is needed, the transaction must be routed through the FPU CCI using the PMC_IOU_SLCR.QSPI_IOP_INT_ROUTE register. If coherency is not enabled, the FPD CCI can be bypassed for higher performance.

Note: In most applications, coherency is managed by software. The hardware coherency through the CCI slows down the DMA transfers and can severely impact the performance of the APU.

QoS

There are three classes of QoS transactions, which are explained in Quality of Service. Normally, the best effort class is chosen. The class selection is controlled by the QSPI_IOP_INTERCONNECT_QOS register.

System Signals

The system signals connected to the QSPI controller include:

- Clocks
- Controller Resets
- System Interrupt
- System Error

The controller connects to several signals coming from and going to the system.

Clocks

The controller receives two clocks from the system.

IOP Switch Interface Clock

The three system interfaces are clocked by the IOP_IRO_CLK clock used for the IOP switch.

Reference Clock

The controller reference clock is generated by the PMC clock controller using a CRP register.

- OSPI_REF_CLK controlled by the CRP.OSPI_REF_CTRL register

Controller Resets

Other resets are used to clear separate parts of the controller under software control by writing to the QSPI.FIFO_CTRL register. The QSPI-specific resets include:

- [RST_GEN_FIFO]
- [RST_TX_FIFO]
- [RST_RX_FIFO]

System Interrupt

The QSPI register module OR's the individual controller interrupts and generates a single system interrupt, IRQ #157.

System Error

The APB programming interface generates an address decode error if it detects an access violation. This is OR'd together with the APB address decode errors from other PMC blocks to create the PMC APB error in the PMC_GLOBAL.PMC_ERR2_ISR [0] register bit.

I/O Interface

The I/O interface is only available on the PMC MIO pins.

I/O Wiring Diagrams

The I/O wiring connections are shown in Wiring Diagrams. The diagrams for boot modes are shown in Quad SPI Boot Mode.

I/O Signals

The QSPI I/O signals are listed in the table in the MIO Signal Tables.

Programming Model

The controller is managed by the QSPI register module. This include memory-mapped control and status registers. The DMA has a simple programming model that is controlled by QSPI registers.

Modes and States

Operating Modes

The controller operate modes include:

- Low-level protocol
- Programmed I/O (PIO) protocol
- DMA read flash, write system memory
- Read data polling

Start-up

The normal bring up process:

- Establish the QSPI_REF_CLK (PMC clock controller)

- Release the reset
- Set the baud-rate divisor (QSPI divider)
- Configure the controller
- Enable interrupts
- Set PIO or DMA mode
- Enable the controller
- Issue controller command
- PIO operating mode
  - Read/write data via TX/RXFIFOs
  - Issue controller commands
  - Monitor FIFOs in interrupt handler
- DMA operating mode
  - Configure and launch the DMA transfer
    - Program DMA controller
    - Program interrupts
    - Send SPI command to flash device using controller commands sent to the command FIFO
    - Initiate DMA transfer
  - Wait for the DMA done interrupt [DONE] to be generated

Reset

The QSPI registers are reset by a POR or by the PMC reset controller using the RST_QSPI register.

A controller reset is required when:

- The QSPI_REF_CLK clock frequency is changed. The clock control is described in System Perspective
- When both the baud-rate divisor and the I/O device mode is changed. For example, a reset is required before changing from a single, or stacked device mode with a baud rate setting of 4 and then switching to the dual-parallel mode with a baud rate of 2.

PIO Mode

For PIO mode operation, follow these steps.

1. Select the generic quad SPI controller by writing a `1` to the generic_qspi_sel register bit.
2. Set the mode_en bits = `2'b00` of the GQSPI_CFG register.
3. Check to make sure that the generic FIFO is not full and then write the data into the generic FIFO using a read or write command request on the APB interface.
4. Write the TX data into the TXFIFO when there is a write transfer over the APB interface.
5. When there is a write request, the generic quad SPI controller sends the command, address, dummies from the generic FIFO and sends write data from the TXFIFO.
6. When there is a read request, the generic quad SPI controller sends the command, address, dummies from the generic FIFO and sends read data into the RXFIFO.
7. Read requests are issued from the APB interface to receive the RX data.

When two flash devices are connected in stacked mode, the generic quad SPI controller checks for the data bus select field of the generic FIFO and sends the requests accordingly.

DMA Mode

For DMA mode operation, follow these steps.

1. Select the generic mode by writing a 1 to the QSPI.GEN_SEL register bit.
2. Write the command, address, dummies in the generic FIFO using the read request.
3. The generic quad SPI controller sends the command as programmed in the generic FIFO and reads the data into the RXFIFO.
4. The DMA controller issues DMA requests using the AXI master interface and sends the RXFIFO data.

I/O Functionality

The flash device I/O interface has three configurations. The configurations are controlled by the command word written to the GEN_FIFO data port.

For timing, see the Versal ACAP data sheets.

Configurations

The I/O interface configurations are summarized in the following table. For more information, see Quad SPI Boot Mode topics.

Table 201: **Quad-SPI I/O Configurations**

I/O Type	Device Count	Chip Selects	Data Signals
Single 4-bit	1	Either[1]	Up to 4
Dual stacked	2	Either[1]	Up to 4
Dual parallel	2	Both[2]	8

Notes:

1. In the first case, either chip select can be used.
2. QSPI0_CS_b is associated with lower four data bits. QSPI1_CS_b is associated with upper four data bits.

Clocking

The QSPI_SCLK clock signal for the flash device I/O interface comes from the QSPI baud-rate generator. The baud-rate generator takes in the QSPI_REF_CLK clock and divides it down using the QSPI.CFG [BAUD_RATE_DIV] field to generate QSPI_SCLK.

The QSPI_REF_CLK clock frequency must be 2x the QSPI_SCLK I/O device clock frequency for higher than 37.5 MHz clocking.

SCLK I/O Loopback Clock

The QSPI_LPBK_CLK is generated from the QSPI_SCLK and routed through the output buffer to a PMC MIO pin and returned back through the pin's input buffer to the controller for I/O delay compensation for greater timing accuracy. The I/O loopback clock signal is only used for I/O clocking >37.5 MHz. When the QSPI_SCLK device clock frequency is >37.5 MHz, the QSPI_LPBK_CLK must be routed to PMC MIO [6] and must be left unconnected on the PCB.

Clock Tap Control Settings

The three clock frequency ranges are shown in the following table.

Table 202: **QSPI Clock Tap Delay Settings**

Control	QSPI_CLK Frequency Range			Register Bit Field
	≤ 37.5 MHz	>37.5 to 100 MHz	>100 to 150 MHz	
Data tap delay unit bypass	Bypass (1)	Bypass (1)	Enable (0)	PMC_IOP_SLCR.IOP_TAPDLY_BYPASS [LQSPI_RX] 0: Enabled, use tap delay 1: Bypass
Clock loopback pin enable	Disable (0)	Enable (1)	Enable (1)	LPBK_DLY_ADJ [USE_LPBK][1] 0: Disable 1: Enable
Data tap delay settings	00, 000	00, 000	01, 000	LPBK_DLY_ADJ [DLY1], [DLY0]

Table 202: **QSPI Clock Tap Delay Settings** *(cont'd)*

Control	QSPI_CLK Frequency Range			Register Bit Field
	≤ 37.5 MHz	>37.5 to 100 MHz	>100 to 150 MHz	
Data delay enable	Enable (1)	Enable (1)	Disable (0)	DATA_DLY_ADJ [USE_DATA_DLY]
Data delay adjustment	000	000	000	DATA_DLY_ADJ [DATA_DLY_ADJ]

Notes:

1. If loopback is enabled, the QSPI_LPBK_CLK signal pin must be routed through the PMC MIO pin 6 and left unconnected on the PCB.

I/O Striping Function

Striping Programming Examples

The following table lists QSPI striping examples.

Table 203: **QSPI Striping Examples**

	[imm_data]	[data_xfer]	[stripe]	[transmit]	[receive]	[bus_select]
TX Data: to Both Flash Devices						
	EBh	0	0	1	0	11
TX Data: even Bytes to QSPI0, Odd Bytes to QSPI1						
	64h	1	1	1	0	11
TX Data: to Both Flash Devices (not common, but possible)						
	64h	1	0	1	0	11
RX Data: Note: [stripe] = 0 is N/A when [receive] = 1						
	64h	1	X[1]	0	1	11

Note: [Stripe] = 0 is not applicable when [receive] = 1.

Striping with Odd Byte Count

The generic QSPI controller transfers the data using the programmed data length in the immediate_data field. When the data length bytes are odd, to send the last data byte, the lower data bus is active for extra byte time than the upper data bus. For example, when the immediate_data field is 5 bytes and the stripe option is used, the bytes 0, 2, and 4 (total of 3 bytes) are sent/received on the lower data bus and 1, 3 (total 2 bytes) are sent/received on the upper data bus. The SCLK of the lower and upper are toggled accordingly.

Command Words

Controller commands are buffered in the command FIFO and are processed in order. The commands are used in all operating modes.

Word Format

The word format is designed to closely manage the flow of RX and TX data. Receive data is always via the RXFIFO. Transmit data can be written via the 32-bit TXFIFO or the 8-bit immediate data field in the command word.

Software writes the controller commands to the FIFO_DATA register. They are buffered in the 32-word deep command FIFO. The command word fields are described in the following table.

Table 204: **Controller Command Word Format**

Field Name	Bits	Description
[IMM_DATA]	7:0	Multipurpose field for data or byte count
[DATA_XFER]	8	Select [IMM_DATA] field usage: 0: Immediate write data RXFIFO 1: Data byte count in the format defined by [EXP]; write data is sent via the TXD data port ***Note*:** Read data is always received via the RXD data port (RXFIFO).
[EXP]	9	Select byte count calculation method: 0: Absolute count is in [IMM_DATA] field (maximum of 256 bytes) 1: Exponential count is calculated as byte transfer count = 2 ^ [IMM_DATA]
[MODE]	11:10	00: Reserved, do not use 01: 1-bit data I/O 10: 2-bit data I/O 11: 4-bit data I/O
[CS_LOWER]	12	Lower chip select control: 0: Deassert 1: Assert
[CS_UPPER]	13	Upper chip select control: 0: Deassert 1: Assert
[BUS_SEL]	15:14	Data bus enables for clocking: 00: No bus 01: Lower bus only 10: Upper bus only 11: Both buses Bus width depends on [MODE]

Table 204: **Controller Command Word Format** *(cont'd)*

Field Name	Bits	Description
[TX]	16	Transmit data enable: 0: Disable 1: Send data (immediate or via TXFIFO)
[RX]	17	Receive data enable: 0: Disable 1: Receive data (via RXFIFO)
[STRIPE]	18	Data stripe enable: 0: Disable (same data appears on upper and lower buses) 1: Enable (data is striped across lower and upper buses)
[POLL]	19	RX data polling enable 0: Disable 1: Enable

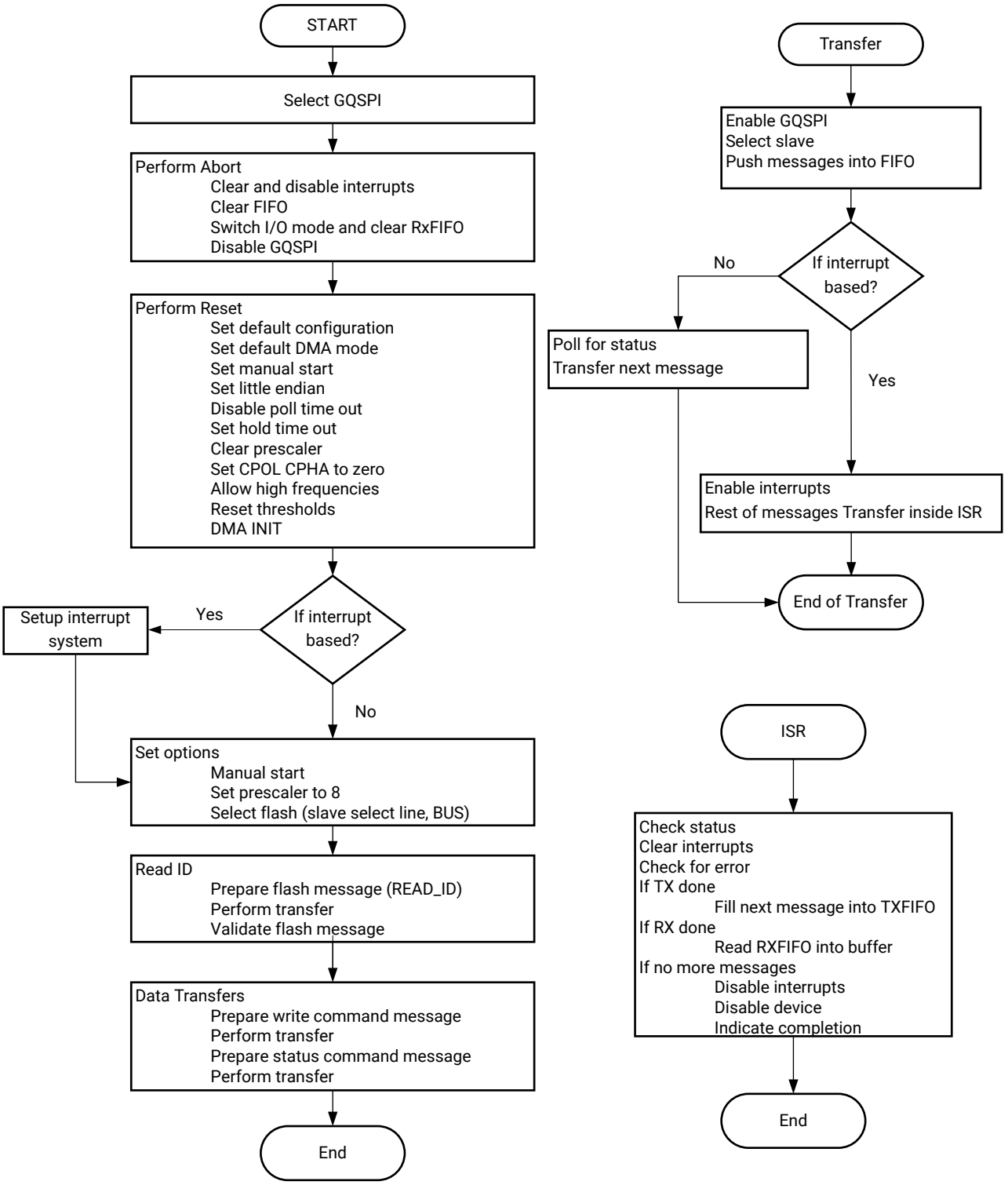
Programming

The overall programming guideline is shown in the Programming Flowchart. This is followed by several programming models:

- PIO Mode Programming Model
- DMA Programming Model
- Polling Programming Model

Programming Flowchart

Figure 95: **QSPI Programming Flowchart**



DMA Data Transfer Length Examples

The length of a data transfer is defined by one or more command words. Each command word includes a byte count. There are two ways to define the byte count. The method is defined by [EXPONENT]:

- 0: Byte count is in [imm_data]. Range is 0 to 128 (80h) bytes.
- 1: Exponent byte count = 2^[imm_data]. Range is 2, 4, 8, 16, and to megabytes.

Programming Examples

There are four data transfer length examples. Set [data_transfer] = 1.

Table 205: **QSPI Data Transfer Length Examples**

Byte Count	[imm_data]	[exponent]
64 Byte Transfer		
	40h	0
128 Byte Transfer		
	80h	0
1000 Byte Transfer - Option 1		
	09h	1
	08h	1
	E8h	0
1000 Byte Transfer - Option 2		
	08h	1
	08h	1
	08h	1
	E8h	0
1 GB Transfer		
	1Eh	1

PIO Mode Programming Model

In the programmed I/O (PIO) mode, the software interacts closely with the flash device protocol to read and write data. The device commands and data are written to the APB registers. The memory writes are sent to the flash device via the generic TXFIFO. Data from the flash device is read from the RXFIFO by reading the RXD register. The controller automatically fills the RXFIFO as data is read. Commands are used to set up data transfers. The RX and TXFIFOs are managed using the QSPI interrupts.

There are two options in the PIO mode to write the SPI command and transfer the data.

The first option is to send multiple controller commands to the command FIFO. The controller command includes an immediate 8-bit field that includes the SPI command and write data. For reads, the controller returns the byte data to the RXFIFO.

The other PIO access option is to initiate a transfer with a controller command and then write the SPI command and data content to the TXFIFO. In this case, the controller takes the content of the TXFIFO to send the SPI command and write data to the flash device. The read case is similar, except the data from the flash memory device is written to the RXFIFO by the controller. Software is expected to read data from the RXFIFO.

DMA Programming Model

The DMA mode reads data from the flash device and writes the data to system memory.

The main memory address is defined by [ADDRESS]. The DMA writes data for a length of [SIZE]. After software starts a DMA transfer, the software normally waits for an interrupt.

DMA Mode

Software sets up the DMA transfer and interrupt. It sends a controller command to initiate the transfer. To define the SPI command (for opcode, address, etc), the software writes multiple controller commands using the immediate field.

Polling Programming Model

The controller polling mode is used to repeatedly read a byte from the flash device and compare it to an expected 8-bit value. This mode is often used for status checking. The polling operation is configured by the QSPI.POLL_CFG register. When the data matches the value in [POLL_DATA] with the [MASK_EN] field applied, the data byte is written into the RXFIFO. This event can be programmed to generate an interrupt.

When two flash devices are used, the controller does not execute the next command until the data from both flash devices matches the value of the [POLL_DATA] with the [MASK_EN] field applied.

Status Checking Use Case

The polling operation is useful when checking the status of a flash device. For example, when a page program is issued to a flash device, the software itself can poll the status to check when the write is completed. This software polling requires multiple read requests from the flash device's status register. The polling operation autonomously reads the data and checks for the expected value independent of software.

Polling Timeout

The polling operation also includes a timeout feature. The timeout operation is configured by the POLL_TIMEOUT register and enabled by the CFG [EN_POLL_TIMEOUT] bit.

If the timeout occurs, the ISR [Poll_Time_Expire] interrupt bit is set to 1 if this interrupt is not masked. This interrupt bit asserts the QSPI system interrupt.

Register Reference

QSPI Registers

There are two register overview tables for the QSPI:

- QSPI register overview
- PMC_IOP_SLCR and CRP related registers for QSPI system control, clocking, and reset.

The QSPI registers are listed in the following table. The base address for these registers is 0xF103_0000.

Table 206: **QSPI Register Set**

Function	Register Name	Address Offset	Access Type	Description
I/O signals	IOP_TAPDLY_BYPASS	0x03C	R/W	I/O read timing
DMA	DMA_DST_ADDR DMA_DST_ADDR_MSB	0x800 0x828	W	AXI system memory address (49 bits) for DMA writes
DMA	DMA_DST_SIZE	0x804	W	DMA transfer size on AXI
DMA	DMA_DST_STS	0x808	R/WTC	DMA live status (polling event is sticky)
DMA	DMA_DST_CTRL	0x80C	R/W	DMA flow control, timeout, and endianess
Commands	DMA_DST_CTRL2	0x824	R/W	Maximum outstanding commands, timeout configuration

Table 206: **QSPI Register Set** *(cont'd)*

Function	Register Name	Address Offset	Access Type	Description
DMA interrupts	DMA_DST_I_STS DMA_DST_I_MASK DMA_DST_I_EN DMA_DST_I_DIS	0x814 0x820 0x818 0x81C	R W W R	DMA interrupts
Configuration	CFG	0x100	Mixed	Configuration: clock phase and polarity, baud-rate, PIO endianess, command FIFO enable, controller mode
Configuration	ENABLE	0x114	RW	Enable controller
Data flow	TXD	0x11C	W	Data to TXFIFO
Data flow	RXD	0x120	R	Data to RXFIFO
Write protect	GPIO	0x130	RW	Active-Low, nWP
I/O signals	LPBK_DLY_ADJ	0x138	RW	Clock loopback adjustment and enable
Commands	GEN_FIFO	0x140	W	Controller command FIFO write port
Controller mode	SEL	0x144	RW	GQSPI mode, set = 1
FIFO control	FIFO_CTRL	0x14C	W	Reset RX, TX, and command FIFOs
PIO interrupts	ISR IMR IER IDR	0x104 0x110 0x108 0x10C	R/WTC W W R	TX and RX FIFO interrupts
Data flow	TX_THRESH	0x128	RW	Configure the TXFIFO thresholds
Data flow	RX_THRESH	0x12C	RW	Configure the RXFIFO thresholds
Controller commands	GF_THRESH	0x150	RW	Controller command FIFO threshold
Polling	POLL_CFG	0x154	RW	Configure the polling byte-value criteria
Polling	POLL_TIMEOUT	0x158	RW	Polling timeout value in number of QSPI_REF_CLK cycles

QSPI I/O Interface

The QSPI has several I/O wiring and boot options. The wiring options are shown in the following figure. This is followed by two tables that describe the I/O signals. The boot options are listed in Boot Modes.

These I/O configurations are supported by the programmed I/O access and DMA read modes.

Wiring Diagrams

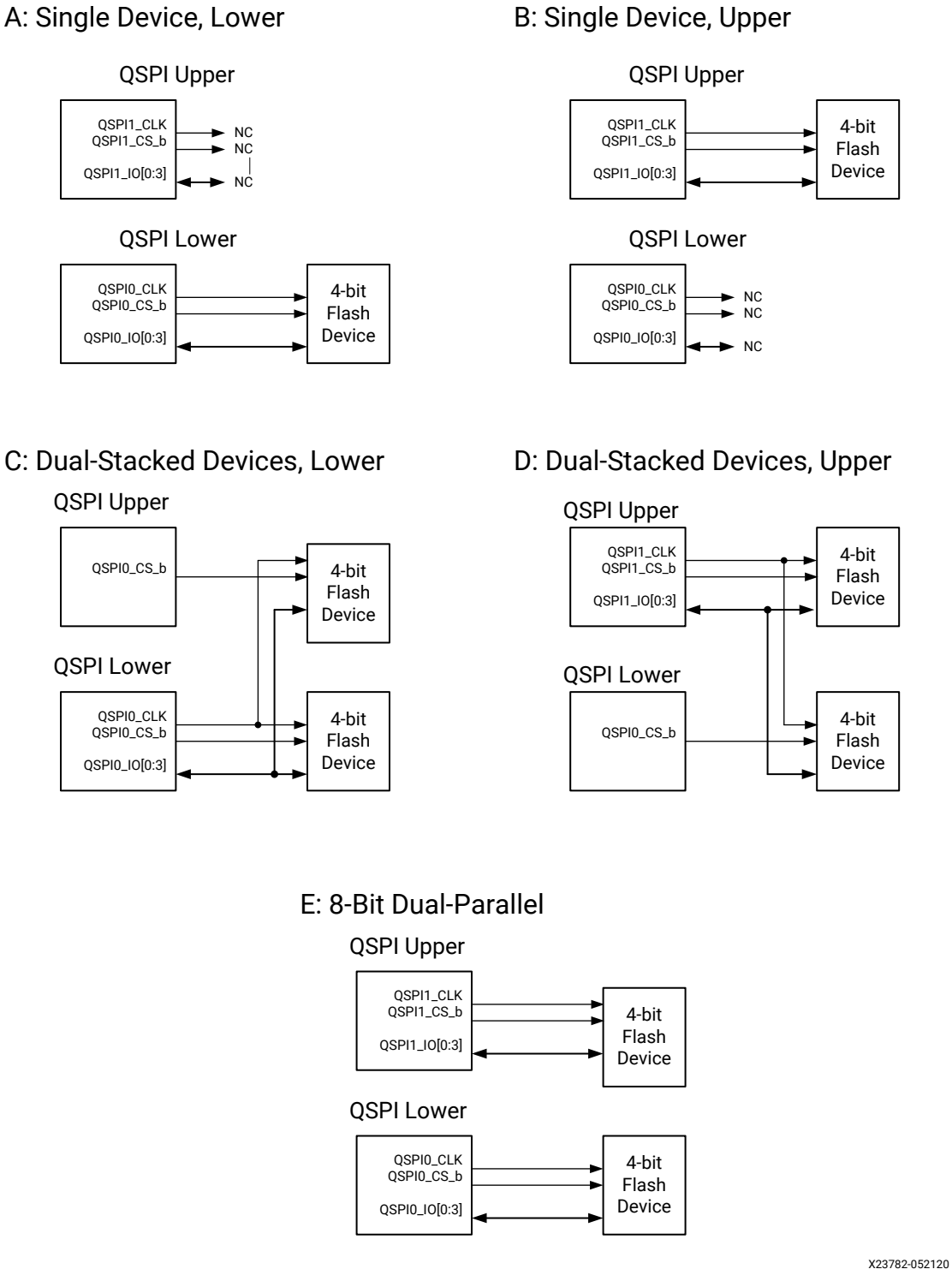
There are five I/O interfacing options for the flash device:

- A: Single 4-bit device on lower interface (a boot mode)

- B: Single 4-bit device on upper interface
- C: Dual-stacked 4-bit on lower clocks and data (a boot mode)
- D: Dual-stacked 4-bit on upper clocks and data
- E: Dual-parallel 8-bit (a boot mode)

For boot modes, see Quad SPI Boot Mode.

Figure 96: **QSPI I/O Interface Connection Diagrams**



MIO Signal Tables

The I/O interface is only available on the PMC MIO pins. The interface is not available on the LPD MIO pins or the PL EMIO interface.

The QSPI controller signals are listed in the following tables. The interface includes lower and upper controls, QSPI0 and QSPI1, respectively. The controller always drives the interface clock signal as an output.

Table 207: **Quad SPI Flash Interface I/O Signals**

Versal ACAP Signal Name	Flash Interface Protocols					
	1-bit		2-bit		4-bit	
	I/O	Name	I/O	Name	I/O	Name
QSPIx_CLK	O	CLK	O	CLK	O	CLK
QSPIx_CS_b	O	CS_b	O	CS_b	O	CS_b
QSPIx_IO[0]	O	MOSI	I/O	IO[0]	I/O	IO[0]
QSPIx_IO[1]	I	MISO	I/O	IO[1]	I/O	IO[1]
QSPIx_IO[2]	O	WP_b	O	WP_b	I/O	IO[2]
QSPIx_IO[3]	O	HOLD_b	O	HOLD_b	I/O	IO[3]
QSPI_LPBK_CLK	Enable for clock frequencies >37.5 MHz; it is a no connect on the PCB.					

MIO Configuration Table

The five MIO connection options are shown in Wiring Diagrams and are listed in the following table.

***Note*:** The loopback clock signal is routed from the controller through the output buffer to the pin and returned back through the pin's input buffer to the controller for I/O delay compensation. The loopback clock signal is used by both QSPIx_CLK outputs via a clock gating circuit.

Table 208: **Quad SPI I/O Signals**

Signal Name	PMC MIO Pin	MIO-at-a-Glance Table	Device Interface Options				
			Single		Dual-Stacked		Dual-Parallel
			Lower Only	Upper Only	Lower for both devices	Upper for both devices	8-bit data
			A	B	D	E	E
Lower Interface							
QSPI0_CLK	0	0	CLK	~	CLK	~	CLK
QSPI0_CS_b	5	5	CS_b	~	CS_b	CS_b	CS_b
QSPI0_IO[0]	4	4	IO[0]	~	IO[0]	~	IO[0]
QSPI0_IO[1]	1	1	IO[1]	~	IO[1]	~	IO[1]

Table 208: **Quad SPI I/O Signals** *(cont'd)*

Signal Name	PMC MIO Pin	MIO-at-a-Glance Table	Device Interface Options				
			Single		Dual-Stacked		Dual-Parallel
			Lower Only	Upper Only	Lower for both devices	Upper for both devices	8-bit data
			A	B	D	E	E
QSPI0_IO[2]	2	2	IO[2]	~	IO[2]	~	IO[2]
QSPI0_IO[3]	3	3	IO[3]	~	IO[3]	~	IO[3]
Upper Interface							
QSPI1_CLK	12	12	~	CLK	~	CLK	CLK
QSPI1_CS_b	7	7	~	CS_b	CS_b	CS_b	CS_b
QSPI1_IO[0]	8	8	~	IO[0]	~	IO[0]	IO[4]
QSPI1_IO[1]	9	9	~	IO[1]	~	IO[1]	IO[5]
QSPI1_IO[2]	10	10	~	IO[2]	~	IO[2]	IO[6]
QSPI1_IO[3]	11	11	~	IO[3]	~	IO[3]	IO[7]
Loopback Clock Output							
QSPI_LPBK_CLK	6	6	For clock frequencies >37.5 MHz				

SD/eMMC Controller

The two SD/eMMC controllers have the same features and are operated independently. The controller communicates with SDIO devices, SD memory cards, and eMMC cards and devices with up to eight data signals.

The controller includes an AXI slave for its programming interface and an I/O data port. The controller also includes an AXI master interface for the controller's DMA.

The SD/eMMC I/O interface is routed through the PMC MIO or the EMIO. The I/O interface is not available through the LPD MIO.

Compatibility

The controller is compatible with the following specifications:

- SD host controller standard specification version 3.00
- SD memory card specification version 3.01
- SD memory card security specification version 1.01
- SDIO card specification version 2.0, 3.0
- eMMC specification version 4.51

Boot Device

The controller has three SD boot modes and one eMMC boot mode. Boot mode pins [3:0]:

- `0110`: eMMC1 with 8-bit 1.8V interface
- `0011`: SD0 using 3.0 protocol with 4-bit interface
- `0101`: SD1 using 2.0 protocol with 4-bit interface
- `1110`: SD1 using 3.0 protocol with 4-bit interface

For more information, see Boot Modes.

Features

The controller key features are listed in the following sections.

SD/SDIO Mode

- 1- and 4-bit data
- Operating mode maximum clock rates
  - Standard, default speed mode at 25 MHz
  - High-speed mode at 50 MHz
  - SDR12 at 25 MHz
  - SDR25 at 50 MHz
  - SDR50 at 100 MHz
  - SDR104 at 200 MHz, up to 800 Mb/s data rate
  - DDR50 mode at 50 MHz
- Variable-length data transfers
- Cyclic redundancy check CRC7 for command and CRC16 for data integrity
- Performs read wait control, suspend/resume operation SDIO card
- Card detection (insertion/removal) and write protect input signals
- Designed to work with I/O cards, read-only cards, and read/write cards
- Read wait control, suspend/resume operation
- Control signals for external voltage level shifter

eMMC Mode

The eMMC I/O interface includes data widths up to 8 bits with a clock frequency of up to 200 MHz.

- 1-bit, 4-bit, and 8-bit data
- Operating mode with maximum clock rate:
  - Legacy MMC speed mode at 25 MHz
  - High-speed SDR and DDR modes at 50 MHz
  - HS200 mode at 200 MHz for up to 1600 Mb/s data rate
- Cyclic redundancy check CRC7 for command and CRC16 for data integrity.

Primary Boot Device

The SD/eMMC controllers can be used as a boot device in both SD and eMMC modes.

Power Domain

The SD/eMMC controllers are in the PMC power domain.

Comparison to Previous Generation Xilinx Devices

The SD/eMMC controller is similar in the Zynq® UltraScale+™ MPSoC devices.

Improvements and changes:

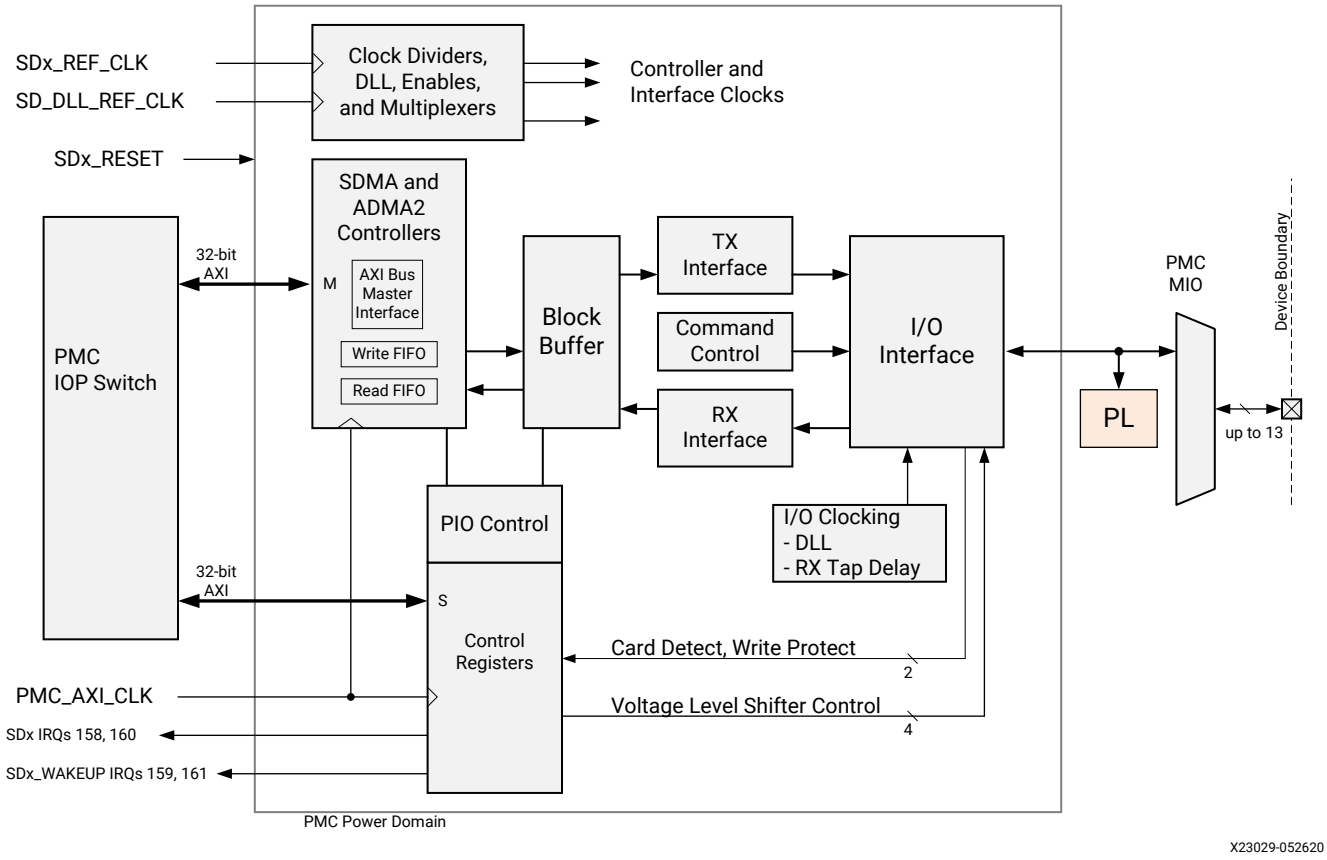
- Enhanced DLL with new programming model
- DLL is used for all frequencies above 25 MHz
- Separate SD 0 and 1 register sets for the DLL TAP delays
- Maximum frequency with external level shifter bumped from 19 to 20 MHz
- Tuning count default value changed from 32 to 40
- SD_REF_CLK divider set = 0 results in a divide by 1

System Perspective

Block Diagram

The SD/eMMC block diagram is shown in the following figure.

Figure 97: **SD/eMMC High-level Block Diagram**



Functional Units

PIO Controller

In PIO mode, the software can access the SD data port register through the AXI slave interface. This is the PIO method in which the host software driver transfers data using the buffer data port register. Only single transfers are allowed (no burst support). Also, only one outstanding read/write transaction is allowed.

PIO SDMA Controller

In SDMA mode, the controller interacts with the registers set and starts the DMA engine for commands with a data transfer. The controller maintains the block transfer counts for PIO operation.

The programming model is explained in SDMA Programming Model.

Descriptor ADMA2 Mode

ADMA2 includes a descriptor-based architecture with scatter-gather capabilities. Software creates descriptor tables in system memory that are processed in the ADMA2 mode.

The programming model is explained in ADMA2 Programming Model.

ADMA2 Controller

The DMA controller supports both SDMA and ADMA2 modes. The DMA controller uses the master AXI interface to transfer data between the block buffer and the system memory. The controller also uses this interface to access descriptor tables in system memory. The DMA controller also implements a host transaction generator to control the host master interface.

The DMA memory transactions can be routed to the FPD CCI for cache coherency with the APU or a non-coherent path including a NoC port to access system memory or AXI routing to the OCM.

System Interfaces

The host controller interfaces to the system bus using the AXI master and slave interface.

AXI Slave Interface for Programmed I/O

The AXI slave programming interface provides software with access to the read/write memory mapped registers for control and status. It also provides port accesses for programmed I/O commands, reads, and writes. All accesses are single 32-bit read/write transactions.

AXI Master Interface for DMA Transfers

The 32-bit AXI master is used by the SDMA for simple programmed I/O access and the ADMA2 for autonomous read and write memory transactions using descriptor tables with scatter-gather capabilities.

System Signals

System signals connected to the SD/eMMC controller include:

- Clocks
- Controller Reset
- System Interrupts
- System Errors

Clocks

There are three clock from the system:

- SDx_REF_CLK for each controller
- SD_DLL_REF_CLK driving a DLL in each controller
- AXI bus interface clock

The SD clocks are described in Clock Functionality.

Controller Reset

The controller can be reset from the PMC reset controller or by writing to the controller's software reset register. The PMC reset affects the entire controller and sets all registers to their reset default state.

The attached eMMC card can be reset using the powercontrol [emmc_hwreset] register bit.

System Interrupts

Each controller generates two system interrupts. The IRQ numbers refer to controllers 0 and 1, respectively.

- Wake-up interrupt (IRQ# 158 and 160)
- Controller interrupt managed by three sets of register controls (IRQ# 159 and 161)
  - Normal interrupts, see the SDIO.normalintrsts register
  - DMA interrupts, see the SDIO.admaerrsts register
  - Error interrupts, see the SDIO.errorintrsts register

The enabled controller interrupts are OR'd together and assert the SD/eMMCx system interrupt. The wake-up interrupt is separate from the controller interrupts. All system interrupts are listed in IRQ System Interrupts.

System Errors

The APB programming interface generates an address decode error if it detects an access violation. The system errors are listed in PMC Error Status Accumulator Registers.

I/O Interface

The controller provides I/O signals for SDIO and eMMC interfacing. These interface signals are routed to the PMC MIO pins.

Configurations

- SD and SDIO
- eMMC

MIO Interface

Each I/O interface is routed separately through the PMC MIO or the PL EMIO. The interface is not available through the LPD MIO. The SD I/O interface signals includes 1 and 4-bit data with card detect, and write protect. The interface also includes signals to control an optional external voltage level shifter for interfacing to the devices at 3.3V and switching to 1.8V for higher speed, SD 3.0 functionality.

The I/O interface signals are listed in SD I/O Signals.

I/O Wiring Diagrams

The I/O wiring connections for boot modes are shown in SD Boot Modes.

Modes and States

Speed Modes

The SD card speed modes are listed in the following table.

Table 209: **SD Card Speed Modes**

Speed Mode[1]	Data Rate	Clock Edge	I/O Width	Frequency (MHz)	Clock Source	Max. MB/s	SD Card Voltage	MIO/EMIO
Default speed[2]	Single	Falling	1, 4	25	DIV_CLK	12.5[3]	3.3V	MIO, EMIO
High speed	Single	Rising	1, 4	50	DLL Clock	25[4]	3.3V	MIO
SDR-12	Single	Rising	4	25	DIV_CLK	12.5	1.8V	MIO, EMIO
SDR-25	Single	Rising	4	50	DLL Clock	25	1.8V	MIO
DDR-50	Double	Both	4	50	DLL Clock	50	1.8V	MIO
SDR-50	Single	Rising	4	100	DLL Clock	50	1.8V	MIO
SDR-104	Single	Rising	4	200	DLL Clock	100	1.8V	MIO

Notes:
1. SD line selection is based on SD 2.0 or 3.0 mode.
2. When using an external voltage level shifter, the maximum frequency is 20 MHz.
3. Throughput is reduced to 3.125 MB/s in 1-bit mode.
4. Throughput is reduced to 6.25 MB/s in 1-bit mode.

The MMC and eMMC speed modes are listed in the following table.

Table 210: **MMC and eMMC Speed Modes**

Speed Mode	Data Rate	Clock Edge	I/O Width	Frequency (MHz)	Clock Source	Max. MB/s [3]	SD Card Interface Voltage	MIO/EMIO
Legacy MMC speed[1], [2]	Single	Falling	1, 4, 8	25	DIV_CLK	25	1.8, 3.3V	MIO, EMIO
HS-SDR	Single	Rising	4, 8	50	DLL Clock	50	1.8, 3.3V	MIO
HS-DDR	Double	Both	4, 8	50	DLL Clock	100	1.8, 3.3V	MIO
HS-200	Single	Rising	4, 8	200	DLL Clock	200	1.8V	MIO

Notes:

1. Legacy MMC speed relates to default MMC speed.
2. The default eMMC boots in legacy MMC speed mode only. Software driver can switch to the high-speed modes for higher throughput.
3. Throughput is based on an 8-bit I/O interface width.

States

The controller has several states:

- Reset
- Configuration
- Normal
- Sleep

Main Functionality

The main functional units include:

- Command Controller
- Transmit Control Unit
- Receive Control Unit
- Timeout Control
- Data Transfer Block Buffer

The I/O interface units are described in I/O Functionality.

Command Controller

The SD command control generates the command sequence on the CMD line of the SD interface for every new command programmed by the software. The command control controller also implements the response reception and checking the validity of the response. It uses the response type field to determine the length of the response and the presence of the CRC7 field. The response is received on the receive clock, which is either the looped back clock or the tuned clock. After the response is received, the contents of the response (start bit, command index, CRC7, end bit) are verified and the response status is written to registers, setting various status bits. The controller also implements a timeout check on the response reception to make sure that the response is received within the defined time (5 or 64 clocks based on command type). The received response is stored into the appropriate bit position in the response register. The SD command controller generates controls to the SD transmit control and SD receive control based on the transfer direction. The SD command controller also generates an auto command (AutoCMD12 or AutoCMD23) when enabled.

Transmit Control Unit

The transmit control unit is used for writing transfers to transfer data to the card. After the command is issued, the controller waits for a block of data to be available in the block buffer and transfers the data onto the SD DAT lines. Based on the configuration of data lines (1-bit, 4-bit, or 8-bit), the data from the block buffer is appropriately routed. The CRC16 is individually calculated on a per-lane basis and is attached at the end of block transfer before the END bit. In DDR operation, the transmit control unit implements a separate CRC16 for each edge of the clock. At the end of block transfer, it waits for the CRC response on the DAT0 line and reports the result of the CRC check to the register set. The controller also checks for a write busy indication (DAT0 line) before transferring the next block of data. A timeout check is implemented to ensure that the write busy is asserted no more than the required limit.

Receive Control Unit

The receive control unit is used for read transfers for receiving data from the card. After the command is issued, the controller waits for the block of data to be received from the card. Based on the configuration of data lines (1-bit, 4-bit, or 8-bit), the data from the SD interface is assembled into bytes and eventually into a 32-bit word before it is written into the block buffer. The CRC16 is individually calculated on a per-lane basis and is checked against the received CRC16 at the end of block transfer before the END bit. In DDR operation, the receive control unit implements a separate CRC16 checker for each edge of the clock. The data is received on the receive clock. This receive clock is either the looped back clock (SD_CLK from the IO_BUF) or the tuned clock using delayed-lock loop (DLL) or delay (DLY) elements. A timeout check is implemented to ensure that the gap between the block is no larger than the required limit.

Timeout Control

The SD timeout control unit implements the timeout check between block transfers. It uses the contents of the timeout control register to implement timeout between blocks.

The timeout control operates under the control of the transmit control and receive control units (based on direction). When a timeout is detected, the event is reported to the transmit control or receive control units.

Data Transfer Block Buffer

The transfer buffer is dual-ported between the DMA units attached to the AXI and read and write I/O interfaces. Transfers are broken down into a data block size. The minimum size is 512 bytes and the maximum is 2 KB. For maximum performance, the buffer must be twice the maximum block size being transferred to enable pipelining.

During a write transaction from the system memory to the TX interface, data is stored in the transfer buffer. When a block of data is written into the buffer (done), the TX interface then sends it out onto the I/O interface. The DMA controller can continue to fetch additional blocks of data if the transfer buffer has space. During a read transaction from the RX interface to system memory, data is stored in the transfer buffer.

The data stored from the RX interface is not committed until the CRC checking is performed. When an RX block of data is available in the transfer buffer, the SDMA or ADMA2 transfers this data to system memory via the AXI master interface. Meanwhile, the RX interface can receive the next block of data, provided there is space available in the transfer buffer (the block memory size is less than half the available transfer buffer memory size for RX transactions).

- Issue a read wait command to the I/O interface (if supported by the external device)
- Stop the SDx_CLK signal

I/O Functionality

The SD interface controller maps the internal signals to the external SD interface and vice versa. Based on the bus width (1, 4, or 8) the internal signals are driven out appropriately.

In the case of a default speed (DS) mode, the outputs are driven on the negative edge of the SD_CLK.

The inputs are latched on the RX_CLK (looped back or tuned clock) and output to the receive control unit for further processing.

Card Detect

The controller monitors the SDx_CD_b input signal to detect when a card is inserted or removed.

Debouncing logic is included on this input to filter false transitions. Insertion and removal events that are detected on the card detect signal are posted in the interrupt status register.

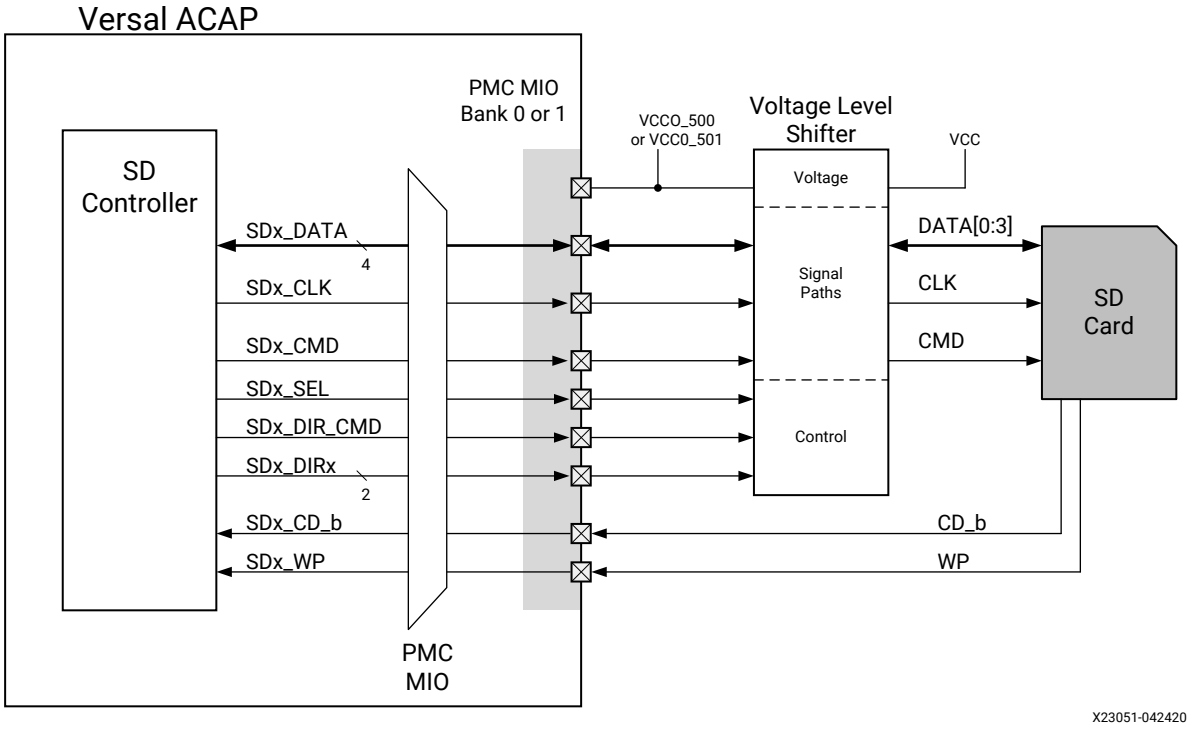
- Card detection uses the SD host control register card detect signal bit as the selection bit
- If the SD control register card detection bit = 1, the card is inserted during boot time or an eMMC
- If the SD control register card detection bit = 0, the SD slot interface is used to identify the insertion and removal of the card using the MIO pin

Voltage Level Shifter Interface

The external voltage level shifter is for interfacing to SD cards. The SD I/O signals are routed through one of the two PMC MIO banks (0 or 1). The I/O voltage for the entire 26 pin MIO bank is from one set of power pins. They are usually supplied with 1.8 or 2.5V. For SD 3.0 boot and other applications, an external voltage level shifter is needed to enable the controller to initially interface at 3.3V at the card and then a lower voltage for high-speed transfers.

The wiring diagram for an SD card connected to a voltage level shifter is shown in the following figure. This example shows the PMC MIO bank 0 and is powered by the VCCO_500 power pins. PMC MIO bank 1 is powered by the VCCO_501 power pins.

Figure 98: **External Voltage Level Shifter Wiring**



Boot Sequence Example

After the boot up, the SEL pin is used to switch from 3.3V to 1.8V to operate at the highest speed modes of the SD cards. The SEL pin is automatically driven by the controller if configured in SD3.0.

The voltage translation function is implemented by an external voltage level translator.

Clock Functionality

The controller supports a wide range of I/O clock frequencies including 400 kHz discovery and the popular 25, 50, 100, and 200 MHz frequencies. The controller always drives the I/O interface clock, SCLK.

The controller receives two reference and one system interface clock from the PMC clock controller:

- DIV_CLK from the SDx_REF_CLK reference clock
- DLL clocks from the SD_DLL_REF_CLK reference clock
- System interfaces clock from PMC_LSBUS_CLK

The SDx_REF_CLK and SD_DLL_REF_CLK clock generators, explained in Reference Clock Frequency Dividers, should be sourced from the same PLL.

I/O Interface Clocks

The controller output clock (SDx_SCLK) has two source clock trees DIV_CLK and DLL clocks. The controller logic is always clocked by DIV_CLK from 10-bit divider and SDx_REF_CLK reference clock. The I/O interface clock source depends on the I/O clock frequency:

- <= 25 MHz uses DIV_CLK
- > 25 MHz uses DLL clocks derived from the SD_DLL_REF_CLK

The clock architecture is shown in the I/O Clock Block Diagram.

DIV_CLK

The DIV_CLK is generated from a simple 10-bit clock divider that is driven by SDx_REF_CLK and programmed by the SDIO.clockcontrol register.

DLL Clocks

The DLL clocks are generated by the DLL that is driven by SD_DLL_REF_CLK and programmed by the registers shown in DLL Clocks Programming Model.

System Interface Clock

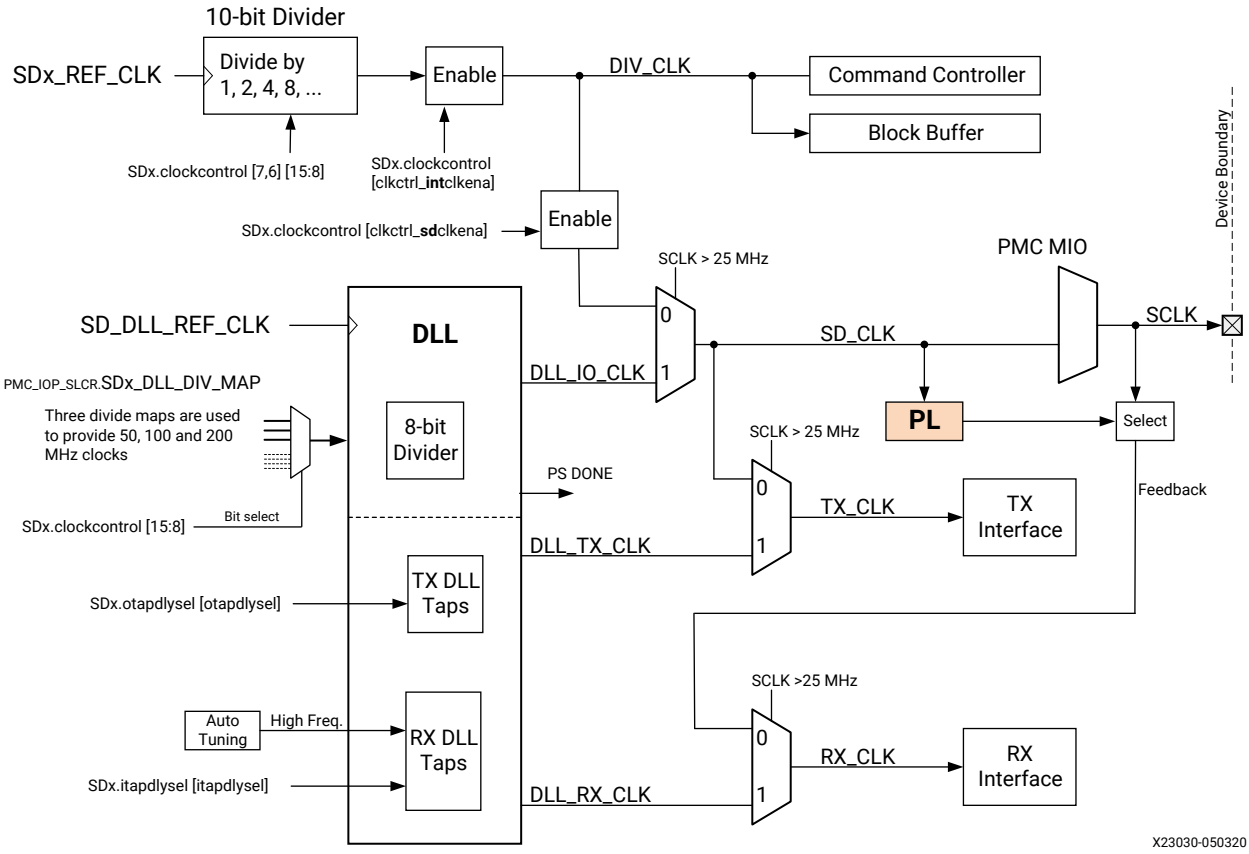
There are two AXI interfaces, they are both clocked by PMC_AXI_CLK from the PMC_IRO_CLK.

- AXI 32-bit slave interface for data and control register programming
- AXI 32-bit master interface for DMA

I/O Clock Block Diagram

The clock routing paths and selection mechanisms are shown in the following figure. There are two controllers (the figure shows one controller). The controllers have separate SDx_REF_CLK clocks, but share a single SD_DLL_REF_CLK.

Figure 99: **SD I/O Clock Block Diagram**



Clock Controls

The registers to control the clock frequency and tap delays are shown in the following table. These controls are also shown in the I/O Clock Block Diagram.

Table 211: **Clock Programming Registers**

SDIO Register	Field	Bits	<= 25 MHz	> 25 MHz	Description and Usage
DIV_CLK Controls					
CLOCKCONTROL	[clkctrl_sdclkfreqsel_upperbits] [clkctrl_sdclkfreqsel]	[7:6] [15:8]	1	1	Clock frequency divider
	[clkctrl_intclkena]	[0]	1	1	Divider or DLL clock and TX output enable
	[clkctrl_sdclkena]	[2]	1	0	Divider or DLL output enable
DLL Clock Controls					
CLOCKCONTROL	[clkctrl_sdclkfreqsel]	[15:8]	~	1	Clock frequency divider

Table 211: **Clock Programming Registers** *(cont'd)*

SDIO Register	Field	Bits	<= 25 MHz	> 25 MHz	Description and Usage
OTAPDLYSEL	[otapdlysel]	[5:0]	~	[2]	Output tap delay select
ITAPDLY	[itapdlysel]	[7:0]	~	[2]	Input tap delay select

Notes:

1. Set these bits based on the required frequency of the SDx_REF_CLK and SDx_CLK. For frequencies above 25 MHz, the SDx_REF_CLK is set to 200 MHz. See DIV_CLK Programming Model.
2. These values are used to manually tune the DLL clock phases RX. See DLL Clocks Programming Model.

DLL Presets

Software can use the DLL presets to automatically switch the DLL output frequency. There are three DLL presets. They are programmed using the SDx_DLL_DIV_MAP0 register. Software programs the SD_DLL_REF_CLK output frequency to 1200 MHz.

Table 212: **SD DLL Div Map Register Examples**

I/O Frequency	Preset	DIV_MAP register		Divide By
		Field	Value	
200 MHz	0	[DIV_0]	0Ch	6
	1	[DIV_1]	18h	12
	2	[DIV_2]	30h	24
100 MHz	0	[DIV_0]	18h	12
	1	[DIV_1]	30h	24
50 MHz	0	[DIV_0]	30h	24
30 MHz	0	[DIV_0]	50h	40

DIV_CLK Programming Model

The clock frequency divider on the SDx_REF_CLK is controlled by the SDIO.clockcontrol [clkctrl_sdclkfreqsel] and [clkctrl_sdclkfreqsel_upperbits] bit fields. The DIV_CLK control signals are shown in table in Clock Controls.

Controller Start-up

During start-up, the controller and I/O operate at 400 kHz using the DIV_CLK. This is accomplished by setting SDx_REF_CLK to 200 MHz and the clock divider fields to 100h (divide by 512). After the software has determined the capabilities of the SD/eMMC device, it reprograms the clock divider to generate the DIV_CLK to match the desired I/O frequency. If this frequency is over 25 MHz, the software needs to program the DLL,the SD_DLL_REF_CLK and configure the controller. After the DLL has locked, the SCLK output and the RX/TX interface clocking switch over to the DLL clock outputs.

DIV_CLK Frequency Table

The divider for DIV_CLK is controlled by the SDIOx.clockcontrol register.

Table 213: **SD/eMMC DIV_CLK Clock Frequency Settings**

SDx_CLK Frequency (MHz)	SDx_REF_CLK Frequency (MHz)	Clock Control Register	
		[clkctrl_ sdclkfreqsel]	Divider Value (decimal)
25.00	200	4	8
20.00		5	10
16.67		6	12
25.00	100	2	4
16.67		3	6
12.50		4	8
10.00		5	10
25.00	50	1	2
12.50		2	4
8.30		3	6
6.25		4	8
5.00		5	10
12.50	25	1	2
6.125		2	4
4.12		3	6
3.12		4	8
2.50		5	10

DLL Clocks Programming Model

Programming Sequence

Select the DLL tap using these steps:

1. Disable SD clock. Write 0 to the SDIO.clockcontrol [clkctrl_sdclkena], bit2.
2. Program new output tap value. Write to the SDIO.otapdlysel [otapdlysel] field.
3. Program new input tap value using the SDIO.itapdly register:
   a. Disable clock output. Write 1 to the [itapchgwin] bit.
   b. Write new input tap value. Write 1 to [itapdlyena] and tap value to [itapdlysel].
   c. Enable clock output. Write 0 to the [itapchgwin] bit.
4. Wait for the SD clock to stabilize. Read SDIO.clockcontrol [sdhcclkgen_intclkstable_dsync] until it is = 1.

5. Enable the SD clock. Write 1 to [clkctrl_sdclkena].

Note: In auto-tune mode, the tuning logic might wait for the SDIO.clockcontrol [sdhcclkgen_intclkstable_dsync, 1] each time before issuing the read tuning command to the SD card.

I/O Clocks

The I/O timing has several modes depending on the interface mode and the clock frequency.

The TX interface has two clock modes:

- AXI and APB Isolation
- TX Clocking > 25 MHz using DLL clocking with 180-tap unit

The RX interface has two clock modes:

- 25 MHz Clocking with SCLK clock looped back
- RX Clocking >25 MHz using DLL clocking with 180-tap unit

SCLK Clock Edge

The SCLK clock edge that is used to drive and latch data. The clock edges are defined by the controller mode. The active clock edge for each mode is listed in the tables in Speed Modes.

RX/TX Tuning Methods

The tuning methods are shown in the following table.

Table 214: **Tuning Methods**

Controller Mode	Speed Mode	Clock Rate (MHz)	Tuning Method
			DLL RX Taps
SD	Default speed	25	~
	High speed	50	Manual
	SDR-12	25	~
	SDR-25	50	Manual
	DDR-50	50	Manual
	SDR-50	100	Auto-tuning
	SDR-104	200	Auto-tuning

Table 214: **Tuning Methods** *(cont'd)*

Controller Mode	Speed Mode	Clock Rate (MHz)	Tuning Method
			DLL RX Taps
eMMC	Legacy MMC	25	~
	HS-SDR	50	Manual
	HS-DDR	50	Manual
	HS-200	200	Auto-tuning

Auto Tuning Note

During auto-tuning, one of the following must occur before sending any command sequence including CMD19, CMD21, or any other command sequence:

- SDx.clockcontrol [sdhcclkgen_intclkstable_dsync] (bit 1) reads = 1
- PMC_IOP_SLCR.SDx_DLL_CTRL [SDx_DLL_PSDONE] reads = 1

The maximum number of tap delays in DLL mode (phases of the clock) is 180, but the useful number of tap delays is reduced as the clock frequency goes up.

25 MHz Clocking

At <= 25 MHz, the RX interface is clocked by a clock signal that is looped back from the SCLK output pad. The SCLK output is driven by the DIV_CLK.

The TX interface is clocked by the TX_CLK, which is multiplexed from the DIV_CLK derived from the 10-bit counter. The clock edge timing is fixed and cannot be adjusted.

The clock structure is shown in I/O Clock Block Diagram. The frequency control is explained in DIV_CLK Programming Model.

TX Clocking > 25 MHz

For clock frequencies greater than 25 MHz, the DLL generates the DLL_IO_CLK for the I/O SCLK output and the DLL_TX_CLK for the TX interface to drive the command and data output signals.

- DLL_IO_CLK to the SCLK clock output pad
- DLL_TX_CLK to the TX interface for clocking-out the command and data output pads

The DLL_TX_CLK does not affect the SCLK output.

The timing of the DLL_TX_CLK relative to the DLL_IO_CLK is adjusted using a 180-tap unit. The TX tap for DLL_TX_CLK is selected by the SDIO.sd0_otapdlysel [otapdlysel] bit field. The clock frequency determines the number of useful taps.

- 200 MHz: 8 taps
- 100 MHz: 15 taps
- 50 MHz: 30 taps
- 33 MHz: 45 taps

Example programming values are shown in DLL Programming Example.

RX Clocking >25 MHz

For clock frequencies greater than 25 MHz, the DLL generates the DLL_IO_CLK for the I/O SCLK output and the DLL_RX_CLK for the RX interface to latch the data input signals. Two separate, asynchronous clocks:

- DLL_IO_CLK to the I/O SCLK clock output pad
- DLL_RX_CLK to the RX interface for latching the data inputs

The DLL_RX_CLK does not affect the SCLK output.

The timing of the DLL_RX_CLK relative to the DLL_IO_CLK is adjusted using a 180-tap unit. The RX tap is selected by the SDIO.sd0_itapdlysel [itapdlysel] bit field. The clock frequency determines the number of useful taps.

- 200 MHz: 30 taps
- 100 MHz: 60 taps
- 50 MHz: 120 taps
- 33 MHz: 180 taps

Example programming values are shown in DLL Programming Example.

DLL Programming Example

The example clock divider and tap settings assume the following:

- SDx_REF_CLK is set to 200 MHz
- SD_DLL_REF_CLK is set to 1200 MHz

Clock Divider

The clock divider is programmed using two fields in the SDIO.clockcontrol register. For the DLL, the reference frequency must be set to 1200 MHz.

- [clkctrl_sdclkfreqsel] shown Clock Controls
- [clkctrl_sdclkfreqsel_upperbits] set = 0

TX DLL Tap Setting

The TX DLL tap settings depend on the controller mode, but are independent of the controller and the MIO path. The TX DLL tap is selected using the SDIO.otapdlysel [otapdlysel] bit field.

RX DLL Tap Setting

The RX DLL tap settings depend on the controller mode, the controller number, the MIO path, and board layout. The RX DLL tap is selected using the SDIO.itapdlysel [itapdlysel] bit field.

The following table shows example settings for manual tuning, which should be useful as a starting point.

Table 215: **SD/eMMC DLL Setting Example**

Controller Mode	Clk_Divider	Frequency (MHz)	RX DLL Tap Value				TX DLL Tap Setting
			SD/eMMC 0		SD/eMMC 1		
			MIO Bank 0	MIO Bank 1	MIO Bank 0	MIO Bank 1	
SD 50	12	100	14h	13h	13h	14h	03h
SD DDR	24	50	14h	14h	14h	14h	03h
SD HSD	24	50	17h	17h	17h	17h	04h
eMMC DDR	24	50	14h	14h	14h	14h	05h
eMMC HSD	24	50	17h	17h	17h	17h	05h

RX Tap Programming Note

To avoid clock glitches from propagating to the external device, shut off the clock while programming the RX tap unit. Use the SDIO.ITAPDLY [itapchgwin] bit to gate the clock:

- Turn off the clock and set [itapchgwin] bit = 1
- Program the RX tap value
- Turn on the clock and set [itapchgwin] bit = 0

SD and eMMC Commands

The registers to generate SD commands are listed in the following table.

Table 216: **SD Commands**

Register	SDMA Command	ADMA2 Command	CPU Data Transfer	Non DAT Transfer
SDMA system address, argument 2	Yes/No	No/Auto CMD23	No/Auto CMD23	No/No

Table 216: **SD Commands** *(cont'd)*

Register	SDMA Command	ADMA2 Command	CPU Data Transfer	Non DAT Transfer
Block size	Yes	Yes	Yes	No (protected)
Block count	Yes	Yes	Yes	No (protected)
Argument 2	Yes	Yes	Yes	No (protected)
Transfer mode	Yes	Yes	Yes	No (protected)
Command	Yes	Yes	Yes	Yes

The table shows register settings for three transactions: SDMA generated transactions, ADMA2 generated transactions, and CPU data transfers and non-DAT transfers. When initiating transactions, the host driver programs these registers sequentially from `000h` to `00Fh`. The beginning register offset is calculated based on the type of transaction. The last written offset is always `00Fh` because writing to the upper byte of the command register triggers the issuance of the SD command.

The command number is selected using command [cmdindex].

Table 217: **SD Controller Commands**

Command	Description	Response	Related Registers
CMD17	Single block read		blocksize [xfer_blocksize]
CMD18	Multi-block read		
CMD24	Single block write		
CMD25	Multi-block write		
CMD38			
CMD52			command_datapresent
CMD53	I/O read/write extended		
CMD55			
Auto CMD6			
Auto CMD12			xfermode_autocmdena [errorintrsts_autocmderror]
Auto CMD23			sdmasysaddrlo [sdma_sysaddress] xfermode_autocmdena [errorintrsts_autocmderror]
Auto CMD41			
Auto CMD42			
Auto CMD51			

PIO Data Port Programming Model

Software can read and write data to and from the transfer buffer using the register data ports SD.reg_dataport using 32-bit read/write transactions.

SDMA Programming Model

In SDMA mode, the controller interacts with the registers set and starts the DMA engine for commands with a data transfer. The controller maintains the block transfer counts for PIO operation.

The controller interacts with the registers set and starts the DMA engine when a command with data transfer is involved. The DMA controller interfaces to the host (AXI) master interface to generate memory transfers. The DMA controller also interfaces with the block buffer to store/fetch block data.

SDMA is used for programmed I/O mode. The SDMA maintains the block transfer counts for PIO operations.

The DMA memory transactions can be routed to the FPD CCI for cache coherency with the APU L2 cache or a non-coherent path including a NoC port to access system memory or AXI routing to the OCM. Software selects the AXI transaction using PMC_IOP_SLCR.SDx_IOP_INTERCONNECT_ROUTE [SDx].

ADMA2 Programming Model

ADMA2 includes a descriptor-based architecture with scatter-gather capabilties. Software creates descriptor tables in system memory that are processed in the ADMA2 mode.

Software Routines

- SD configuration
- SD clock frequency change
- SD card initialize
- SD CMD transfer

- SD set block size
- Setup ADMA2 descriptor table
- SD read polled
- SD write polled
- SD select card
- eMMC card initialize
- SD get bus width
- SD change bus width
- SD change bus speed
- SD change clock frequency
- SD send pull-up command
- Get eMMC EXT CSD
- Resetting the DLL
- Manual tuning

Register Reference

The register set implements the SD host controller specification (version 3.00). The host controller register set also implements the data port registers for the programmed I/O (PIO) mode transfers.

The register set provides the control signals to the rest of the controller, monitors the status signals to set the interrupt status bits, and eventually generates interrupt signal.

The registers are programmed by the software through the AXI slave interface. Interrupt status and control registers detect events and monitor system state to generate system interrupts. Each controller can generate a wake-up interrupt or an OR of several interrupts in the interrupt status register.

The SD/eMMC controller registers are in the SDIO register sets:

- SD0 base address is `0xF104_0000`
- SD1 base address is `0xF105_0000`

The DLL, timing, and system-related configuration registers are in the PMC_IOP_SLCR set:

- Base address is `0xF106_0000`

These registers are accessed with single 32-bit read/write transactions to the APB programming interface. The registers are summarized in the SD/eMMC controller register overview and SD/eMMC PMC_SLCR register overview tables in the following sections.

SDIO Registers

The following table lists the SD/eMMC controller registers in the SDIO register set.

Table 218: **SD/eMMC Controller Registers**

Register Name	Offset Address	Type	Description
Generate SD Card Commands			
SDMASYSADDRLO SDMASYSADDRHI	0x000, 0x002	RW	System address for DMA transfers and second argument in auto CMD23
BLOCKSIZE	0x004	RW	Number of bytes in a data block
BLOCKCOUNT	0x006	RW	Number of data block
ARGUMENT1LO ARGUMENT1HI	0x008 0x00A	RW	32-bit command argument
TRANSFERMODE	0x00C	RW	Data transfers
COMMAND	0x00E	RW	Controller commands
SD Card Response			
RESPONSE0 (REGS 0:7)	0x010	R	SD card responses
Data Port			
DATAPORT	0x020	RW	Software access port to transfer buffer
Configuration and Control			
PRESENTSTATE	0x024	R	Controller status
HOSTCONTROL1	0x028	RW	DMA modes, LED control, data-transfer width, high-speed enable, card detect test level, and signal selection
POWERCONTROL	0x029	RW	Control bus power and voltage level
BLOCKGAPCONTROL	0x02A	Mixed	Control lock gap request, read wait control, and block gap interrupt
WAKEUPCONTROL	0x02B	RW	Program wake-up
CLOCKCONTROL	0x02C	Mixed	Clock frequency select, generator select, clock enable, and internal clock state
TIMEOUTCONTROL	0x02E	RW	Data timeout counter
SOFTWARERESET	0x02F	Clear on Write CLRONWR	Software reset for data, command, and for all
Normal and Error Interrupts			
NORMALINTRSTS NORMALINTRSTSENA NORMALINTRSIGENA	0x030 0x034 0x038	WTC, R RW, R RW, R	Normal interrupts: status, enable, interrupt signal

Table 218: **SD/eMMC Controller Registers** *(cont'd)*

Register Name	Offset Address	Type	Description
ERRORINTRSTS ERRORINTRSTSENA ERRORINTRSIGENA	0x032 0x036 0x03A	WTC RW RW, R	Normal errors: status, enable, interrupt signal
AUTOCMDERRSTS	0x03C	R	CMD12 response error for auto CMD12 and CMD23 error for auto CMD 23
HOSTCONTROL2	0x03E	Mixed	UHS select mode, execute tuning, sampling clock select, asynchronous interrupt enable, and preset value enable
Controller Capabilities			
CAPABILITIES	0x040	R	Controller implementation
MAXCURRENTCAP	0x048	R	Maximum current capability per voltage
Force Event			
FORCEEVENTFORAUTOCMDERRORSTATUS FORCEEVENTFORERRINTSTS	0x050 0x052	W R, W	Ports to write to the Auto CMD error status register and the error interrupt status registers
ADMA2			
ADMAERRSTS	0x054	R	ADMA2 state
ADMASYSADDR0, ADMASYSADDR1 ADMASYSADDR2, ADMASYSADDR3	0x058, 0x05A 0x05C, 0x05E	RW	Starting system address for the ADMA2 transfer by the AXI master
Preset Values			
PRESETVALUE0, PRESETVALUE1 PRESETVALUE2, PRESETVALUE3 PRESETVALUE4, PRESETVALUE5 PRESETVALUE6, PRESETVALUE7	0x060, 0x062 0x064, 0x0660x068, 0x06A 0x06C, 0x06E	R	SD_CLK freq select, clock generator, drive strength: Initialization, Default speed High-speed, SDR12 SDR25, SDR50 SDR104, DDR50
Miscellaneous			
BOOTTIMEOUTCNT	0x070	RW	Boot timeout value counter
SLOTINTRSTS	0x0FC	R	OR of interrupts from interrupt status and wake-up
HOSTCONTROLLERVER	0x0FE	R	Vendor and specification versions

SLCR Registers

The controllers are further configured by certain registers in the PMC_IOP_SLCR register set (example, PMC_IOP_SLCR.SD0_CLK_CTRL).

Table 219: **SDIO PMC SLCR Register Overview**

Register Name	Offset Address		Access Type	Description
	SD/eMMC 0	SD/eMMC 1		
Clock and Control				
SDx_CLK_CTRL	0x400	0x480	RW	SD feadback clock routing
SDx_CTRL_REG	0x404	0x484	RW	Controller mode: SD or eMMC
SDx_CONFIG_REG1 SDx_CONFIG_REG2 SDx_CONFIG_REG3	0x410 0x414 0x418	0x490 0x494 0x498	RW	Configuration registers
Presets				
SDx_INITPRESET, SDx_DSPPRESET SDx_HSPDPRESET, SDx_SDR12PRESET SDx_SDR25PRESET, SDx_SDR50PRESET SDx_SDR104PRESET, SDx_DDR50PRESET	0x41C, 0x420 0x424, 0x428 0x42C, 0x430 0x434, 0x438	0x49C, 0x4A0 0x4A4, 0x4A8 0x4AC, 0x4B0 0x4B4, 0x4B8	RW	Initialization for SD Init preset, default speed high speed, SDR12 SDR25, SDR50 SDR104, DDR50
Miscellaneous Registers				
SDx_MAXCUR1P8 SDx_MAXCUR3P0 SDx_MAXCUR3P3	0x43C 0x440 0x444	0x4BC 0x4C0 0x4C4	RW	Maximum current: 1.8, 3.0, and 3.3V
SDx_DLL_CTRL	0x448	0x4C8	Mixed	SD DLL status
SDx_CDn_CTRL	0x44C	0x4CC	RW	SD card detect
SDx_RX_TUNING_SEL	0x454	0x4D4	R	DLL RX clocking
SDx_DLL_DIV_MAP0	0x458	0x4D8	RW	DLL divider mapping.
SDx_IOP_COHERENT_CTRL SDx_IOP_INTERCONNECT_ROUTE SDx_IOP_INTERCONNECT_QOS	0x460 0x464 0x46C	0x4E0 0x4E4 0x4EC	RW	AXI master transaction: coherency, route to CCI, and QoS

System Clock and Reset Registers

The resets and reference clock frequencies for the controllers are controlled by the CRP registers. The base address is 0xF126_0000.

Table 220: **SD/eMMC PMC CRP Registers for SD Overview**

Register Name	Offset Address		Access Type	Description
	SD/eMMC 0	SD/eMMC 1		
SD/eMMC Reference Clock				
SDIOx_REF_CTRL	0x0124	0x0128	RW	Set reference clock frequency. Write protected.
SD/eMMC DLL Reference Clock				

Table 220: **SD/eMMC PMC CRP Registers for SD Overview** *(cont'd)*

Register Name	Offset Address		Access Type	Description
	SD/eMMC 0	SD/eMMC 1		
SD_DLL_REF_CTRL	0x0160		RW	Set DLL reference clock frequency. Write protected.

I/O Signals

SD I/O Signals

The SD controller I/O interfaces are routed to the PMC MIO pins and the EMIO. They are not available on the LPD MIO pins. When the EMIO interface is used, the LPD must be powered up.

The I/O signals are summarized in the following table and shown in MIO-at-a-Glance Tables. The I/O group options must be assigned together. The free options can be assigned to either pin option.

Table 221: **SD Controller MIO Signals**

MIO									EMIO	
Signal Name			I/O	PMC MIO Pin				MIO-at-a-Glance Table	Signal Name	I/O
				SD 0		SD 1				
SD 2.0	SD 3.0	eMMC		A	B	C	D			
A, B, C, D Group Options:										
SD0_CLK SD1_CLK		eMMC0_CLK eMMC1_CLK	O	18	38	0	26	2		
SD0_CMD SD1_CMD		eMMC0_CMD eMMC1_CMD	I/O	23	40	3	29	3		
SD0_DATA[0] SD1_DATA[0]		eMMC0_DATA[0] eMMC1_DATA[0]	I/O	13	41	4	30	4		
SD0_DATA[1] SD1_DATA[1]		eMMC0_DATA[1] eMMC1_DATA[1]	I/O	14	42	5	31	5		
SD0_DATA[2] SD1_DATA[2]		eMMC0_DATA[2] eMMC1_DATA[2]	I/O	15	43	6	32	6		
SD0_DATA[3] SD1_DATA[3]		eMMC0_DATA[3] eMMC1_DATA[3]	I/O	16	44	7	3	7		
~	SD0_SEL SD1_SEL	eMMC0_DATA[4] eMMC1_DATA[4]	I/O	19	45	8	34	8		

Table 221: **SD Controller MIO Signals** *(cont'd)*

MIO									EMIO	
Signal Name			I/O	PMC MIO Pin				MIO-at-a-Glance Table	Signal Name	I/O
				SD 0		SD 1				
SD 2.0	SD 3.0	eMMC		A	B	C	D			
~	SD0_DIR_CMD SD1_DIR_CMD	eMMC0_DATA[5] eMMC1_DATA[5]	I/O	20	46	9	35	9		
~	SD0_DIR0[1] SD1_DIR0	eMMC0_DATA[6] eMMC1_DATA[6]	I/O	21	47	10	36	10		
~	SD0_DIR1[1] SD1_DIR1	eMMC0_DATA[7] eMMC1_DATA[7]	I/O	22	48	11	27	11		
Free Option Signals[2]										
SD0_DETECT[3] SD1_DETECT		~	I	24	39	2	28	1		
SD0_WP SD1_WP		~	I	25	37	1	50	0		
SD0_BUSPWR SD1_BUSPWR		eMMC0_RST eMMC1_RST	O	17	49	12	51	12		

Notes:

1. The DIR0 signal controls the direction of the DATA[0] signal and DIR1 signal controls the direction of the DATA[1:3] signals for the external voltage level shifters.
2. The free option signals are essentially DC and do not necessarily need to be in the same group as the I/O signals.
3. The SDx_DETECT signal is separate from the traditional SDx_DATA[3] signal.

Signaling Protocol

Default Speed Clock Edge

In the case of a default speed (DS) mode, the outputs are driven on the negative edge of the SD_CLK.

Clocks, Resets, and Power

This section includes these chapters:

XILINX

Chapter 77

Clocks

There are many clocks in the Versal™ ACAP for clocking logic and I/O. This chapter describes the clocks that are mainly used by the PMC and PS. Other clocks are described in other documents.

PMC and PS Clocks

The clocks associated with the PMC and PS are described in the following sections:

- The Clock Distribution Diagram shows the major internal clocks for the PMC and PS (LPD and FPD)
- Three PMC Source Clocks originate in the PMC:
  - REF_CLK (clock input pin)
  - PMC_IRO_CLK (internal ring oscillator)
  - RTC (real-time clock)
- Five programmable PLL Clock Generators exist in the PMC, PS, and CPM
- Dozens of programmable Reference Clock Frequency Dividers are used to generate clocks for various blocks in the system

CPM

The clocks for the coherent PCIe® module are described in the *Versal ACAP CPM CCIX Architecture Manual* (AM016).

NoC, AI Engine, and DDR Memory Controller Clocks

The PMC includes four programmable clock dividers with outputs routed to the PL for general purpose usage. The PMC also includes programmable clock divider outputs for the NoC, AI engine, and DDR memory controllers.

PL Clocks

The PL includes its own clock arrays that are programmed when blocks are instantiated. The PL also includes programmable clock modules can be driven by clocks from input pins and other sources.

I/O Transceiver Clocks

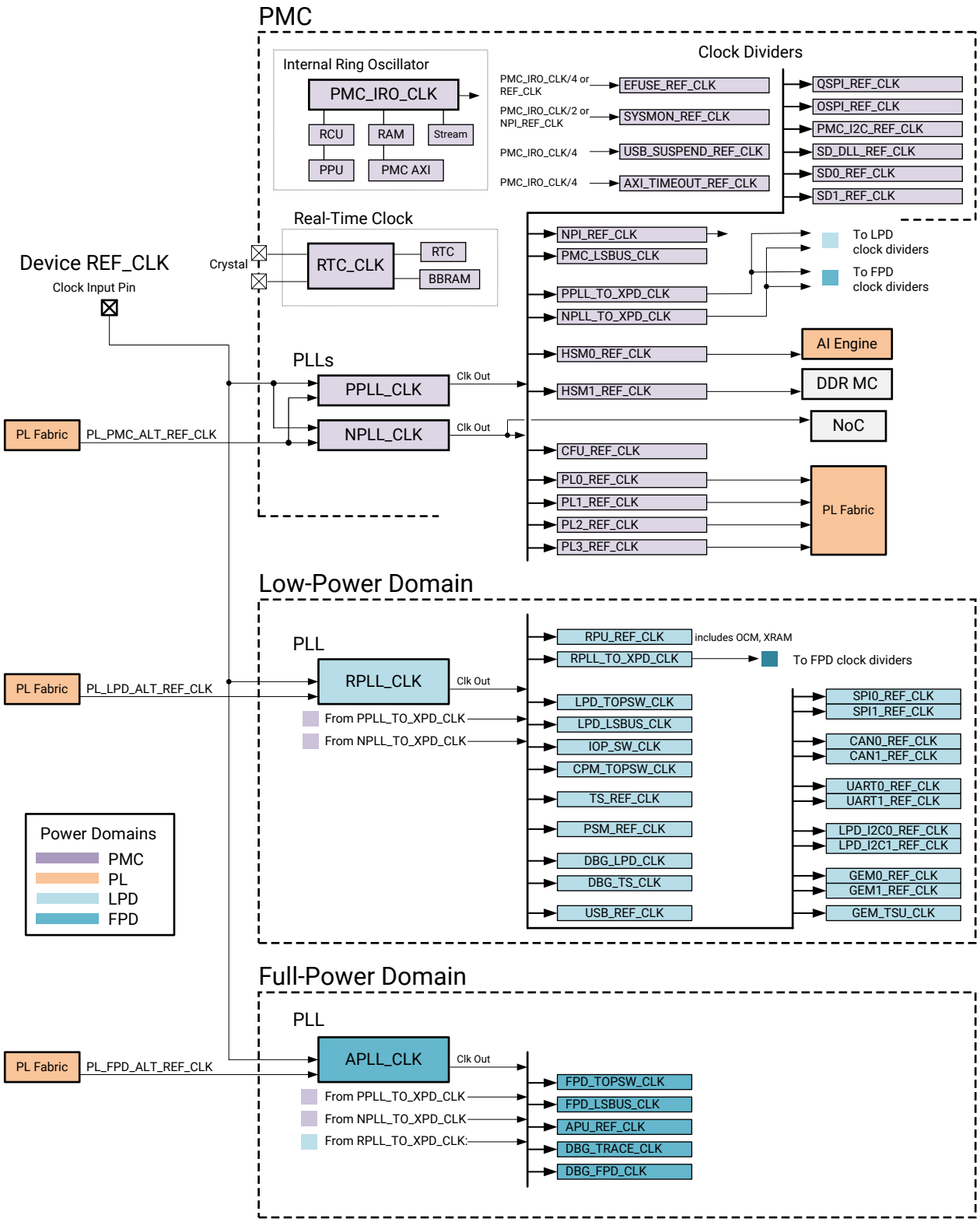
There are local PLLs in the XPIO banks (for the PL, XPHY, and DDRMC) and the gigabit transceivers (GT). These high-speed I/Os use PLL clocks for precision I/O timing. These I/O buffers and transceivers are introduced in the I/O Connectivity chapter of the Section II: Hardware Architecture section. The I/O transceiver clocks are described in their associated documents:

- GTY transceiver PLLs: *Versal ACAP GTY Transceivers Architecture Manual* (AM002)

Clock Distribution Diagram

The PLL and reference clocks are shown in the following figure.

Figure 100: **PMC and PS Clock Distribution Diagram**



X23003-070620

Cross-Domain Clock Routing Consideration

When using clocks from another power domain, consider power management software that can turn off a power domain.

Note: In the Vivado CIPS wizard, the auto-select mode restricts some clocks from being routed across a power domain. For example, the LPD and FPD divider clock outputs are not routed to the PMC power domain because power management software might turn off the FPD or both the LPD and FPD.

The power states are described in Power.

PMC Source Clocks

The PMC has three source clocks:

- REF_CLK device pin input
- PMC_IRO_CLK, an internal ring oscillator (IRO)
- Real-time clock (RTC) driven by an external crystal

REF_CLK Device Pin input

The REF_CLK is typically driven by a 33 MHz external LVCMOS clock signal and is used to drive the five PLL clock generators in the PMC, LPD, FPD, and CPM.

PMC_IRO_CLK Oscillator

The PMC_IRO_CLK is internal to the device and is generated by a self-starting internal ring oscillator (IRO). This clock is used within the PMC for the RCU and PPU processors, their AXI interconnect, and the security module. The IRO frequency is trimmed to the specification defined in the Versal ACAP data sheets.

Crystal-driven RTC

The RTC is driven by an external 32.768 kHz crystal. The clock is consumed within the RTC time keeper. The RTC provides a calibrated time reference based on the attached clock crystal. The clock can be calibrated for greater accuracy. When the device is powered down, the RTC is operated by the battery. The RTC is described in Real-Time Clock.

Summary of Primary Clock Sources and Their Destinations

The primary clock sources are listed in the table.

Table 222: **Primary Clock Sources and Usages**

Clock Name	Usages
REF_CLK device pin	PLL clock generators in PMC, LPD, FPD, and CPM
PMC_IRO_CLK oscillator	PMC-only: processors, interconnect, and security module
RTC_CLK crystal	RTC and battery-backed RAM within the PMC

PLL Clock Generators

The PMC, PS, and CPM PLLs all have similar functionality and programming models.

The five PLLs are:

- PMC:
  - PMC PLL (PPLL)
  - NoC PLL (NPLL)
- PS:
  - LPD PLL (RPLL)
  - FPD PLL (APLL)
- CPM PLL (CPLL)

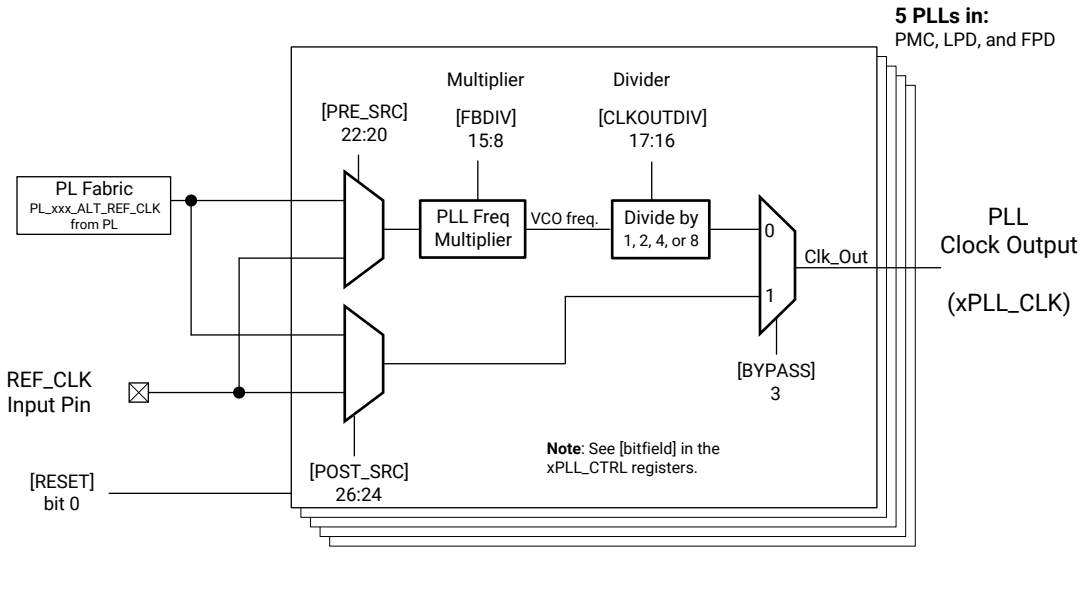
Features

The PLLs have similar features and programming models as previous product generations.

Block Diagram

The architecture of the PLL clock generator is shown in the following figure. There are multiple generators in the system.

Figure 101: **PLL Clock Generator**



Reference Clock Frequency Dividers

There are many clock frequency dividers in the PMC, LPD, FPD, and CPM that provide a reference clock for each block or group of blocks. See the Clock Distribution Diagram for an overview.

The PMC, PS, and CPM clock dividers all have similar programming models. The CRx clock registers select the PLL source clock, define the 10-bit divider value, and enable the divider clock output. The clock divider register sets include:

- PMC clocks: CRP register set
- LPD clocks: CRL register set
- FPD clocks: CRF register set
- CPM clocks: CPM_CRCPM register set

Features

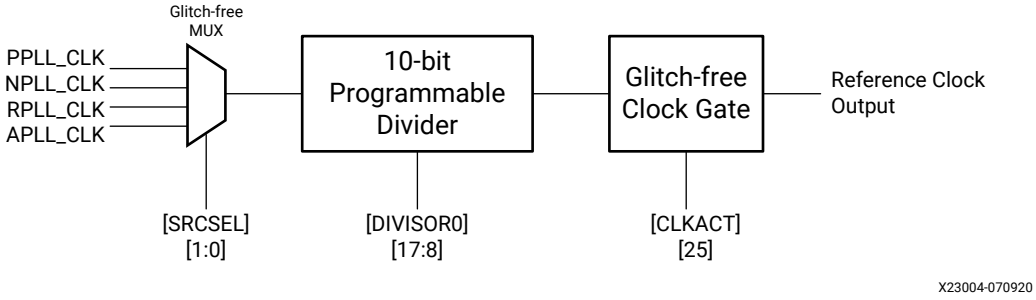
The reference clock generators have a multiplexer to select the source clock, a 10-bit divider, and a glitch-free output enable.

Block Diagram

All clock generators have the same functionality and programming model. The basic clock generator is shown in the following figure.

***Note*:** The PLL source clock choices shown are not always available.

Figure 102: **Clock Generator Block Diagram Example**



Registers

PLL clock generators:

- Clock Generators

Clock dividers:

- PMC Reference Clocks
- LPD Reference Clocks
- FPD Reference Clocks

Clock Generators

The PLL clock generator registers are included in four sets of registers: PMC, LPD, FPD, and CPM. The controller names and register sets are listed in the following table.

Table 223: **PLL Clock Generator Control Registers**

PLL Clock Output Name	Power Domain	Registers			PLL Clock Input Options
		Control Register	Configuration Register	PLL Status Fields	
		Fields: [RESET], [BYPASS], [FBDIV], [CLKOUTDIV], [PRE_SRC], [POST_SRC]	Fields: [RES], [CP], [LFHF], [LOCK_CNT], [LOCK_DLY]	Fields: [xPLL_LOCK], [xPLL_STABLE]	
PPLL_CLK	PMC	CRP.PMCPLL_CTRL	CRP.PMCPLL_CFG	CRP.PLL_STATUS	REF_CLK PL_PMC_ALT_REF_CLK
NPLL_CLK		CRP.NOCPLL_CTRL	CRP.NOCPLL_CFG		

Table 223: **PLL Clock Generator Control Registers** *(cont'd)*

PLL Clock Output Name	Power Domain	Registers			PLL Clock Input Options
		Control Register	Configuration Register	PLL Status Fields	
		Fields: [RESET], [BYPASS], [FBDIV], [CLKOUTDIV], [PRE_SRC], [POST_SRC]	Fields: [RES], [CP], [LFHF], [LOCK_CNT], [LOCK_DLY]	Fields: [xPLL_LOCK], [xPLL_STABLE]	
RPLL_CLK	LPD	CRL.RPLL_CTRL	CRL.RPLL_CFG	CRL.PLL_STATUS	REF_CLK PL_LPD_ALT_REF_CLK
APLL_CLK	FPD	CRF.APLL_CTRL	CRF.APLL_CFG	CRF.PLL_STATUS	REF_CLK PL_FPD_ALT_REF_CLK
CPLL_CLK	PL	CPM_CRCPM.CPLL_CTRL	CPM_CRCPM.CPLL_CFG	CPM_CRCPM.PLL_STATUS	REF_CLK

PMC Reference Clocks

The control registers are used to select an input from a PLL clock generator, or other source, and divide down its frequency. The PMC reference clocks are listed in the following tables. All control registers are in the CRP register set.

Table 224: **PMC IOP Reference Clock Registers**

Reference Clock	Clocks		CRP Control Register
	Output Name	Divider Input Options	
PMC I2C controller	PMC_I2C_REF_CLK	PPLL_CLK, NPLL_CLK	I2C_REF_CTRL
QSPI controller	QSPI_REF_CLK		QSPI_REF_CTRL
OSPI controller	OSPI_REF_CLK		OSPI_REF_CTRL
SD delay-lock loop	SD_DLL_REF_CLK		SD_DLL_REF_CTRL
SD/eMMC 0 controller	SD0_REF_CLK		SDIO0_REF_CTRL
SD/eMMC 1 controller	SD1_REF_CLK		SDIO1_REF_CTRL
USB 2.0 controller located in the LPD	USB_SUSPEND_CLK	PMC_IRO_CLK/4	USB_SUSPEND_CTRL

Table 225: **PMC System Reference Clock Registers**

Reference Clock	Clocks		Control Register
	Output Name	Divider Input Options	
High-speed clock for DDR and AI Engine	HSM0_REF_CLK	PPLL_CLK, NPLL_CLK	HSM0_REF_CTRL
	HSM1_REF_CLK		HSM1_REF_CTRL
General purpose reference clock routed to the PL fabric	PL0_REF_CLK		PMC_PL0_REF_CTRL
	PL1_REF_CLK		PMC_PL1_REF_CTRL
	PL2_REF_CLK		PMC_PL2_REF_CTRL
	PL3_REF_CLK		PMC_PL3_REF_CTRL
Divided-down PPLL_CLK routed the clock controllers in the LPD and FPD power domains	PPLL_TO_XPD_CLK	PPLL_CLK	PPLL_TO_XPD_CTRL

Table 225: **PMC System Reference Clock Registers** *(cont'd)*

Reference Clock	Clocks		Control Register
	Output Name	Divider Input Options	
Divided-down NPLL_CLK routed the clock controllers in the LPD and FPD power domains	NPLL_TO_XPD_CLK	NPLL_CLK	NPLL_TO_XPD_CTRL

Table 226: **PMC Miscellaneous Reference Clock Control Registers**

Description	Clocks		Control Register
	Output Name	Divider Input Options	
APB programming interfaces	PMC_LSBUS_CLK	PPLL_CLK, NPLL_CLK	PMC_LSBUS_REF_CTRL
NPI programming interfaces	NPI_REF_CLK		NPI_REF_CTRL
Configuration frames unit	CFU_REF_CLK		CFU_REF_CTRL
AXI interconnect timeout block	AXI_TIMEOUT_CLK	PMC_IRO_CLK/4	SWITCH_TIMEOUT_CTRL
eFUSE controller	EFUSE_REF_CLK	PMC_IRO_CLK/4, REF_CLK	EFUSE_REF_CTRL
System Monitor (SYSMON)	SYSMON_REF_CLK	PMC_IRO_CLK/2, NPI_REF_CLK	SYSMON_REF_CTRL

LPD Reference Clocks

The LPD reference clocks are listed alphabetically in the following table.

Table 227: **LPD IOP Reference Clock Registers**

Description	Clocks		CRL Control Registers
	Output Name	Divider Input Options	
CAN 0 controller	CAN0_REF_CLK	RPLL_CLK, PPLL_TO_XPD_CLK, NPLL_TO_XPD_CLK	CAN0_REF_CTRL
CAN 1 controller	CAN1_REF_CLK		CAN1_REF_CTRL
GEM 0 controller	GEM0_REF_CLK		GEM0_REF_CTRL
GEM 1 controller	GEM1_REF_CLK		GEM1_REF_CTRL
GEM timestamp clock	GEM_TSU_CLK		GEM_TSU_REF_CTRL
LPD I2C 0 controller	LPD_I2C0_REF_CLK		I2C0_REF_CTRL
LPD I2C 1 controller	LPD_I2C1_REF_CLK		I2C1_REF_CTRL
SPI 0 controller	SPI0_REF_CLK		SPI0_REF_CTRL
SPI 1 controller	SPI1_REF_CLK		SPI1_REF_CTRL
UART 0 controller	UART0_REF_CLK		UART0_REF_CTRL
UART 1 controller	UART1_REF_CLK		UART1_REF_CTRL
USB 2.0 controller	USB_2_REF_CLK		USB_2_REF_CTRL

Table 228: **LPD Miscellaneous Reference Clock Registers**

Description	Clocks		Control Register
	Output Name	Divider Input Options	
			TIMESTAMP_REF_CTRL
CPM AXI interconnect	CPM_TOPSW_CLK	RPLL_CLK PPLL_TO_XPD_CLK NPLL_TO_XPD_CLK	CPM_TOPSW_REF_CTRL
RPU: TCM, GIC, OCM, and Interconnect	CPU_R5F_CLK		CPU_R5F_CTRL
LPD CoreSight™ components except the TSU. Includes: ROM, GPR, CTI, funnel	DBG_LPD_CLK		DBG_LPD_CTRL
CoreSight timestamp generator	DBG_TS_CLK	RPLL_CLK PPLL_TO_XPD_CLK NPLL_TO_XPD_CLK	DBG_TSTMP_CTRL
AXI/AHB interconnect switch	IOP_SW_CLK		IOP_SWITCH_CTRL
LPD APB programming interfaces	LPD_LSBUS_CLK		LPD_LSBUS_CTRL
LPD AXI main switch	LPD_TOPSW_CLK		LPD_TOP_SWITCH_CTRL
PS manager processor	PSM_REF_CLK		PSM_REF_CTRL
Divided-down RPLL_CLK routed to the clock controllers in the FPD power domains	RPLL_TO_XPD_CLK	RPLL_CLK	RPLL_TO_XPD_CTRL

FPD Reference Clocks

FPD Clock Divider Control Registers

The FPD clock dividers are controlled by registers in the CRF register set. The FPD reference clock control registers are listed in the following table.

Table 229: **FPD Reference Clock Control Registers**

Description	Clocks		Control Register
	Output Name	Divider Input Options	
APU: CPUs, L2-cache, debug logic, controls CoreSight components: ELA500, funnel, ETF, CTI	APU_REF_CLK	APLL_CLK RPLL_TO_XPD_CLK PPLL_TO_XPD_CLK NPLL_TO_XPD_CLK	ACPU_CTRL
Trace port for CoreSight debug data flow	DBG_TRACE_CLK		DBG_TRACE_CTRL
FPD CoreSight debug components	DBG_FPD_CLK		DBG_FPD_CTRL
FPD APB programming interfaces	FPD_LSBUS_CLK		FPD_LSBUS_CTRL
FPD AXI main switch	FPD_TOPSW_CLK		FPD_TOP_SWITCH_CTRL

Resets

There are several layers of resets with overlapping effects. The highest-level resets are generally aligned with power domains. Below these are the power island resets, which are then followed by individual functional unit resets. In some cases, functional units have local resets that affect part of the block. The reset hierarchy is as follows:

- Subsystem resets (power domains)
- Power-island resets
- Functional unit (block) resets
- Partial resets of a block (some cases)

Some reset functionality traverses power domains, and, consequently, reset domains. Two examples are CoreSight debug and the USB controller that resides in the LPD with an I/O PHY controller in the PMC.

Resets are triggered by many different sources, including:

- POR_b reset pin (external POR - entire chip)
- Registers controlled by the PMC software (PLM) and PSM firmware (all levels)
- System error events (cause internal POR and system resets)

The PLM and PSM include global registers that are used by system software running in the RPU (or other processor) to request reset services. The PLM and PSM receive the request and work to satisfy the reset request. This is normally done by gracefully bringing the subsystem or functional unit to a stop before applying the reset signal. This functionality is up to the PLM software and PSM firmware implementation.

The cause of a high-level reset is recorded in the CRP.RESET_REASON register.

Comparison to Previous Generation Xilinx Devices

The Versal™ ACAP has a POR_b reset input pin similar to the Zynq® UltraScale+™ MPSoC, but it does not have the system reset pin, PS_SRST_b.

The programming model for the reset controllers are similar to the Zynq UltraScale+ MPSoC.

Persistent Registers

Persistent registers are only cleared by the POR reset, which include:

- PMC_GLOBAL.PMC_BOOT_ERR (BootROM errors)
- CRP.RESET_REASON
- Persistent storage registers (32-bit)
  - PMC_GLOBAL.PERS_GLOB_STORAGE (0 to 4)
  - PMC_LOCAL.PERS_PMC_LCL_STORAGE (0 to 4)
  - PSM_GLOBAL.PERS_GLOB_STORAGE (0 to 7)
  - PSM_LOC_GEN_STORAGE (0 to 3)
- System error status registers
  - PMC_GLOBAL.PMC_ERR1_STATUS
  - PMC_GLOBAL.PMC_ERR2_STATUS
  - PSM_GLOBAL.PSM_ERR1_STATUS
  - PSM_GLOBAL.PSM_ERR2_STATUS

Register Reference

This section organizes the registers associated with reset. It includes reset controls and reset service request registers.

- Reset Reason Register
- Software Reset Service Requests
- Block-Level Software Reset Control
- System-Level Resets

Reset Reason Register

The reset reason register latches the cause of the previous system-level reset. All register bits are read and write 1 to clear (R, W1C). The CRP.RESET_REASON register is located at `0xF126_0220` and protected by the CRP.WPROT register. The RESET_REASON register is only reset by the external POR_b reset pin.

Table 230: **CRP Reset Reason Register**

Bit Field	Description
POR Resets	
[external_por]	The external POR_b reset input was asserted (reset default)
[sw_por]	An internal POR was caused by software
[err_por]	A system error caused an internal POR, see System Errors
System Resets	
[dap_sys]	The JTAG TAP controller initiated a system reset
[err_sys]	A system error initiated system reset, see System Errors.
[sw_sys]	Software initiated system reset

Software Reset Service Requests

Service Requests to PSM

The PSM software-reset trigger register, PSM_GLOBAL.REQ_SWRST_TRIG, is located at `0xFFC9_0420`. The bit assignments and a description of each reset is listed in the following table. These are used to request a reset operation to the PSM firmware.

Table 231: **Software Reset Service Requests to PSM**

Name	Register Bits	Description
APU core 0 APU core 1	0 1	Individually reset an APU core
APU MPCore	4	Reset the whole APU MPCore
RPU	18	Real time processing unit
GEM0 GEM1	20 21	Reset the GEM Ethernet interface
USB 2.0	24	Reset the USB 2.0 interface
LPD IOP	27	Reset the LPD IOP
PS	28	Processing system
LPD	29	Low power domain
FPD	30	Full power domain

Service Requests to PMC

Software requests a reset by setting PMC_GLOBAL and PSM_GLOBAL registers. The software reset request registers are listed in the following table.

Table 232: **Software Reset Request Register Overview**

Register Name	Address	Access Type	Description
PSM_GLOBAL.REQ_SWRST_TRIG Register			
	0xFFC9_0420	W	Subsystems and power islands: APU 0, APU 1, APU MPCore, RPU, GEM 0, GEM 1, USB 2.0, LPD IOP, PS, LPD, FPD. See Service Requests to PSM.
PMC_GLOBAL.REQ_SWRST_TRIG Register			
	0xF111_0420	W	Subsystems: PS, LPD, SoC, PL

Block-Level Software Reset Control

PMC Resets

The PMC blocks are reset by registers in the CRP register set located at 0xF126_0000. The registers are summarized in the following table.

Table 233: **PMC Block Reset Controller Registers**

Register Name	Bit Field	Offset Address	Access Type	Description
RST_QSPI	[RESET]	0x0300	RW	Quad SPI controller
RST_OSPI	[RESET]	0x0304	RW	Octal SPI controller
RST_SDIO0 RST_SDIO1	[RESET]	0x0308 0x030C	RW	SD/eMMC0 controller SD/eMMC1 controller
RST_I2C	[RESET]	0x0314	RW	PMC I2C controller
RST_GPIO	[RESET]	0x0318	RW	PMC GPIO controller
RST_SBI	[RESET]	0x0324	RW	Slave boot interface
RST_PDMA	[RESET0] [RESET1]	0x0328	RW	PMC DMA 0 and 1 resets
RST_PL	[RESET0] [RESET1] [RESET2] [RESET3]	0x0330	RW	General purpose reset signals routed to the PL

Table 233: **PMC Block Reset Controller Registers** *(cont'd)*

Register Name	Bit Field	Offset Address	Access Type	Description
RST_DBG	[RESET]	0x0400	RW	CoreSight™ reset: PMC, LPD, FPD (includes CRL and CRF RST_DBG_LPD regiters) ***Note***: Does not reset the DPC inside of the PMC.
	[DPC]			DPC within the PMC
RST_USB	[PHY_RST]	0x0334	RW	PHY controller in PMC

LPD Resets

The LPD blocks are reset by registers in the CRL register set located at 0xFF5E_0000. The LPD reset registers are summarized in the following table.

Table 234: **LPD Block Reset Controller Registers**

Register Name	Bit Field	Offset Address	Access Type	Description
RST_LPD_DMA	[RESET]	0x0304	RW	LPD DMA unit
RST_GEM0 RST_GEM1	[RESET]	0x0308 0x030C	RW	GEM controllers
RST_USB	[RESET]	0x0314	RW	USB 2.0, also see CRP.RST_USB register for PHY controller in PMC
RST_UART0 RST_UART1	[RESET]	0x0318 0x031C	RW	UART SBSA controllers
RST_SPI0 RST_SPI1	[RESET]	0x0320 0x0324	RW	SPI controllers
RST_CAN0 RST_CAN1	[RESET]	0x0328 0x032C	RW	CAN controllers
RST_LPD_I2C0 RST_LPD_I2C1	[RESET]	0x0330 0x0334	RW	LPD I2C controllers
RST_DBG_LPD	[RESET]	0x0338	RW	LPD CoreSight debug logic in LPD and FPD
	[RESET_HSDP]			High-speed data port, PCIe DMA
	[RPU_DBG0_RESET] [RPU_DBG1_RESET]			Debug logic resets
RST_GPIO	[RESET]	0x033C	RW	LPD GPIO controller
RST_TTC	[TTC0_RESET] [TTC1_RESET] [TTC2_RESET] [TTC3_RESET]	0x0344	RW	Triple timer counter timers
RST_TIMESTAMP	[RESET]	0x0348	RW	Timestamp

Table 234: **LPD Block Reset Controller Registers** *(cont'd)*

Register Name	Bit Field	Offset Address	Access Type	Description
RST_LPD_SWDT	[RESET]	0x034C	RW	LPD watchdog timer
RST_OCM	[RESET]	0x0350	RW	OCM memory
RST_IPI	[RESET]	0x0354	RW	Inter-processor interrupt controller
PSM_RST_MODE	[rst_mode]	0x0370	RW	PSM processor mode after a reset
	[wakeup]			PSM processor wake-up

FPD Resets

The FPD blocks are reset by registers in the CRF register set located at 0xFD1A_0000. The FPD reset registers are summarized in the following table.

Table 235: **FPD Block Reset Controller Registers**

Register Name	Bit Field	Offset Address	Access Type	Description
RST_APU	[APU0]	0x0300	RW	APU0 core
	[APU1]			APU1 core
	[APU_GIC]			APU GIC interrupt controller
	[APU_L2]			APU L2 cache
	[APU0_PWRON]			APU0 power status register
	[APU1_PWRON]			APU1 power status register
RST_DBG_FPD	[RESET]	0x030C	RW	LPD CoreSight debug logic
RST_FPD_SRST	[RESET]	0x0314	RW	System reset

System-Level Resets

The system-level resets include the PORs and resets of the subsystems:

- Entire device
- PMC
- PS
- PL
- SoC

POR Resets

The reset registers in the PMC and LPD control internal POR resets.

Table 236: **Internal POR Reset Control Registers**

Register Name	Bit Field	Offset Address	Description	Reset Reason
CRP Registers (PMC)				
CRP.RST_PS	[PMC_POR] [PS_POR] [PL_POR]	0x031C	Entire chip internal POR PS internal POR PL internal POR	[sw_por] ~ ~
CRP.RST_NONPS	[NOC_RESET] [NOC_POR] [NPI_RESET]	0x0320	NoC, not NPI SPD (NoC, NPI, DDRMC) NPI	~ ~ ~
CRL Registers (LPD)				
LPD.RST_FPD	[POR]	0x0360	FPD internal POR reset	~

System Reset Control

The CRP.RST_PS and CRP.RST_NONPS registers control the system resets.

Table 237: **Software Reset Control Registers**

Register Name	Bit Field	Access Type	Description	Reset Reason
RST_PS	[PMC_SRST] [PS_SRST] [PL_SRST]	RW	Entire chip system reset PS system reset PL system reset	[sw_sys] ~ ~
RST_NONPS	[SYS_RST_1] [SYS_RST_2] [SYS_RST_3]	RW		~ ~ ~

Power

The device power architecture includes power domains and PS-based power islands. The power domains are large areas of the device that have their own set of power pins. The power islands are smaller areas with in the LPD and FPD power domains. The power island are controlled by on-board power FETs. These FETs are controlled by register bits. The power domains and islands are shown in Power Diagram.

The power domain states and the transitions from one state to another have some restrictions. For the power domains and the power islands, the interconnect traffic must be brought to a halt before power-down. The PS power islands are controlled by the PSM controller in the LPD. The power-up process includes sequencing of power, clocks, and resets.

Power management is described in Power Management.

Power Domains

- PMC power domain: platform management controller and functional units
- LPD (low-power domain): real-time processing unit and functional units
- FPD (full-power domain): application processing unit and functional units
- PL power domain
  - PL building blocks and clock structures; the count is device dependent
  - CCIX PCIe module (CPM), if present
  - AI Engine, if present
- SPD (system power domain)
  - NoC and NPI interconnect
  - DDR memory controllers
- BPD (battery power domain)
  - Real-time clock (RTC)
  - Battery-backed RAM (BBRAM)
- Gigabit transceivers for I/O (XPIO, GTM, GTY)

Power Domain States

The PMC power is always required. General rules include:

- PMC is required to operate the device
- LPD is required for the FPD
- SPD is required for the PL
- PL and LPD are required for CPM; FPD is assumed active

PL and PS-centric modes:

- PMC and PL
- PMC and LPD
- PMC, LPD, FPD
- PMC, LPD, FPD, and PL

***Note*:** The PL power domain includes the AI Engine and CPM, if they are present.

There are restrictions regarding EMIO signals.

- LPD is required for PMC EMIO signals

Common power modes are listed in Power Modes.

Power Islands

The LPD and FPD processors and some subsystem units are on their own power islands. These are controlled by the PSM.

- RPU processor (all cores together)
- APU cores (individually)
- APU L2-cache
- 4 MB XRAM supports a total of 16 power islands (1 per 256 KB sub-bank)

The power island controls and service request registers are listed in Power Islands.

Power Reduction Features

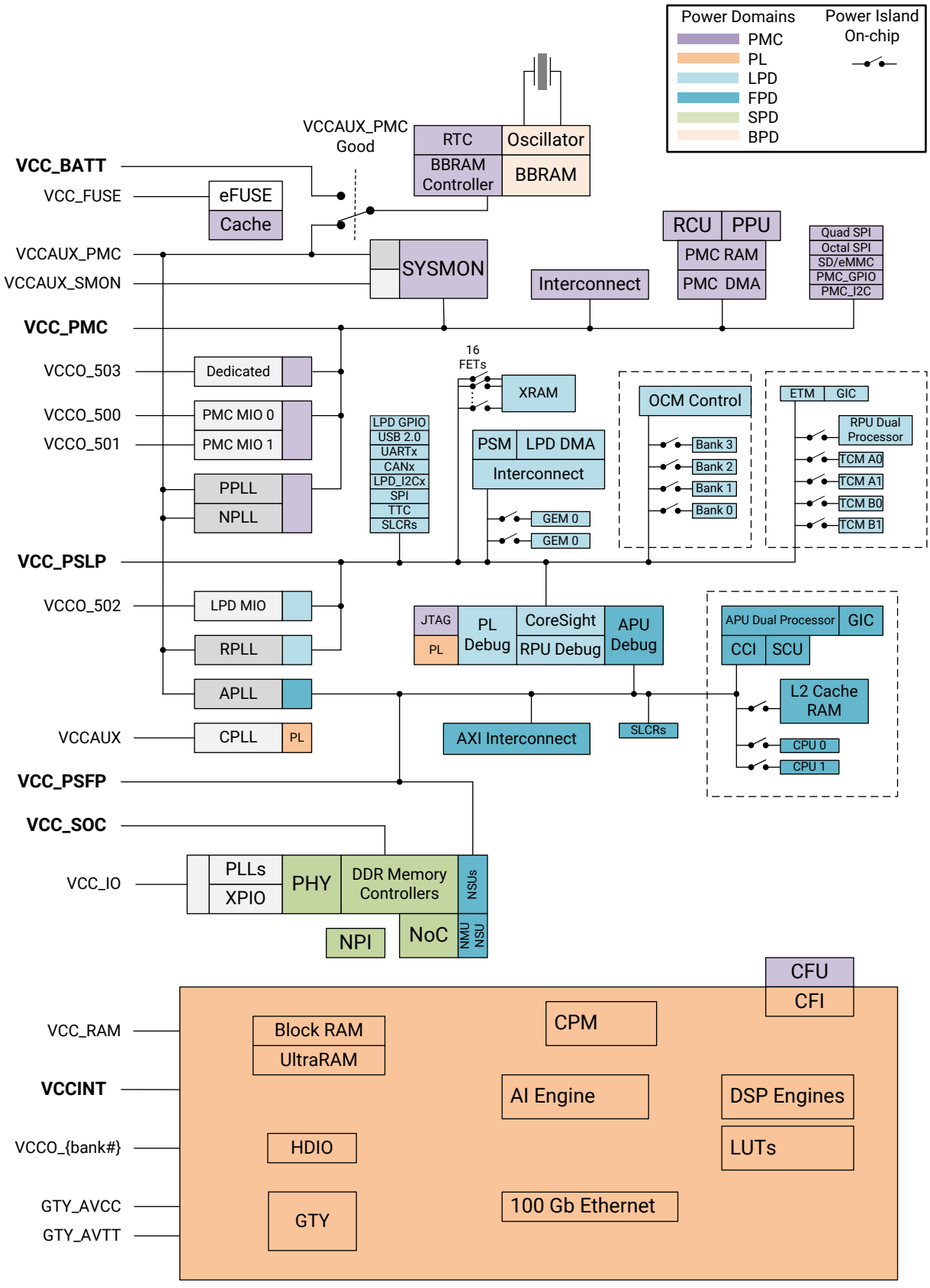
There are several power reduction features. In addition to power control, the processors have sleep modes. The power domains are controlled by output pins attached to external power supplies. The PS power islands are controlled by on-chip power FETs that are controlled by registers accessible to the PSM.

- Processor sleep/wake feature
- Clock frequency reduction

Power Diagram

The following figure shows the Versal™ ACAP power domains. The power domains are color coded. The power islands are shown with small switch symbols.

Figure 103: **Power Domains and Islands Diagram**



Power Domains

The primary power domains are listed in Power Management.

Power Islands

There are many power islands in the LPD and FPD. These are shown in the Power Diagram. System software makes power-up and power-down requests to the PSM registers as shown in PSM Service Requests.

XILINX

Section XV

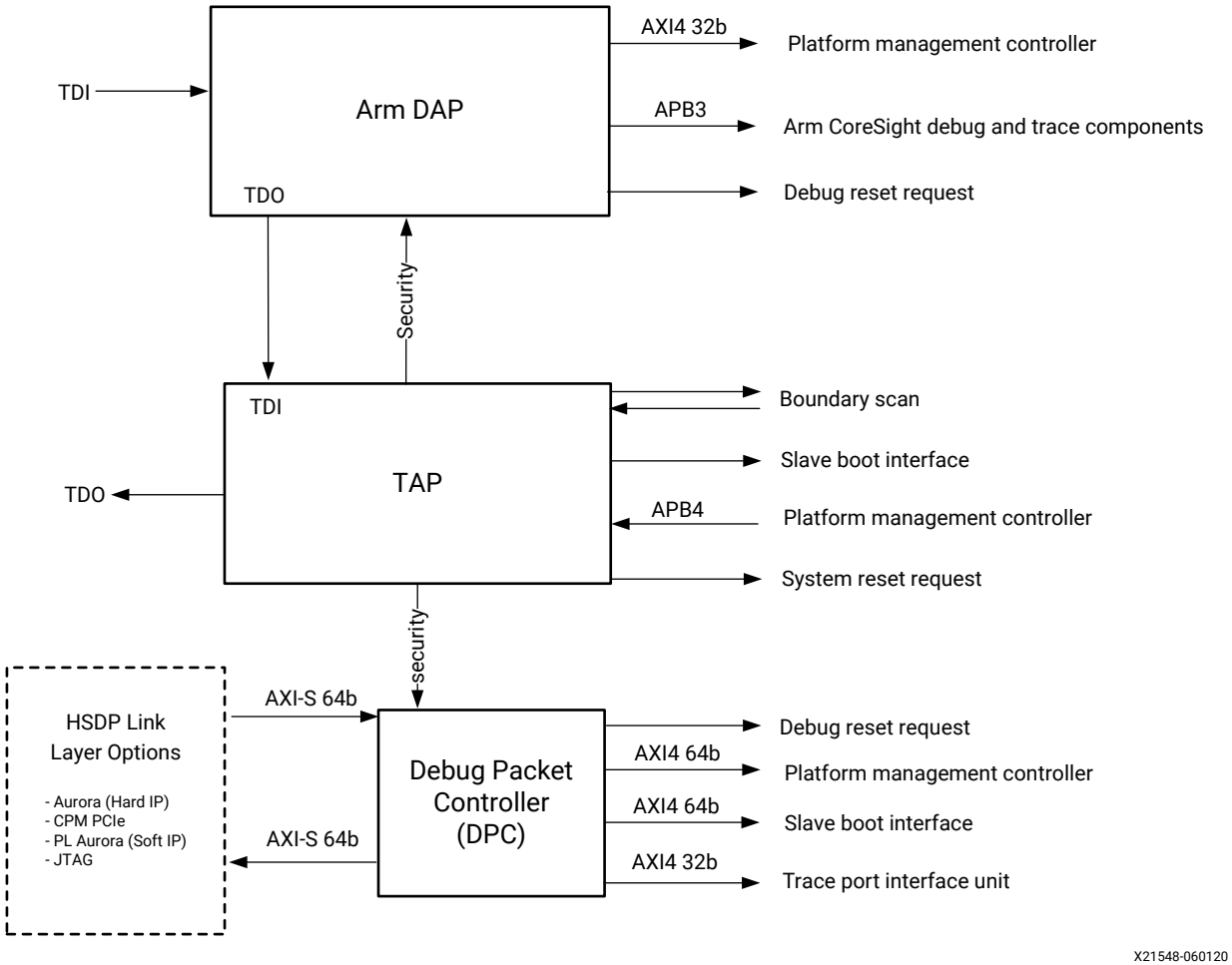
Test and Debug

This section includes these chapters:

Integrated Debug

The Versal™ ACAP has integrated debug that resides in the PMC. The integrated debug subsystem includes the test access port (TAP) controller, the Arm® debug access port (DAP) controller, and the debug packet controller (DPC). The PMC TAP controller supports PL configuration, ChipScope™ debug, and JTAG boundary-scan operations. The Arm DAP controller supports the Arm CoreSight™ debug and trace. The DPC is part of the HSDP solution and allows access to all debug resources including Arm CoreSight debug and trace and ChipScope.

Figure 104: **Debug Interface Block**

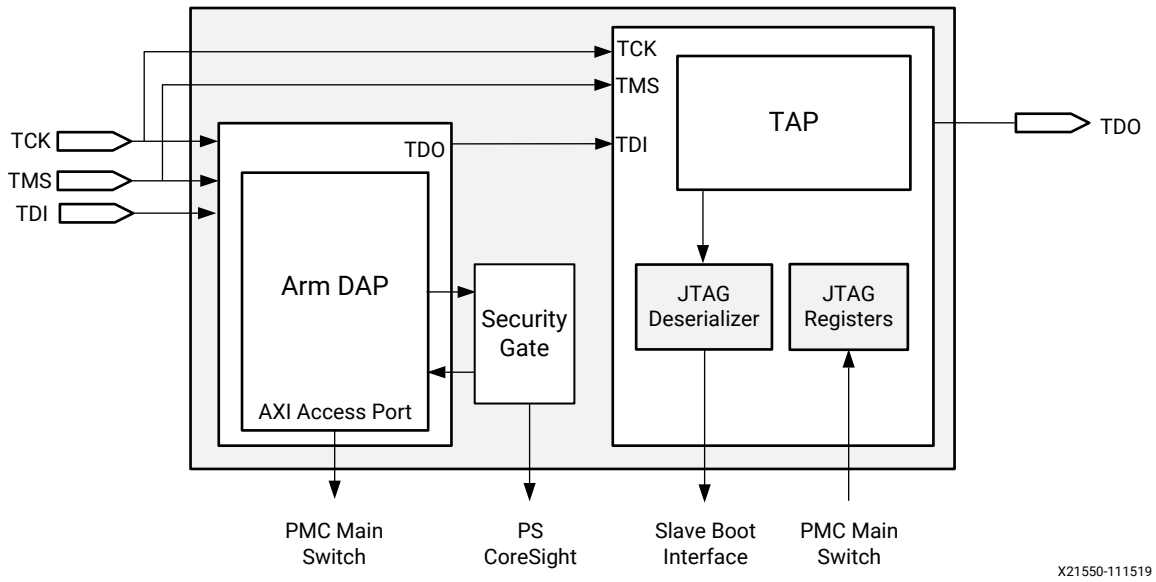


JTAG and Boundary-Scan

The Versal ACAP architecture is compatible with the IEEE Standard Test Access Port and Boundary-Scan Architecture (IEEE Std 1149.1) and includes all the mandatory elements defined by the standard. These elements include the TAP, TAP controller, instruction register, instruction decoder, boundary register, and bypass register. Versal ACAPs also support a 32-bit device identification register and a JTAG configuration register that adds additional readback and configuration JTAG capability.

The primary debug access port is the JTAG interface. The JTAG chain order in the Versal ACAP is fixed with the DAP controller followed by the TAP controller as shown in the following figure.

Figure 105: **JTAG Chain**



The JTAG dedicated I/O supports boundary-scan operations, status register access, PL readback, and a single-stepping hardware analyzer in the PL and AI Engine. The JTAG interface provides base debug to assist with board or device bring-up issue isolation.

On power-up the default boot mode is secure and the JTAG interface accepts the standard boundary-scan instructions and JTAG instructions listed regardless of the boot mode: BYPASS, ERROR_STATUS, EXTENDED_IDCODE, EXTEST, EXTEST_PULSE, EXTEST_TRAIN, HIGHZ_IO, IDCODE, READ_DNA, SAMPLE/PRELOAD, SEC_DBG, STATUS, and USERCODE. For non-secure boots, after the boot is complete, successfully or unsuccessfully, the full suite of JTAG instructions are enabled. For secure boots, if the boot is completed successfully, the authenticated software is capable of enabling the additional JTAG commands. In the event of a failed secure boot, the JTAG capabilities are dependent on how the device was provisioned.

TAP Controller

Test Access Port Interface

The device test access port (TAP) contains the four mandatory, dedicated pins (TDI, TDO, TMS, and TCK) as specified by the protocol. Test reset (TRST) and enable pins are optional control pins sometimes used by devices from other manufacturers, but are not provided on Xilinx devices.

IMPORTANT! *Be aware of optional signals when interfacing Xilinx devices with parts from different vendors, because driving these optional pins could have different requirements.*

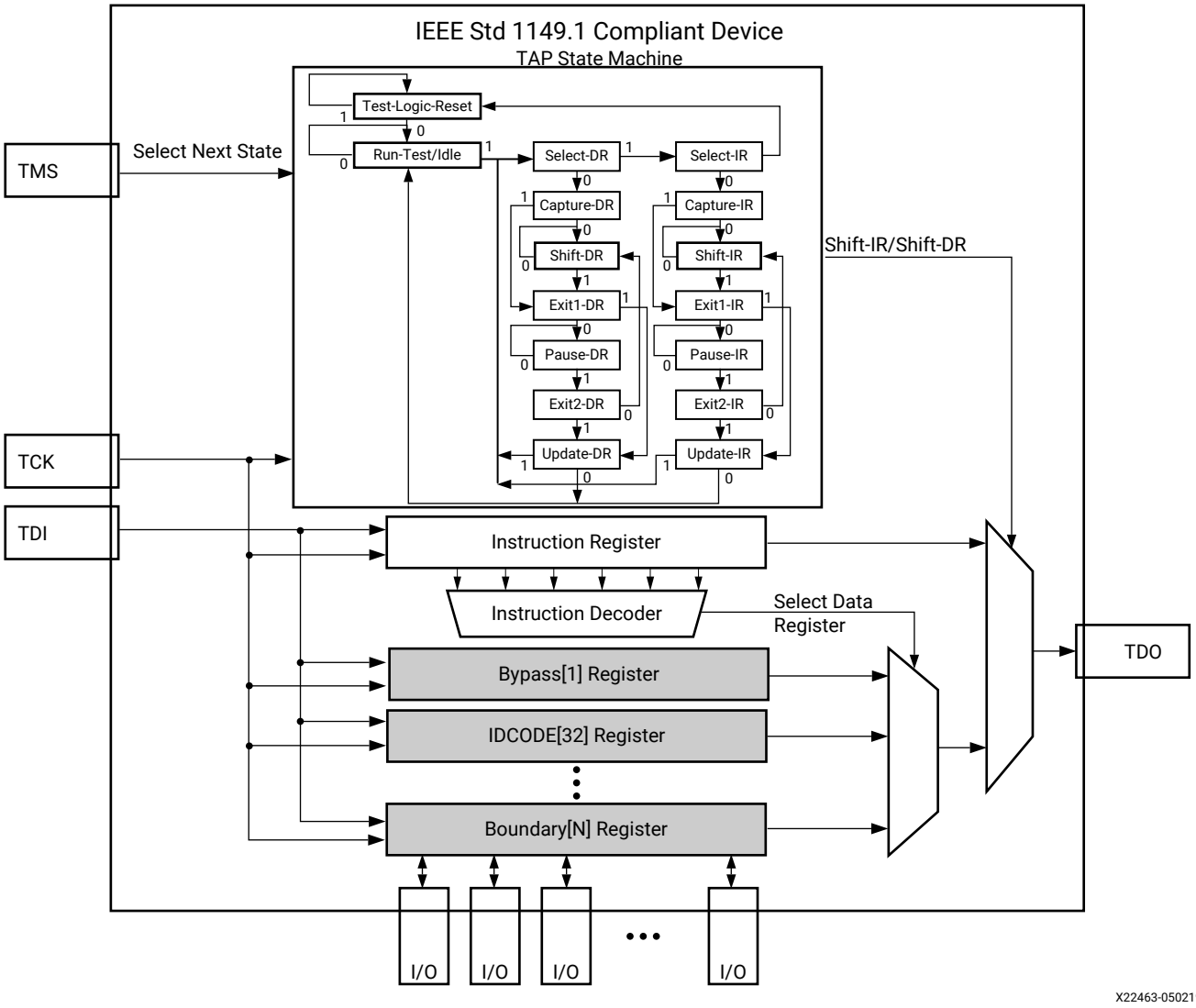
Table 238: **TAP Interface**

Pin	Type	Direction	Description
TDI	Dedicated	Input	Test data in (TDI): This pin is the serial input to all JTAG instruction and data registers. The state of the TAP controller and the current instruction determine the register that is fed by the TDI pin for a specific operation. TDI has an internal resistive pull-up to provide a logic High to the system if the pin is not driven. TDI is applied to the JTAG registers on the rising edge of TCK.
TDO	Dedicated	Output	Test data out (TDO): This pin is the serial output for all JTAG instruction and data registers. The state of the TAP controller and the current instruction determine the register (instruction or data) that feeds TDO for a specific operation. TDO changes state on the falling edge of TCK and is only active during the shifting of instructions or data through the device. TDO is an active driver output. TDO has an internal resistive pull-up to provide a logic High if the pin is not active.
TMS	Dedicated	Input	Test mode select (TMS): This pin determines the sequence of states through the TAP controller, which change on the rising edge of TCK. TMS has an internal resistive pull-up to provide a logic High if the pin is not driven.
TCK	Dedicated	Output	Test clock (TCK): This pin is the JTAG test clock. TCK sequences the TAP controller and the JTAG registers. TCK has an internal resistive pull-up to provide a logic High if the pin is not driven.

Test Access Port (TAP) Controller

The following figure shows the JTAG standard 16-state finite state machine. The four TAP pins control how data is scanned into the various registers. The state of the TMS pin at the rising edge of TCK determines the sequence of state transitions. There are two main sequences, one for shifting data into the data register and the other for shifting an instruction into the instruction register. A transition between the states only occurs on the rising edge of TCK, and each state has a different name. The two vertical columns with seven states each represent the instruction path and the datapath. The data registers operate in the states whose names end with "DR," and the instruction register operates in the states whose names end in "IR." The states are otherwise identical.

Figure 106: **Example JTAG TAP Controller and TAP Registers**



JTAG Register Reference

The Versal™ ACAPs provide JTAG registers that can be accessed through the JTAG interface for boundary-scan operations and debug. Several of the JTAG registers provide valuable status indicators for the device start-up and boot. The JTAG TAP registers in the Versal ACAP are listed in the following table

Table 239: **JTAG Registers**

Register Name	Register Length	Description
BOUNDARY	Varies per device	Controls and observes input, output, and output enable
BYPASS	1-bit	Bypasses the device

Table 239: **JTAG Registers** *(cont'd)*

Register Name	Register Length	Description
ERROR_STATUS	160-bit	Captures the error management status for the PMC
EXTENDED_IDCODE	32-bit	Captures the device extended IDCODE
DEVICE_IDENTIFICATION (IDCODE)	32-bit	Captures the device IDCODE
INSTRUCTION	6-bit	• Holds the current instruction opcode and captures internal device status • Total Versal ACAP instruction register length is DAP (4-bit instruction register length) + TAP (6-bit instruction register length)
JTAG_CONFIG	Varies	• Connects the JTAG pins to the slave boot interface when using the JTAG TAP instructions JCONFIG and JRDBK • These JTAG TAP instructions write or read a programmable device image into the Versal ACAP
DNA	128-bit	Captures the device DNA value
SECURE_DEBUG	32-bit	Shifts in the authenticated data packet to authenticate in secure mode
STATUS	36-bit	Captures the platform management controller overall status
SYSTEM_RESET	1-bit	Bypasses the device
USER_DEFINED (USER1, USER2, USER3, USER4)	Design specific	Design-specific register
USERCODE	32-bit	Captures the user-designated value

ERROR_STATUS Register

The ERROR_STATUS register is a 160-bit register that provides key error conditions from across the Versal ACAP. The ERROR_STATUS register is accessed via JTAG.

Table 240: **ERROR_STATUS Register Format**

Bit	Field	Description
159:155	RSVD_READS_0	reserved, returns 0
154:148	RESERVED	reserved
147:136	BOOTROM FIRST ERROR	BootROM first error code detected
135:124	BOOTROM LAST ERROR	BootROM last error code detected
123:110	PLM MAJOR ERROR	PLM major error code
109:94	PLM MINOR ERROR	PLM minor error code
93:64	GSW ERROR	General software error code for PLM
63	RESERVED	reserved
62	BOOTROM NCR	BootROM non-correctable error Set by RCU BootROM during boot
61	PLM CR	Platform loader and manager boot correctable error Set by PLM during boot

Table 240: **ERROR_STATUS Register Format** *(cont'd)*

Bit	Field	Description
60	PLM NCR	Platform loader and manager boot non-correctable error Set by PLM during boot
59	GSW CR	General software correctable error after boot
58	GSW NCR	General software non-correctable error after boot
57	CFU ERROR	CFU error
56	CFRAME ERROR	CFRAME error
55	PSM CR	PSM correctable error
54	PSM NCR	PSM non-correctable error
53	DDRMC MB CR	DDRMC MicroBlaze correctable ECC error
52	DDRMC MB NCR	DDRMC MicroBlaze non-correctable ECC error
51	NOC CR	NoC correctable error
50	NOC NCR	NoC non-correctable error
49	NOC USER ERROR	NoC user error
48	MMCM LOCK ERROR	MMCM lock error
47	AIE CR	AI Engine correctable error
46	AIE NCR	AI Engine non-correctable error
45	DDRMC MC ECC CR	DDRMC MC (memory controller) correctable ECC error
44	DDRMC MC ECC NCR	DDRMC MC (memory controller) non-correctable ECC error
43	GT CR	GT correctable error
42	GT NCR	GT non-correctable error
41	SYSMON CR	SYSMON correctable error
40	SYSMON NCR	SYSMON non-correctable error
39	USER PL0 ERROR	User defined PL error
38	USER PL1 ERROR	User defined PL error
37	USER PL2 ERROR	User-defined PL error
36	USER PL3 ERROR	User-defined PL error
35	NPI ROOT ERROR	NPI root error
34	SSIT ERROR3	SSI technology SLR error
33	SSIT ERROR4	SSI technology SLR error
32	SSIT ERROR5	SSI technology SLR error
31	PMC APB ERROR	PMC APB error. Includes errors from registers: PMC_LOCAL, PMC_GLOBAL, CRP, PMC_IOP_SECURE_SLCR, PMC_IOP, BBRAM_CTRL, PMC_ANLG, RTC
30	PMC BOOTROM ERROR	PMC BootROM validation error
29	RCU HARDWARE ERROR	RCU hardware error
28	PPU HARDWARE ERROR	PPU hardware error
27	PMC PAR ERROR	PMC switch and PMC IOP parity errors
26	PMC CR	PMC correctable errors
25	PMC NCR	PMC non-correctable errors

Table 240: **ERROR_STATUS Register Format** *(cont'd)*

Bit	Field	Description
24	PMC SYSMON0 ALARM	PMC temperature shutdown alert and power supply failure detection errors from SYSMON Collected from PMC_ANLG sysmon_alarm[0]
23	PMC SYSMON1 ALARM	PMC temperature shutdown alert and power supply failure detection errors from SYSMON Collected from PMC_ANLG sysmon_alarm[1]
22	PMC SYSMON2 ALARM	PMC temperature shutdown alert and power supply failure detection errors from SYSMON Collected from PMC_ANLG sysmon_alarm[2]
21	PMC SYSMON3 ALARM	PMC temperature shutdown alert and power supply failure detection errors from SYSMON Collected from PMC_ANLG sysmon_alarm[3]
20	PMC SYSMON4 ALARM	PMC temperature shutdown alert and power supply failure detection errors from SYSMON Collected from PMC_ANLG sysmon_alarm[4]
19	PMC SYSMON5 ALARM	PMC temperature shutdown alert and power supply failure detection errors from SYSMON Collected from PMC_ANLG sysmon_alarm[5]
18	PMC SYSMON6 ALARM	PMC temperature shutdown alert and power supply failure detection errors from SYSMON Collected from PMC_ANLG sysmon_alarm[6]
17	PMC SYSMON7 ALARM	PMC temperature shutdown alert and power supply failure detection errors from SYSMON Collected from PMC_ANLG sysmon_alarm[7]
16	PMC SYSMON8 ALARM	PMC temperature shutdown alert and power supply failure detection errors from SYSMON Collected from PMC ANLG sysmon_alarm[8]
15	PMC SYSMON9 ALARM	PMC temperature shutdown alert and power supply failure detection errors from SYSMON Collected from PMC_ANLG sysmon_alarm[9]
14	CFI NCR	CFI non-correctable error
13	SEU CRC ERROR	SEU CRC error
12	SEU ECC ERROR	SEU ECC error
11:10	RSVD_READS_0	reserved, returns 0
9	RTC ALARM	RTC alarm error
8	NPLL ERROR	PMC NPLL lock error
7	PPLL ERROR	PMC PPLL lock error
6	CLOCK MONITOR ERROR	Clock monitor errors
5	PMC TIMEOUT ERROR	PMC interconnect timeout errors from interconnect mission interrupt status register, interconnect latent status register, and timeout interrupt status register
4	PMC XMPU ERROR	PMC XMPU errors from register access error on APB. Includes read permission violation, write permission violation or security violation
3	PMC XPPU ERROR	PMC XPPU errors from register access error on APB. Includes Master ID not found, read permission violation, Master ID access violation, Master ID parity error, TrustZone violation
2	SSIT ERROR0	SSI technology SLR error

Table 240: **ERROR_STATUS Register Format** *(cont'd)*

Bit	Field	Description
1	SSIT ERROR1	SSI technology SLR error
0	SSIT ERROR2	SSI technology SLR error

EXTENDED_IDCODE Register

The EXTENDED_IDCODE register is a 32-bit register that is accessed via JTAG. The EXTENDED_IDCODE register provides the extended device family code that is used with the IDCODE register for device identification. See Device Identification for the Versal ACAP member information.

Table 241: **EXTENDED_IDCODE Register Format**

Bit	Field	Description
31:28	RESERVED	Reserved
27:14	EXTENDED FAMILY CODE	Versal device family code extension
13:0	RESERVED	Reserved

IDCODE Register

The Device_Identification (IDCODE) register is a 32-bit register accessed via JTAG. The IDCODE register provides base family device and revision information with the manufacturer code. The IDCODE register and the EXTENDED_IDCODE register provide the device identification. See Device Identification for the Versal ACAP member information.

Table 242: **IDCODE Register Format**

Bit	Field	Description
31:28	VERSION CODE	4-bit version code Used to identify production device
27:12	BASE FAMILY CODE	16-bit base family code
11:1	MANUFACTURER CODE	11-bit Xilinx manufacturer's code is `11b'00001001001`
0	RSVD_READS_1	As specified by the IEEE Std 1149.1, this bit is always 1

DNA Register

The DNA register is a 128-bit register accessed via JTAG. The DNA register provides the unique device identifier. See Device Identification for the Versal ACAP member information.

Table 243: **DNA Register Format**

Bit	Field	Description
127	RSVD_READS_1	As specified by the Xilinx DNA, this bit is always 1
126	RSVD_READS_1	As specified by the Xilinx DNA, this bit is always 1
125:2	DNA	124-bit DNA represents the specific Versal ACAP unique device identifier information
1	RSVD_READS_1	As specified by the Xilinx DNA, this bit is always 1
0	RSVD_READS_0	As specified by the Xilinx DNA, this bit is always 0

STATUS Register

The Versal ACAP STATUS register is 36-bits and provides key device information. The register includes the selected boot mode, critical voltage supplies detection, bus width detection, and security feature status. The STATUS register format is shown in the following table.

Table 244: **STATUS Register Format**

Bit	Field	Description
35	RESERVED	Reserved
34	DONE	Boot and configuration status indicator A value of 1 on DONE indicates boot and configuration is complete
33	JRDBK ERROR	JTAG readback status indicator A value of 1 on JRDBK indicates an error reading data from SBI
32	JCONFIG ERROR	JTAG data load error indicator A value of 1 means the SBI is not ready to accept data
31:28	PMC VERSION	PMC version
27:24	RESERVED	Reserved
23	JTAG SEC GATE	Security gate status A value of 1 means DAP AXI transactions are allowed
22	RESERVED	Reserved
21	PMC SCAN CLEAR DONE	Scan clear done indication A value of 1 means the scan clear is complete
20	PMC SCAN CLEAR PASS	Scan clear pass indication A value of 1 means the scan clear passed
19:16	RESERVED	Reserved
15:12	BOOT MODE [3:0]	Boot mode value captured from the MODE pins at release of POR_b
11	VCC_PMC DETECTED	VCC_PMC supply detected
10	VCC_PSLP DETECTED	VCC_PSLP supply detected
9	VCCINT DETECTED	VCCINT supply detected
8	VCC_SOC DETECTED	VCC_SOC supply detected
7	AES KEY ZEROIZED	AES key zeroized indicator A value of 1 indicates all keys are zeroized

Table 244: **STATUS Register Format** *(cont'd)*

Bit	Field	Description
6	BBRAM KEY ZEROIZED	BBRAM key zeroized indicator A value of 1 indicates that the BBRAM key is zeroized
[5:4]	SELECTMAP BUS WIDTH	SelectMAP boot mode bus width detected 00 = No bus width detected 01 = SelectMAP 8-bit 10 = SelectMAP 16-bit 11 = SelectMAP 32-bit
3	SBI JTAG ENABLED	SBI JTAG indicator A value of 1 indicates the SBI is configured to receive data from the JTAG interface
2	SBI JTAG BUSY	SBI JTAG BUSY indicator A value of 1 indicates the SBI is BUSY and cannot accept data when in JTAG mode
1	RSVD_READS_0	Reserved, returns 0
0	RSVD_READS_1	Reserved, returns 1

TAP Instructions

The TAP instructions supported by the Versal ACAP are listed in the following table.

Table 245: **TAP Instructions**

Instruction Name	Binary Code [5:0]	Description
BYPASS	111111	Enables BYPASS instruction
JCONFIG	000101	Accesses the slave boot interface for boot and configuration of the Versal ACAP via JTAG
JRDBK	000100	Accesses the slave boot interface for readback via JTAG
DPC	110110	Accesses the debug packet controller
ERROR_STATUS	111110	Accesses the error management status register
EXTENDED_IDCODE	011001	Used with IDCODE for extended device identification
EXTEST	100110	Enables the boundary-scan EXTEST instruction
EXTEST_PULSE	111100	Enables the IEEE Std 1149.6 functions in the GTs for testing AC-coupled connections between GTs
EXTEST_TRAIN	111101	Enables the IEEE Std 1149.6 functions in the GTs for testing AC-coupled connections between GTs
HIGHZ_IO	001010	3-stated user I/O pins but not GTs while enabling the bypass register
IDCODE	001001	Accesses the Versal ACAP IDCODE
READ_DNA	110010	Accesses the Versal ACAP unique device DNA value
SAMPLE/PRELOAD	000001	Enables boundary-scan SAMPLE/PRELOAD instruction
SEC_DBG	110101	Secure debug disables the JTAG interface
STATUS	011111	Access to the Versal ACAP status register value
SYS_RST	110111	Resets the Versal ACAP via JTAG

Table 245: **TAP Instructions** *(cont'd)*

Instruction Name	Binary Code [5:0]	Description
USER1	000010	Access to the user-defined register 1
USER2	000011	Access to the user-defined register 2
USER3	100010	Access to the user-defined register 3
USER4	100011	Access to the user-defined register 4
USERCODE	001000	Access to the user designated value

Arm DAP Controller

The Arm debug access port (DAP) controller uses the Arm debug interface version 5 (ADIv5). The debug port is used to access the DAP from an external debugger, and there are access port components to access on-chip system resources. The DAP is referred to debug port and access ports. The DAP controller supports the following features:

- Central controller for the CoreSight debug and trace components with the PS
- Interface to the Arm debug tools through the JTAG interface
- Invasive and non-invasive debug control
- Secure and non-secure debug support

Arm DAP Registers

The Versal ACAPs provide Arm DAP registers listed in the following table.

Table 246: **Arm DAP Registers**

Register Name	Register Length	Description
BYPASS	1-bit	Bypasses the Arm DAP
IDCODE	32-bit	Captures the Arm DAP IDCODE (5BA00477h)
INSTRUCTION	4-bit	Holds the current instruction opcode. The total Versal ACAP instruction register length is the DAP (4-bit instruction register length) + TAP (6-bit instruction register length).

Arm DAP Instructions

The Versal ACAPs support the Arm DAP instructions listed in the following table.

Table 247: **Arm DAP Instructions**

Instruction Name	Binary Code [3:0]	Description
BYPASS	1111	Accesses the Arm JTAG debug port Bypass register
IDCODE	1110	Accesses the Arm JTAG debug port IDCODE register
ABORT	1000	Accesses the Arm JTAG debug port Abort register
DPACC	1010	Accesses the JTAG debug port DP register
APACC	1011	Accesses the JTAG debug port AP register

High-Speed Debug Port

The high-speed debug port (HSDP) provides a unified debug and trace capability for programmable logic, processing system, and the AI Engines. The HSDP leverages gigabit transceivers from various sources in the device to make debug less intrusive to the system configuration.

The HSDP debug packet controller (DPC) can be accessed by the host through a high-speed cable or PCIe to perform high-speed transactions on behalf of the host. The solution also supports at-speed debug of PL designs through ChipScope™.

The HSDP has two layers, a transaction layer and a link layer. The PMC DPC is the transaction layer that processes the packets. The link layer is the interface to communicate to the DPC, with four options:

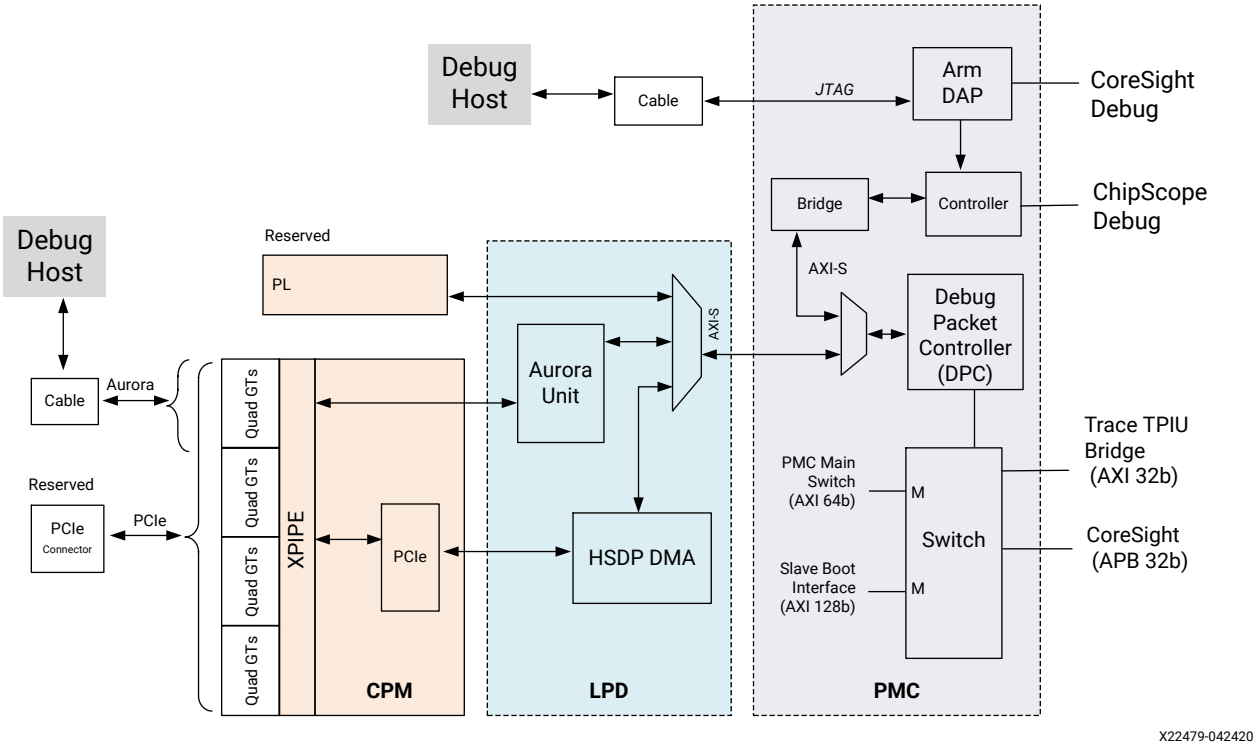
- JTAG interface
- Hardened Aurora via XPIPE and gigabit transceivers
- PCIe via XPIPE, gigabit transceivers, and CPM PCIe block
- PL fabric via AXI4-Stream into PL to enable protocols like soft Aurora

The packets processed by DPC are referred to as debug and trace packets (DTP). Each packet consists of a header, payload, and CRC fields. The DPC decodes the payloads to determine the commands, destinations, and any required higher level flow control and management tasks. DPC processes the DTP sent by a host, executes commands embedded in them and generates responses including errors sent back to the host.

Host Debug Ports

The following figure provides a view of the JTAG and HSDP interfaces.

Figure 107: **Host Debug Port Interfaces**



Debug Packet Controller

The debug packet controller (DPC) is included in the PMC. The DPC is the central part of the high-speed debug port solution (HSDP). The HSDP defined protocol is separated into two layers, the transaction layer with the DPC, and the link layer with Aurora, PCI Express, JTAG, or PL interfaces. The DPC receives and executes input packets from one of the supported interfaces. The DPC then generates reply packets and transmits them to the interface. The DPC is divided into the following blocks:

- FIFO captures the input stream from AXI
- Demultiplexer and decode identifies the packet boundaries and decodes non-queued and queued packets
- Command buffers to buffer the queued command packets
- Processing engine to process the queued packets and interfaces with the external switch
- Reply buffers generate reply packets that are waiting to be transmitted to the output to the AXI

Device Identification

The Versal™ ACAP has multiple device identification methods:

- Top package marking
- IDCODE + EXTENDED_IDCODE register value (see Versal ACAP data sheets)
- Device DNA register value

Security features such as PUF can also be used to create a unique identifier. For more information on using PUF, see Security Management.

Package Marking

The Versal ACAPs have a top package marking that includes the Versal family name and a 2D barcode for device-level tracking. The 2D barcode information can be accessed several ways including with the Xilinx web-based tool or the Xilinx GO mobile application. For more information, see the *Versal ACAP Packaging and Pinouts Architecture Manual* (AM013).

IDCODE Introduction

The Versal ACAP has a 32-bit identification stored in the IDCODE register. The IDCODE is a fixed, vendor-assigned value used to electrically identify the manufacturer. The IDCODE used in conjunction with the EXTENDED_IDCODE can also identify the type of Xilinx device.

The IDCODE register can be read via the JTAG interface or from the AXI interface using the IDCODE register address. When the IDCODE instruction is selected by the JTAG TAP, the IDCODE register is connected between the JTAG TDI and TDO pins, and the value can be shifted out through TDO for examination with tools such as the Vivado design suite. The least significant bit of the IDCODE register is always `1` (based on the JTAG IEEE Std 1149.1).

Table 248: **IDCODE Register**

Register Type	Register Name	Address	Description
Read only	IDCODE[31:0]	`0xF11A_0000`	Versal ACAP base family code The IDCODE register with the EXTENDED_IDCODE register provides the device identification See IDCODE Register for bit field details

EXTENDED_IDCODE Introduction

The Versal ACAP has a 32-bit device extended family code that is stored in the EXTENDED_IDCODE register. The EXTENDED_IDCODE vendor-assigned value is used with the IDCODE to identify the Xilinx device.

The EXTENDED_IDCODE register value can be read via the JTAG interface or from the AXI interface using the EXTENDED_IDCODE register address. When the EXTENDED_IDCODE instruction is selected by the JTAG TAP, the EXTENDED_IDCODE register is connected between the JTAG TDI and TDO pins, and the value can be shifted out through TDO for examination with tools such as the Vivado design suite.

Table 249: **EXTENDED_IDCODE Register**

Register Type	Register Name	Address	Description
Read only	EXTENDED_IDCODE[31:0]	0xF125_0018	Versal ACAP extended family code The Versal ACAP EXTENDED_IDCODE register with the IDCODE register provides the device identification See EXTENDED_IDCODE Register for bit field details

DNA Introduction

The device DNA is a single unique 128-bit factory-programmed identifier for each device. The JTAG TAP instruction, READ_DNA, reads the DNA value through the JTAG interface. The device DNA value can also be read from the AXI interface using the combined value from the DNA_0, DNA_1, DNA_2, and DNA_3 registers. The power-on reset (POR_b) pin must be released before reading the DNA unique device identifier. If the POR_b pin is not released, the DNA value is 0.

Table 250: **DNA Registers**

Register Type	Register	Address (Hex)	Description
Read only	DNA	NA	DNA register is accessed via the JTAG instruction READ_DNA and contains the DNA[127:0] See DNA Register for bit field details
Read only	DNA_0	0xF125_0020	DNA_0 register contains DNA bits[31:0]
Read only	DNA_1	0xF125_0024	DNA_1 register contains DNA bits[63:32]
Read only	DNA_2	0xF125_0028	DNA_2 register contains DNA bits[95:64]
Read only	DNA_3	0xF125_002C	DNA_3 register contains DNA bits[127:96]

CoreSight Debug

The CoreSight™ debug functionality is controlled by the DAP controller on the JTAG chain.

The width of the CoreSight output trace data bus can be 1, 2, 4, 8, or 16 bits wide. Higher bandwidth output can be obtained using the high-speed data port (HSDP).

Trace Port I/O Signals

The trace I/O signals have two pinout options in the PMC MIO multiplexer. The trace inteface is also available on the EMIO to the PL. The trace I/O signals for CoreSight™ are listed in the following table.

Table 251: **Trace I/O Signals for CoreSight**

MIO					EMIO	
Signal Name	I/O	PMC Multiplexer Pins		MIO-at-a-Glance Number	Signal Name	I/O
		A	B			
TRACE_CLK	O	6	32	0		O
TRACE_CTRL	O	4	30	1		O
TRACE_DATA[0]	O	5	31	2		O
TRACE_DATA[1]	O	7	33	3		O
TRACE_DATA[2:3]	O	8, 9	34, 35	4, 5		O
TRACE_DATA[4:7]	O	10:13	36:39	6:9		O
TRACE_DATA[8:15]	O	14:21	40:47	10:17		O

XILINX

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help → Documentation and Tutorials**.
- On Windows, select **Start → All Programs → Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

***Note*:** For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this guide:

1. Versal ACAP product data sheet:
   - *Versal Architecture and Product Data Sheet: Overview* (DS950)
2. Versal ACAP electrical data sheets:
   - *Versal Prime Series Data Sheet: DC and AC Switching Characteristics* (DS956)
   - *Versal AI Core Series Data Sheet: DC and AC Switching Characteristics* (DS957)
3. *Versal ACAP GTY Transceivers Architecture Manual* (AM002)
4. *Versal ACAP Clocking Resources Architecture Manual* (AM003)
5. *Versal ACAP DSP Engine Architecture Manual* (AM004)
6. *Versal ACAP Configurable Logic Block Architecture Manual* (AM005)
7. *Versal ACAP System Monitor Architecture Manual* (AM006)
8. *Versal ACAP Memory Resources Architecture Manual* (AM007)
9. *Versal ACAP AI Engine Architecture Manual* (AM009)
10. *Versal ACAP CPM CCIX Architecture Manual* (AM016)
11. *Versal ACAP Design Guide* (UG1273)
12. *Bootgen User Guide* (UG1283)
13. *Versal ACAP System Software Developers Guide* (UG1304)
14. *Xilinx AI Engine and Their Applications* (WP506)
15. *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* (PG313)
16. *Versal ACAP CPM Mode for PCI Express Product Guide* (PG346)

Additional References

1. Recommendation for Applications Using Approved Hash Algorithms NIST Special Publication 800-107
2. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, NIST FIPS PUB 202

Arm References

1. Cortex™-R5F Technical Reference Manual
2. Cortex-A72 Crypto Technical Reference Manual
3. Cortex-A53 MPCore Processor Technical Reference Manual
4. Arm® Architecture Reference Manual Arm v8

IP versions are listed in the IP Versions chapter.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

This document contains preliminary information and is subject to change without notice. Information provided herein relates to products and/or services not yet available for sale, and provided solely for information purposes and are not intended, or to be construed, as an offer for sale or an attempted commercialization of the products and/or services referred to herein.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright